Karlsruhe Institute of Technology

FONDAZIONE BRUNO KESSLER

Personalized News Event Retrieval for Small Talk in Social Dialog Systems

Master's Thesis of

Lucas Bechberger

at the Department of Informatics
Institute for Anthropomatics and Robotics

Reviewer:	Prof. Dr. Alexander Waibel
Second reviewer:	Dr. Sebastian Stüker
Advisor:	M.A. Maria Schmidt
Second advisor:	Prof. Dr. Marcello Federico (FBK)

20 October 2015 – 19 April 2016

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, den 19. April 2016

. . . . . .

(Lucas Bechberger)

Acknowledgments

This thesis was written at Fondazione Bruno Kessler (FBK) in Trento (Italy) as part of the InterACT exchange program funded with an IGEL ("**I**nformatik **GrE**nzen**L**os") scholarship which was provided by the "Begabtenkolleg Informatik" of the Department of Computer Science at KIT. I would like to thank all people who made my stay in Italy possible by supporting these programs, and especially Margit Rödder for her support with respect to all administrative questions.

I would like to thank professor Alex Waibel for supporting my idea of writing my thesis in Italy from the very beginning and for making my stay at FBK possible. I would also like to thank Marcello Federico, head of the HLT-MT group at FBK, for giving me a warm welcome, for supervising my work, and for giving me valuable feedback during the past six months. Many thanks also to Marco Rospocher and Anne-Lyse Minard who acted as my advisors with respect to the KnowledgeStore and NewsReader systems, respectively. Moreover, I would also like to thank my supervisor Maria Schmidt who always gave me the freedom to explore my own ideas and approaches and who provided me with guidance whenever I needed it.

Also many thanks to my colleagues at FBK for the warm welcome, the nice atmosphere, and the fruitful discussions. Finally, I would like to thank the eleven annotators who helped me to create the data set for the ranking problem, as well as everyone who participated in the user study for evaluating my system.

Abstract

This thesis explores the area of personalized news event retrieval in the context of social dialog systems.

We developed the NewsTeller system which retrieves a relevant news event based on a user query and the user's general interests (both represented as list of keywords). The retrieved news event can then be used by a social dialog system to initiate news-related small talk.

The NewsTeller system is implemented as a pipeline with four stages:

In a first step, a (large) set of potentially relevant news events is retrieved.

As about 84% of the found events do not fulfill our syntactic and semantic well-formedness criteria, the second step in the pipeline is concerned with filtering the found events. This filtering is done by a classifier which was trained on a data set of about 6,000 events. The results obtained in a ten-fold cross-validation indicate that a global random forest classifier is superior to an ensemble of specialized classifiers that were trained on specific subproblems. The global classifier reaches a precision of 63.04% and a recall of 59.55%.

The third step in the pipeline is concerned with ranking the remaining events according to their expected relevance and selecting the event with the highest expected relevance. Four ordered classes are used to describe an event's relevance: Irrelevant, Partially Relevant, Relevant, and Very Relevant. Following the "learning to rank" approach from information retrieval, this task is framed as a regression problem on the relevance values of the events which is solved by training a random forest regressor on a data set of about 3,200 events. Two different approaches were compared: using only features defined on the events and the user query and using also features defined on the user's general interests. The results in a user-level leave-one-out evaluation indicate that both regressors have comparable performance in avoiding Irrelevant events and that taking into account the user's interests helps to improve the detection of Relevant and Very Relevant events.

In the fourth step of the pipeline, a summary of the selected news event is created. This is done by extracting the sentence in which the event was mentioned.

For evaluating the system, a user study with 48 participants was conducted. The results of this evaluation show that the two regression-based approaches are significantly better than a random baseline with respect to avoiding Irrelevant events. We could however not confirm our hypothesis that using information about the users' general interests helps to improve the detection of Relevant and Very Relevant events.

Zusammenfassung

Diese Masterarbeit beschäftigt sich mit dem personalisierten Retrieval von News Events im Kontext sozialer Dialogsysteme.

Wir entwickelten das NewsTeller System, das basiered auf einer Nutzeranfrage sowie den generellen Interessen des Nutzers (beide als Listen von Keywords repräsentiert) ein relevantes News Event findet. Dieses News Event kann dann von einem sozialen Dialogsystem verwendet werden, um Small Talk über Nachrichten zu initiieren.

Das NewsTeller System wurde als vierstufige Pipeline implementiert:

Zunächst wird eine (große) Menge potentiell relevanter Events gesammelt.

Da ca. 84% der gefundenen Events nicht unseren syntaktischen und semantischen Wohlgeformtheits-Kriterien entsprechen, werden die Events im zweiten Schritt der Pipeline gefiltert. Dies wird mithilfe eines Klassifikators durchgeführt, der auf einem Datenset von ca. 6.000 Events trainiert wurde. Die Ergebnisse einer 10-fachen Cross Validation zeigen, dass ein globaler Random Forest Klassifikator einem Ensemble spezialisierter Klassifikatoren, die auf Teilproblemen trainiert wurden, überlegen ist. Der globale Klassifikator erreicht eine Precision von 63.04% und einen Recall von 59.55%.

Der dritte Schritt der Pipeline besteht aus dem Ranking der verbleibenden Events gemäß ihrer erwarteten Relevanz und dem Auswählen des relevantesten Events. Es werden vier geordnete Klassen verwendet, um die Relevanz eines Events zu beschreiben: Irrelevant, Partially Relevant, Relevant und Very Relevant. Gemäß dem "learning to rank"-Ansatz aus dem Information Retrieval deuten wir diese Aufgabe als Regressionsproblem, das mithilfe eines Random Forest Regressors gelöst wird, der auf einem Datensatz von ca. 3,200 Events trainiert wurde. Zwei verschiedene Ansätze wurden verglichen: Ein Ansatz benutzt ausschließlich Features, die basierend auf dem Event und der Nutzeranfrage definiert wurden, der andere Ansatz benutzt zusätzlich Features, welche die generellen Nutzerinteressen berücksichtigen. Die Ergebnisse in einer Leave-one-out-Evaluation auf dem Nutzer-Level legen nahe, dass beide Regressoren vergleichbar gut irrelevante Events vermeiden und dass die Einbeziehung von Nutzerinteressen die Detektion relevanter und sehr relevanter Events verbessern kann.

Im vierten Schritt der Pipeline wird eine Zusammenfassung des ausgewählten News Events generiert, indem der Satz extrahiert wird, in welchem das Event erwähnt wurde.

Zur Evaluation des Systems wurde eine Nutzerstudie mit 48 Probanden durchgeführt. Die Ergebnisse dieser Studie zeigen, dass beide regressionsbasierten Ansätze signifikant besser irrelevante Events vermeiden als eine zufallsbasierte Baseline. Die Hypothese, dass die Berücksichtigung von Nutzerinteressen die Detektion von relevanten und sehr relevanten Events verbessern kann, konnte allerdings nicht bestätigt werden.

Contents

List of Figures

List of Tables

1. Introduction

1.1. Motivation

This thesis is concerned with the field of social dialog systems – dialog systems capable of non-task-oriented behavior serving mainly social purposes. More specifically, it explores the area of news-related small talk, i.e., small talk referring to recent news events.
Why is this topic relevant and worth researching? There are two main motivations:

The first motivation is concerned with household robots. With the aging society being a challenge of the years and decades to come, intelligent household robots could allow people to continue living an independent life in their own homes instead of moving into a retirement home. This could not only improve their quality of life, but also help to reduce nursing costs. Therefore, the creation of intelligent household robots is a promising research area. One example of research in this direction is the humanoid robot ARMAR-III [2] which was developed at KIT and which operates in a kitchen environment.

Reeves & Nass [42] argue that humans treat computer systems as social actors and expect their behavior to be in line with social norms. If a computer system fails to fulfill its social role, users tend to perceive it as stupid and impolite. This of course also applies to household robots (especially humanoid robots with a human-like appearance). In order to be well accepted by their users, they will need to fulfill their role as social actors. This includes (among other things) the ability to conduct small talk. One of the potential topics of small talk are recent news events. Therefore, as a first step towards more "social" human computer interaction, it might be worthwhile to explore approaches for making small talk about current news events.

The second motivation is less of a practical nature but more philosophical: Instead of using small talk as means to an end, it can also be seen as a (small) step towards strong artificial intelligence. "Strong AI" or "Artificial General Intelligence" describes the goal of creating intelligent machines that have a similar competence level as humans in a variety of tasks [41]. It is distinguished from "weak AI" or "narrow AI" approaches that aim to build specialized systems with better-than-human performance in relatively narrow domains.

One often cited test for strong AI is the Turing test [52] which can be summarized in a simplified form as follows: "A system can be called intelligent if humans interacting with it over a command-line interface cannot tell whether they are interacting with a machine or another human." A system passing the Turing test would of course also need to be capable of conducting small talk. Although the Turing test was not devised as a real benchmark for AI and should be regarded as a thought experiment, the motivation is still

quite clear: A human-like system needs to have human-like abilities in relevant tasks. If the system is supposed to interact with humans in a natural way (which is quite likely), it needs to be able to deal with social aspects of the interaction – including small talk. Therefore, research on news-related small talk can also be seen as related to the ultimate goal of strong AI.

Although there are certainly more arguments than shown here, the two motivations given above already show that there are both practical and "philosophical" reasons for research in this area.

1.2. Problem Statement

This thesis project is a cooperation between the ISL (Interactive Systems Lab) at KIT (Karlsruhe Institute of Technology, Karlsruhe/Germany) and the HLT (Human Language Technology) research group at FBK (Fondazione Bruno Kessler, Trento/Italy).
The goal of this thesis is to develop a system for initiating news-related small talk which can be used by the social dialog system developed at the ISL. The system developed in this thesis uses events extracted from news articles by the NewsReader NLP pipeline which was developed by the HLT group in the course of an EU project. In analogy to the NewsReader system it uses, the system developed in this thesis was named "NewsTeller" and will be referred to by this name in the remainder of this thesis.

For the scope of this thesis, only a subtask of news-related small talk is considered: providing the user with basic information about a news event relevant to his/her utterance and interests. This can then serve an an entry point into a more elaborate small talk conversation about the event presented to the user.

It is assumed that the social dialog system provides a set of keywords extracted from the user utterance to the NewsTeller system (referred to as "user query"). Moreover, it is also assumed that a set of keywords representing the user's interests is given (referred to as "user model"). The task of the NewsTeller system is then to extract news events relevant to the user query from the KnowledgeStore (the storage component of the NewsReader system), to rank them according to their expected relevance with respect to both the query and the user's interests, to pick the most relevant event and to return a sentence about it. This output can then be used by the social dialog system to start a small talk conversation.

Let us consider an example: A user is interested in politics, science and soccer (all three keywords would be given as the user's interests). He tells the social dialog system: "I will go to Berlin next week to visit some museums." The social dialog system extracts the keywords "Berlin" and "museum" and passes them to the NewsTeller system. The NewsTeller system in turn queries the KnowledgeStore for events that are connected to the entities "Berlin" or "museum", and ranks them based on their expected relevance. This expected relevance is based on the user query (e.g., how many of the keywords match one of the event's constituents?), the user model (i.e., is the event related to politics, science, and/or soccer?) and other aspects (e.g., how long is the sentence in which the event is

mentioned?). The NewsTeller system then picks the event that seems to be most relevant according to these criteria. For the given query and interests, this could for example be a news event about the opening of a new science museum in Berlin. In a final step, the NewsTeller system extracts the sentence in which the selected event is mentioned from the original news article. This sentence is then handed back to the social dialog system which returns it to the user.

Note that the NewsReader pipeline is not explicitly used in this example workflow – only implicitly by using the content of the KnowledgeStore (i.e., the results of running the NewsReader pipeline on a corpus of news articles).

Overall, the NewsTeller system can be seen as a bridge between the NewsReader system (whose results it uses) and the social dialog system (by which it is used as a module).

The focus of this thesis was to develop the NewsTeller system as a standalone system. This also means that it was evaluated intrinsically. Moreover, the actual integration into the social dialog system was not part of this thesis project.

1.3. Structure of the Thesis

The remainder of this thesis is structured as follows:
Chapter 2 gives an overview of relevant related work.
Chapter 3 describes the architecture of the NewsTeller system.
Chapter 4 presents the results of a user study conducted in order to evaluate the system.
Chapter 5 concludes this thesis and shows some potential starting points for future work.

Appendix A contains implementation details of the NewsTeller system.
Appendix B gives more detailed information about the user study.

2. Related Work

2.1. Overview

This chapter serves two purposes:
On the one hand, it gives the necessary background about the two main systems that are being used in this thesis: the KnowledgeStore component of the NewsReader project (Section 2.2) and the ISL social dialog system (Section 2.3).
On the other hand, it classifies this thesis with respect to the research areas it touches and gives an overview of relevant related work in these fields (Sections 2.4 to 2.7).

This thesis does not strictly belong to any of the classical research areas, but touches and intersects several of them. In the following, its relation to these fields is sketched, highlighting both similarities and differences:

The NewsTeller project is related to **Information Retrieval** (Section 2.4) in the sense that relevant events are retrieved from the KnowledgeStore based on a user query. Although standard information retrieval approaches are usually concerned with the retrieval of documents (and not events), the need to rank potential results according to their expected relevance to a user query is common to both problems. However, the NewsTeller system also uses the user's interests to bias this ranking which is usually not done in information retrieval. Moreover, in contrast to standard information retrieval approaches, the NewsTeller system does not output a list of ranked candidate events, but selects a single event (the most relevant one) and outputs a sentence about it.

This makes it somehow related to **Question Answering** (Section 2.5) which has the goal of providing a single natural language answer to a natural language question posed by the user. In question answering, information retrieval is usually used as a first step before ranking potential answers and selecting the best of them. This workflow is similar to the one of the NewsTeller system. However, the NewsTeller system deals with a rather "fuzzy" information need compared to the factoid questions considered in question answering: The user basically asks the open questions "are there any news about X?" and there is no single correct response – many different events might be relevant and acceptable.

This property highlights the strong relationship to **Recommendation Systems** (Section 2.6) which try to recommend relevant items to a user based on a user profile. Especially the subfield of news access systems is relevant where the items to be recommended are news articles. Most approaches in this area consider a news website setting with the

goal of adapting the list of presented news articles to the user's interests. The NewsTeller system is however designed to be used in a dialog system, therefore a finer granularity was chosen: As it is not feasible to read out a whole news article to the user during a conversation, the focus is put on single news events. Moreover, only a single event is picked for "recommendation". Furthermore, most recommendation systems recommend their items based exclusively on the user model. The NewsTeller system however also takes into account a user query.

This work can also be seen as a variant of **Automated Summarization** (Section 2.7): The goal of the NewsTeller system is to create a short summary sentence based on a large amount of news articles. Most summarization approaches try to give a comprehensive summary of a complete document without omitting important information. The NewsTeller system, however, focuses for its summary only on one specific event and thus omits most of the articles' content.

In the following sections, related work from the research fields listed above will be presented and their relationship to this project will be highlighted.

2.2. The NewsReader System

2.2.1. Overview

The NewsReader system [1, 55][1] was developed in the course of an identically named EU project as cooperation between VU University Amsterdam, Universidad Del Pais Vasco, Fondazione Bruno Kessler, LexisNexis, ScraperWiki, and SynerScope. It is an NLP pipeline that processes massive amounts of online news articles and automatically extracts events from them (i.e., *what* happend *when* and *where*, and *who* was involved?). Moreover, it offers means for visualizing and exploring the resulting news stories. It is intended to be used by decision makers in business enterprises who need a comprehensive summary of current news stories in order to make good decisions.

The NewsReader system assumes that the events to be extracted may be mentioned in different articles and that information from these articles can be both complementary and contradictory. A variety of different NLP tools (e.g., part of speech tagging, named entity recognition, and word sense disambiguation) are used to process the input documents. Both entities and events mentioned in different documents are linked and merged where necessary by using coreference resolution. The KnowledgeStore subsystem serves as a central repository for all modules in this NLP pipeline. It stores both the original news articles, the annotations generated by the individual modules, and the extracted events, entities, and relationships. The latter abstract representation is then used to discover news stories spanning longer periods of time (e.g., the global financial crisis) and to visualize them [53].

[1] See http://www.newsreader-project.eu

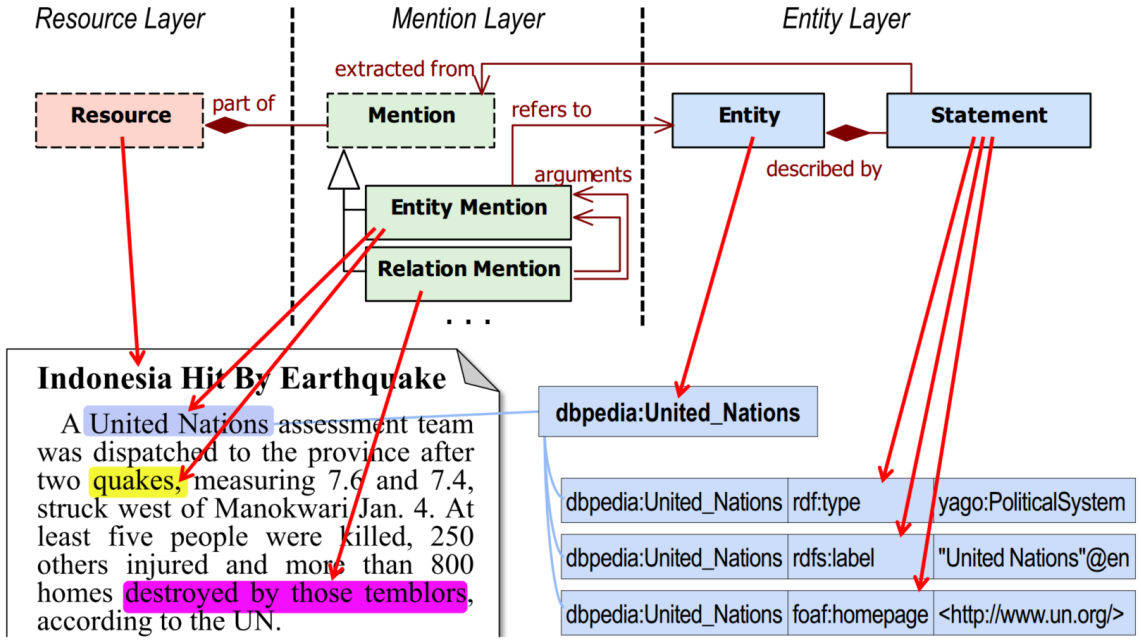

Figure 2.1.: Illustration of the three layers of the KnowledgeStore (Figure taken from [11]).

For the scope of this thesis, only the content of the KnowledgeStore component is of interest (i.e., the results of executing the NewsReader NLP pipeline). Therefore, it will be described in more detail in the following section.

2.2.2. The KnowledgeStore

The central data storage component of the NewsReader project is called "KnowledgeStore" [11]. It consists of three layers (illustrated in Figures 2.1 and 2.2):

- **Resource Layer**: stores unstructured content (i.e., the raw text files of the news articles) plus associated metadata (e.g., time of creation).
- **Mention Layer**: indexes snippets of resources from the resource layer that denote an entity (i.e., a group of characters in the news article referring to an entity).
- **Entity Layer**: stores structured content (i.e., events, entities, and their relationships) in form of RDF triples.

Information from all three of these layers is used in the NewsTeller system. We will now proceed to describe the three layers in more detail, starting with the entity layer and proceeding in a top-down manner.

As already stated, information in the entity layer is stored in the form of RDF triples. RDF stands for "Resource Description Framework" and is a widely accepted semantic web standard for representing symbolic information. In RDF, information is stored in form of `(subject, predicate, object)` triples. Both the subject and the predicate are

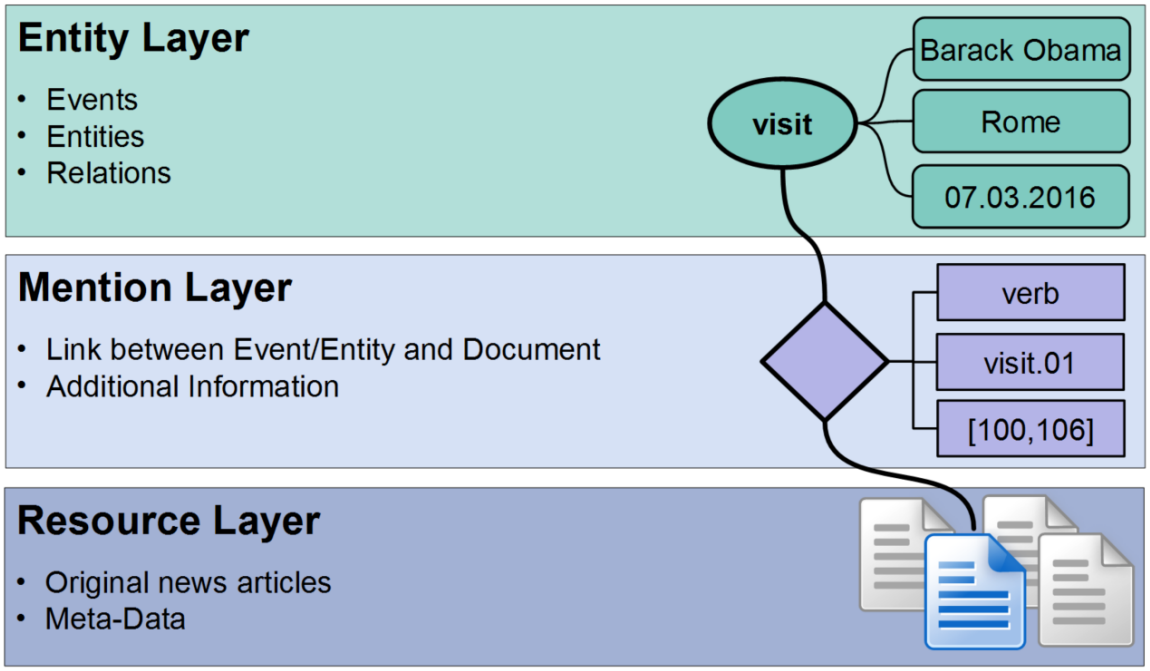

Figure 2.2.: Alternative illustration of the three KnowledgeStore layers and their content.

```
SELECT ?event
WHERE
{
  ?event rdf:type sem:Event
}
```

Figure 2.3.: A simple SPARQL query.

URIs and the object can be either an URI or a literal value (e.g., a string or an integer). Figure 2.1 depicts for example the RDF triple `(dbpedia:United_ Nations, foaf:homepage, <http://www.un.org>)`. This represents the piece of knowledge that "the United Nations' homepage is `http://www.un.org`". Note that both `dbpedia` and `foaf` are namespace prefixes.

One can also think of the `(subject, predicate, object)` triples as `(entity, property, property value)` triples, or alternatively as labeled directed edges `(start node, link label, target node)`. The latter interpretation is especially useful for visualization purposes and is the reason why a set of RDF triples is usually called an "RDF graph". See Pan [39] and Grobe [19] for more information about the RDF standard.

One way to formulate queries about an RDF graph is provided by the SPARQL query language [56]. Having a syntax similar to SQL, it allows to define templates against which the RDF graph will be matched. Figure 2.3 shows an example of a simple SPARQL query. `?event` in the **`SELECT`** clause is a variable whose possible assignments will be returned as a

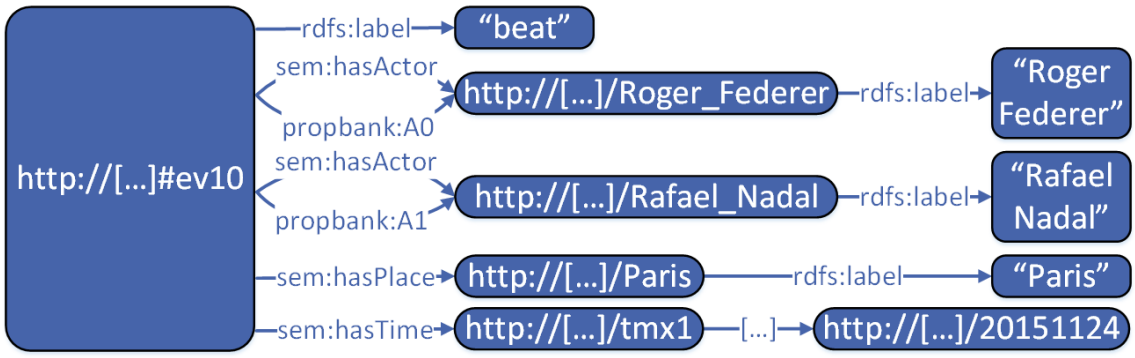

Figure 2.4.: Visualization of a simple event.

result of the query. In the **WHERE** clause, a simple template is defined: The node assigned to the variable `?event` must be subject of an RDF triple with predicate `rdf:type` and object `sem:Event` (where `rdf:` and `sem:` are namespace prefixes). The semantics of the query is "return all nodes of type event", or shorter "return all events".

Of course, SPARQL also allows for more complex and powerful templates to be defined (including regex matching, grouping, and constraining the search to named subgraphs). Due to space limitations, no deeper introduction into SPARQL can be given at this point, but the reader is encouraged to refer to the W3C recommendation [56] for further information.

The triples of the KnowledgeStore's entity layer make use of several namespace prefixes, e.g., `dbpedia`, `propbank`, and `sem`. The first two of them are based on the respective resources, i.e., the DBpedia database [3] (which contains RDF triples extracted from Wikipedia) and the PropBank project [38] (which is concerned with the annotation of verb arguments). The `sem` namespace refers to the "Simple Event Model" which was developed in the context of the NewsReader project [54]. It defines types of entities and relationships that are important in the context of event processing. For instance, the `sem:Event` node used in the SPARQL query in Figure 2.3 represents the abstract concept of an event. Moreover, useful relationships like `sem:hasActor`, `sem:hasTime` and `sem:hasPlace` between an event and its constituents (actors, times, and places, respecitvely) are defined.

Figure 2.4 shows the visualization of an event representing the sentence "Roger Federer beat Rafael Nadal in Paris on Nov 24 2015". The event node to the left is represented by an URI and is the anchor point for all information about the event.

The triple `(http://[...]#ev10, rdfs:label, "beat")` represents the fact that the label of this event is the word "beat". Most of the time, the labels of events are verbs, but in some circumstances also nouns can be valid event labels (e.g., "earthquake").

The simple event model described above provides link types that indicate the actors, places, and times involved in the event (e.g., the entity denoted by the URI `http://dbpedia.org/resources/Roger_Federer` participates as actor in the event under analysis).

property URI	explanation
Basic properties of all mentions	
`ks:refersTo`	Entity or event from the entity layer which this mention refers to.
`ks:mentionOf`	Resource from the resource layer out of which this mention was extracted.
`nif:beginIndex`	Index of the character in the resource text where this mention starts.
`nif:endIndex`	Index of the first character in the resource text after the end of this mention.
`nif:anchorOf`	The substring from `nif:beginIndex` to `nif:endIndex` in the resource text.
Additional properties of event mentions	
`nwr:pos`	The part of speech tag assigned to the word this mention is referring to (e.g., `nwr:pos_verb`).
`nwr:propbankRef`	IDs of the propbank rolesets that were matched to the word this mention is referring to.
`nwr:nombankRef`	IDs of the nombank rolesets that were matched to the word this mention is referring to.
`nwr:wordnetRef`	IDs of the wordnet synsets that were matched to the word this mention is referring to.
`nwr:pred`	The lemma (i.e., the base form after removing morphology) of the word this mention is referring to.

Table 2.1.: Table showing relevant mention properties.

Moreover, the propbank namespaces indicates which one of the actors is the agent (`propbank:A0`) and which one the patient (`propbank:A1`) – information that is crucial for the semantics of this "beat" event.

Note that of course in practice all nodes shown in Figure 2.4 have many more links to other nodes which are omitted in this simplified illustration.

After the thorough discussion of the entity layer, let us now turn to the mention layer. As already stated earlier, the mention layer connects the entities and events in the entity layer to the news articles in the resource layer. Every mention is identified by its unique URI and has several properties. Event mentions have some additional properties compared to entity mentions (e.g., the part of speech information). The mention properties used in the NewsTeller project are listed in Table 2.1. Mention properties can only be accessed via the CRUD[2] endpoint, not within SPARQL queries. The CRUD endpoint offers an interface that takes the URIs of the mentions of interest and returns a record containing all properties for each of the mentions. Within a record, these properties can be accessed by indexing them with the property URI. For instance, when being interested in the part of speech information of the mention "`http://en.wikinews.org/wiki/New_Beta_`

[2]Create, Read, Update, Delete

Version_of_MSN_Search_Service_from_Microsoft_released#char=14,22", one needs to invoke the CRUD endpoint with this mention URI. One then needs to index the resulting record with `nwr:pos` in order to get the part of speech information.

Finally, let us consider the resource layer. In the resource layer, the original news articles are stored together with some associated meta data. The CRUD endpoint offers two ways of accessing the resource layer: On the one hand, one can use the same functionality as for the mentions to retrieve document properties (i.e., document meta data, e.g., the creation time or the title of the document). On the other hand, there is a specialized method to download the content of the news articles. Both ways of access are used in the NewsTeller system.

The KnowledgeStore instance used in this thesis was populated by the NewsReader pipeline using news articles from the free online news source Wikinews [57]. This KnowledgeStore instance contains more than 100 million RDF triples and over 600,000 news events that have been extracted from almost 20,000 news articles from the period between November 2004 and October 2015. At the time of this writing, the described KnowledgeStore instance is available online under `http://knowledgestore2.fbk.eu/nwr/wikinews/ui`. In principle, however, the NewsTeller system can be used with any KnowledgeStore instance.

2.3. Social Dialog Systems

2.3.1. Spoken Dialog Systems

A spoken dialog system (SDS) is a computer system that can communicate with the user though spoken dialog – i.e., it uses speech as main modality for input and output. This section only serves as a brief introduction to the topic. For further information, the reader is referred to McTear [33] and Skantze [49] who both give thorough overviews of this research area.

A spoken dialog system usually consists of five components (as depicted in Figure 2.5):

- **Automatic Speech Recognition (ASR)**: transcribes the user's speech into text.
- **Natural Language Understanding (NLU)**: extracts a semantic interpretation from the transcribed user utterance.
- **Dialog Management (DM)**: decides which abstract action to take as reaction to the user input (maps the semantic interpretation of the input to a semantic response). May potentially use external data sources like databases, a user model and/or a dialog history.
- **Natural Language Generation (NLG)**: transforms the semantic response into a textual response.

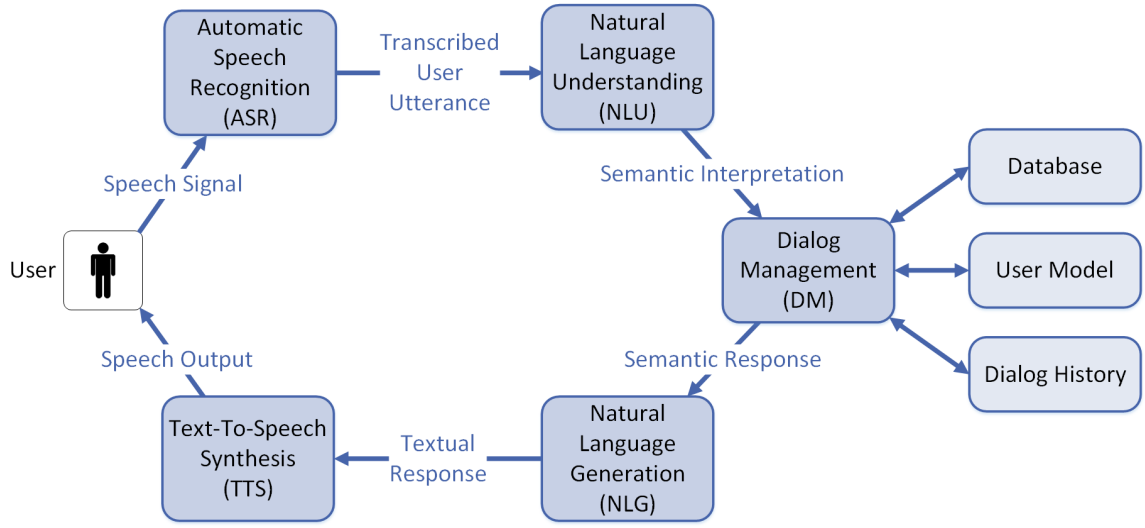

Figure 2.5.: Architecture of a spoken dialog system.

- **Text-to-speech synthesis (TTS)**: generates speech output based on the textual response.

The NewsTeller system presented in this thesis is mainly concerned with the dialog management component (picking the event to talk about, based on an external data source and a user model). NLG is done by using a simple sentence extraction mechanism. ASR, NLU, and TTS are not being dealt with in this thesis and are assumed to be given.

Note that due to the pipeline structure, performance of all components of an SDS is crucial – if one of the components fails, the overall system fails.

Although all components must work reasonably well, the "core" component of an SDS is the DM that decides how to react to the user's utterance. One can classify different dialog management approaches according to the following dimensions:

- **Initiative**:
  - *System Initiative*: The system is in complete control over the dialog flow (active system, passive user).
  - *Mixed Initiative*: Both the system and the user can direct the dialog flow (active system, active user).
  - *User Initiative*: The user has complete control over the dialog flow (passive system, active user).
- **Confirmations & Verification**:
  - *Explicit/Direct*: The system asks yes/no questions to confirm its hypothesis.
  - *Implicit/Indirect*: The system includes its hypothesis in its next utterance giving the user the chance to correct it.

  - *None*: the system does not confirm its hypothesis at all.

- **Dialog Control Strategies**:
  - *State-based*: The system is implemented as a finite state machine.
  - *Frame-based*: The system tries to fill a "frame" of different variables ("slots").
  - *Agent-based*: The system is composed of different agents responsible for different subtasks.
  - *Statistical*: The system is implemented using machine learning techniques (e.g., POMDP).

- **Error Handling**:
  - *Blaming the System*: "I'm sorry, I didn't understand you." The system is perceived as more likable but less competent.
  - *Blaming the User*: "You need to speak more clearly." The system is perceived as more competent but less likable.

- **Purpose**:
  - *Goal-oriented*: The system has a clear goal to reach and a well-defined task (e.g., booking a flight).
  - *Not Goal-oriented*: There is no well-defined goal for the interaction (e.g., social dialog systems).

See McTear [33] for more information on the dimensions of initiative, confirmations & verifications, and dialog strategies. For more information on error handling in SDS, the reader is referred to Skantze [49].

As Bickmore & Cassell note in [5], it is important for a dialog system to have a short response time (they give an upper limit of 1.2 seconds). They argue that a system which takes too long to respond to the user input will be perceived as less intelligent and therefore potentially as a less interesting conversation partner.

For the remainder of this thesis, we will focus on social dialog systems, i.e., dialog systems aiming to keep the user engaged in the conversation without the objective to fulfill any particular goal. As the research area of social dialog systems is still in its infancy, there is not yet any widely accepted definition of social dialog. For the scope of this thesis, the working definition of the ISL is adopted:

> Social Dialog is "natural, conversational interaction, typically focusing on the discussion of attitudes/opinions and (funny) small talk". [46]

In reference to the dimensions presented above, social dialog systems can be classified in the following way:

- Usually, one strives for a *mixed-initiative* dialog which is considered to be the most natural way of initiative distribution: It allows both interlocutors to "steer" the dialog and to re-initiate the conversation if it is lagging (i.e., to break the "awkward silence").
- Explicit confirmations tend to appear unnatural for social dialog situations, therefore *implicit confirmations* are preferable (or even *no confirmations* at all).
- Regarding dialog control, simple state-based approaches are not applicable: There is a wide range of potential topics and it is usually not possible to design a finite state machine covering all of them. The same argument applies to frame-based approaches. Therefore, *agent-based* dialog managers or *statistical* systems seem to be reasonable choices. However, also agent-based systems need a large number of agents in order to cover all relevant topics. Statistical systems on the other hand usually require a large amount of representative training data which currently is difficult to obtain. The work presented in this thesis can be viewed as a statistical agent of the overall system – it is concerned with the relatively specific topic of initiating news-related small talk and is based on machine learning approaches.
- As likability is more important than perceived competence, the *"blaming the system"* strategy is usually used to signal errors.
- Finally, the purpose of social dialog systems is clearly *not goal-oriented*.

Evaluation of social dialog systems tends to be more difficult than evaluation of goal-oriented dialog systems: Most metrics of task success, efficiency, etc. are not applicable due to the inherent lack of a well-defined (and thus measurable) task. Evaluation is therefore mostly done by conducting user studies in order to measure user satisfaction.

As the system developed in this thesis is only a module for a social dialog system, most aspects of spoken dialog systems will be ignored for the remainder of this thesis. It is assumed that the dialog management component that uses the NewsTeller system already takes care of them.

2.3.2. Small Talk

Small talk can in general be defined as non-task-oriented conversation about safe and neutral topics. As Endrass et al. [17] observe, small talk allows to establish social relationships with strangers, to get acquainted with each other and to avoid awkward silence. Bickmore & Cassell [5, 6] argue that small talk is used to establish mutual trust between the conversation partners. They assume that trust is mainly based on three relationship aspects: familiarity (how well do the interlocutors know each other?), solidarity ("like-mindedness") and affect (degree of liking for each other). They further argue that small talk helps to improve these relationship aspects in the following ways:

By keeping the conversation on safe and neutral topics (where people tend to be agreeable), it avoids face threats and helps the interlocutors to have their social role accepted, thus increasing solidarity. Moreover, due to the exchange of short utterances, the interlocutors synchronize with each other. The effect of a smooth conversation in turn increases the affect for each other. Finally, by talking about personal topics (e.g., anecdotes), the interlocutors contribute to the common ground of the conversation (i.e., the knowledge and information that is shared by all conversation partners) which results in increased familiarity with each other. Thus, as Bickmore & Cassell conclude in [6], small talk is an important way to build relationships and trust among each other.

The topics being talked about during small talk can be classified into three groups [17]:

- **Immediate Situation**: the immediate context of the conversation (e.g., the surroundings in which the conversation takes place).
- **Communication Situation**: anything referring to the interlocutors (e.g., their hobbies).
- **External Situation**: the wider context (e.g., recent news).

The specific topics chosen by the interlocutors are highly dependent on the interlocutors themselves (e.g., their cultural background [17]) and the context in which their conversation takes place. The system developed in this thesis will be used to make small talk about events from recent news, i.e., about the external situation.

Endrass et al. [17] define the following prototypical structure of a small talk conversation which is repeated for each topic being talked about:

1. question
2. answer
3. reverse question / understanding / acknowledgement / evaluation
4. zero or more idle moves

They note that steps 3 and 4 can be repeated multiple times and that this prototypical structure of course does not match all possible small talk conversations (one could e.g., imagine that step 1 is omitted when one of the interlocutors starts a new topic by just narrating a story). Nevertheless, this structure can be used as a first approximation for constructing dialog systems capable of doing small talk.

This thesis assumes that in step 1, the user will ask a news-related question and the NewsTeller system will then in step 2 answer with relevant information. This "opens" the topic for further discussion (steps 3 and 4: follow-up questions, comments etc.). This further discussion is however not inside the scope of this thesis.

2.3.3. Examples of Social Dialog Systems

Although the creation of social dialog systems is a relatively new and barely explored research area, there are already some interesting approaches. In this section, five of them will be briefly introduced.

Babu et al. [4] developed a virtual receptionist which can be used for goal-oriented tasks, e.g., leaving messages for lab members. In addition to this functionality, it is also capable of some social behavior: telling jokes and talking about the movies and the weather. They evaluated how users interacted with their virtual receptionist and found that the system was being accepted as a social actor and that its social dialog capabilities were often accessed by the users.

In their work, Babu et al. use the movies and the weather as possible small talk topics. When talking about the movies, their system randomly picks a movie from its database that has not been used in a prior conversation with the user. It then asks whether the user has seen this movie. Also the NewsTeller system keeps track of the events already talked about to avoid talking about the same event twice. However, the selection of the event to talk about is not done randomly in the NewsTeller system but based on a user query and a user model. The system of Babu et al. mostly poses yes/no questions to the user, therefore having a high system initiative. The NewsTeller system on the other hand responds to a user query by providing relevant information and has therefore a relatively high user initiative.

Also Bickmore & Cassell [6] built a so-called "embodied conversational agent" (ECA), i.e., a social dialog system with a virtual avatar. Based on their assumption that small talk helps to build trust between the interlocutors (see Section 2.3.2) they analyzed whether the use of small talk can influence the perception of their virtual agent in a real estate application. Moreover, they analyzed the role of non-verbal behavior (i.e., gestures and facial expressions) in this context. They performed a user study comparing four setups differing in whether the virtual avatar was visible and whether small talk behavior was enabled. Their evaluation results were surprising: Users preferred both embodiment without small talk and small talk without embodiment to the the combination of small talk and embodiment. Bickmore & Cassell suggested that the use of small talk indicates an extroverted personality and that the nonverbal behavior used by their system projected an introverted personality. This hypothesis was supported by observed differences between extroverted and introverted participants. Their results indicate that a good match and synchronization of small talk behavior and nonverbal behavior of the avatar seem to be crucial for user acceptance.

In this thesis, a text-only interface is used, hence nonverbal behavior does not apply. Moreover, the system of Bickmore & Cassell uses small talk in a goal directed manner (to increase trust before asking task-related personal questions, e.g., about the user's financial situation) whereas in this thesis small talk is not viewed as a means to an end but as purpose for itself.

Isbister et al. [25] took a different approach to social dialog: Instead of creating a dialog system that directly interacts with the user, they devised a helper agent for virtual environments. Its purpose is to monitor conversations between human avatars and to initiate small talk when it notices that a conversation is lagging (i.e., when there is a long period of silence). They argue that this is especially useful and necessary in virtual meeting spaces: There is usually barely any context to refer to (the conversation partners might be located in different parts of the world and might have a different cultural background), hence it is difficult to find good small talk topics in this setting. After having proposed a small talk topic to talk about (following a fixed script), the agent leaves the conversation, thus not taking part in the following discourse. Potential small talk topics were manually selected a priori by the designers based on an online survey and are picked randomly at run time. Ibister et al. evaluated their system on conversations between Japanese and American students and found that introducing safe small talk topics tended to lead to a smoother conversation whereas introducing unsafe topics tended to make the conversations more interesting to the conversation partners.

Although this alternative approach to small talk is very interesting, the system developed in this thesis is used for direct user interaction and hence follows a different paradigm. Moreover, instead of proposing a randomly selected topic from a predefined set, the NewsTeller system dynamically reacts to the user's utterance and finds a suitable topic on the fly.

Endrass et al. [17] put their focus on culture-related differences in small talk behavior focusing on the difference between Germany and Japan. After analyzing recordings of small talk conversations, they build a planner-based systems to generate artificial example dialogs typical for the two different cultures.

In contrast to the work presented in this thesis, their focus was on understanding and reproducing cultural differences in small talk and not on creating an interactive social dialog system. Therefore, their work is only partially applicable to the system devised in this thesis.

Finally, Yoshino et al. [58, 59, 60] present a POMDP-based spoken dialog system for news navigation. Their system is capable of presenting news stories to the user (reading out the news headline), summarizing them (using the lead sentences of the article) and answering questions about the news stories. In general, it is assumed that the user formulates his/her interest explicitly in the user utterance. As knowledge source, a collection of raw text news articles is used which is updated on a daily basis. For question answering, the question is transformed into a predicate-argument structure which is then compared to potentially relevant news articles. If there is an exact match, the matching predicate unit is used to generate the system response. If there is no exact match, a partially matching predicate unit is used to create the system response.

Their work is strongly related to the topic of this thesis because it considers an information-seeking dialog in the news domain. Although their system does not explicitly put a focus on small talk, it still has a quite similar setting compared to the NewsTeller system. However, their system does not take into account the user's profile in any way (they expect the user to formulate his/her interests as part of the query). Moreover, topic pre-

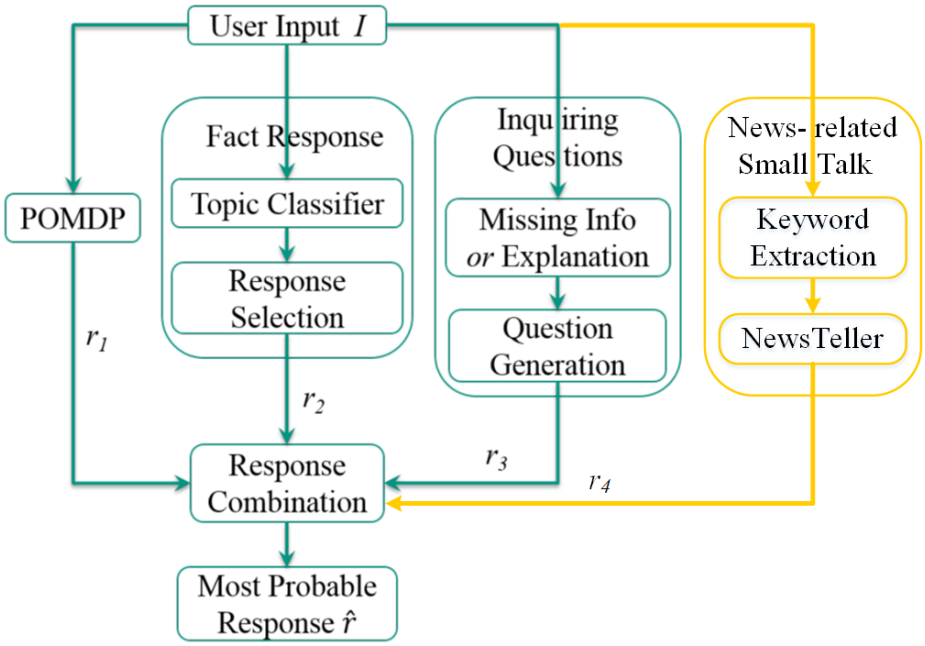

Figure 2.6.: Architecture of the social dialog system (taken from Schmidt et al. [47] and modified by adding the NewsTeller system).

sentation is quite limited as it consists only in returning the headline of the article. Also the summarization capability is very rudimentary, returning simply the lead sentences of the article as summary. Therefore, only a small part of the available information is used for topic presentation and summarization. The NewsTeller system, however, makes use of events extracted from anywhere in the text for presenting a new topic to the user. Although a question answering capability similar to the one presented in the work of Yoshino et al. is a long-term goal of this research, it is not dealt with in this thesis and left for future development.

2.3.4. The ISL Social Dialog System

The system developed in this thesis is intended to be used as a module by the social dialog system developed at the ISL (Interactive Systems Lab) at KIT. Schmidt et al. present this system in [47].

Their system is supposed to conduct small talk dialogs with the user. The system has the main goal of keeping the user interested in the conversation. Figure 2.6 shows the architecture of their system. Right now, their system is text-based, therefore the ASR and TTS components from Figure 2.5 are missing. Moreover, instead of the classical pipeline consisting of NLU, DM, and NLG (with the DM component deciding how to react to the user input), their system consists of different modules that are executed in parallel. Each module maps the user input to a potential response which is annotated with a confidence value. The responses proposed by the different modules are then collected and the one with the highest confidence value is selected and returned to the user. This architecture

can easily be extended with new modules that can be completely independent from the already existing ones. Right now, their system contains three modules (shown in green):

- **POMDP**: a goal-oriented module based on a "Partially Observable Markov Decision Process" (POMDP). It is used to provide information about restaurants to the user.
- **Fact Response**: a module that uses facts extracted from Wikipedia entries and RSS news feeds. After classifying the topic the user is talking about, it responds with a fact that is likely to be relevant to the user utterance.
- **Inquiring Questions**: a module that analyzes the user input for missing information and unknown words. It either generates a question about missing information or asks the user to explain an unknown term.

The NewsTeller system presented in this thesis can be thought of as a fourth module (added to Figure 2.6 in orange): It takes keywords present in the user input and responds with a relevant news event extracted from the KnowledgeStore.

2.4. Information Retrieval

The task of information retrieval (IR) can be stated as follows:

> Given a corpus of documents and a user query (a set of keywords expressing the user's information need), find all relevant documents (where relevance corresponds to the satisfaction of the user's information need) [15].

In most systems, some internal scores or ranking signals are used to determine the degree of relevance for each document. Diaz [15] distinguishes three categories of such ranking signals:

- **Query-document**: features indicating the degree of matching between the user query and the document.
- **Query-independent**: features depending only on the document.
- **Document-independent**: features depending only on the query.

One example of a ranking signal is the cosine similarity of tf-idf (**t**erm **f**requency - **i**nverse **d**ocument **f**requency) vectors [48]. The entires of a tf-idf vector are based on the frequency with which the respective word appears in the given document (term frequency) and this word's general frequency in all other documents (inverse document frequency). A word appearing often in a given document but only rarely in the overall corpus of documents can be thought of as characteristic for this document.

One can either base the ranking on a single ranking signal or one can combine different ranking signals into a global ranking score. This combination can be done either manually or by using machine learning. The latter case is called "learning to rank" and Liu [29] gives a thorough overview over this topic. He distinguishes three main approaches of how the ranking problem is translated into a machine learning problem:

- **Pointwise Approach**: A single document is used as input and its degree of relevance is used as output.
- **Pairwise Approach**: Pairs of documents are used as input and their preference relationship is used as output (typically represented as value from the interval $[-1.0, 1.0]$).
- **Listwise Approach**: The set of all documents is used as input and the output consists either of the degree of relevance for every single document or of an ordered list of all documents representing the relevance order.

The pointwise approach allows for a pretty straightforward application of machine learning algorithms, e.g., regression techniques. The pairwise and the listwise approach are a bit more difficult to implement but offer the possibility to also take into account inter-document relationships (e.g., hyperlinks in web documents).

Independent of whether one uses regression or classification techniques, there is the need for labeled training data. There are three general approaches to labeling in the "learning to rank" context:

- **Binary labels**: The problem is interpreted as binary classification problem with two classes: "irrelevant" and "relevant".
- **Multiple ordered categories**: There is a number of different classes with increasing value. Usually five labels are used [22]: "bad/irrelevant" (0), "fair/partially relevant" (1), "good/relevant" (2), "excellent/highly relevant" (3), and "perfect/perfect match" (4).
- **Ranking**: A permutation on the documents is defined as ground truth. This means that an individual document does not have an individual score, but only the overall ordering of all retrieved documents for a query is assigned a score based on its closeness to the ground truth.

In the following, we will focus on the case of multiple ordered categories in combination with the pointwise approach based on regression.

Usually, when doing regression on multiple ordered categories, one transforms the category labels before applying the regression by using the following formula [10]:

$$regressionValue = 2^{relevanceLabel} - 1$$

This is done to emphasize training examples with a high relevance score – the underlying reasoning is that it is most important to "get the top of the list right", i.e., correctly identifying the most relevant examples.

One of the common regression models used in this setting are random forests. As Mohan et. al [36] showed, simple random forests can compete with state of the art systems (like SVM-based systems or gradient boosted regression trees) and thus offer a "low-cost alternative" (as they can be used "off the shelf" without modifications).

For training random forests on the regression values defined above, usually the mean squared error (MSE) loss is used in point-wise regression-based approaches. However, for evaluating the performance any learning-to-rank system, it makes more sense to look at the overall ordering of the documents retrieved for a query. Using correlation or MSE does give some information about how well the regression problem was solved, but what counts in the end is how good the resulting ordering is: The user does not see the internal scores of the documents, only the resulting ordering.

Liu [29] lists some metrics that are commonly used in the "learning to rank" framework. In the case of binary class labels, one can use the $Precision@k$ metric which is defined for each query as follows:

$$Precision@k = \frac{\text{number of relevant documents in top k positions}}{k}$$

This metric can also be used with multiple ordered categories by mapping the categories into two classes. For instance, one could map the five relevance categories given above to a binary label by defining that "bad/irrelevant" corresponds to the negative class and all other labels map to the positive class.

Another very popular metric used for evaluating the performance of learning to rank systems in the IR community is the "Normalized Discounted Cumulative Gain" (NDCG). It is based on the "Discounted Cumulative Gain" (DCG) which is defined as follows:

$$DCG@k = \sum_{r=1}^{k} G(\pi^{-1}(r)) \cdot \eta(r) \quad \text{with} \quad \eta(r) = \frac{1}{\log_2(r+1)}$$

The function $\pi^{-1}(r)$ returns the document at position r in the resulting ranking, and $G()$ maps this document to a value (using the formula introduced above for the $regressionValue$). The position discount factor $\eta(r)$ reflects the importance of the position r in a ranking. The parameter k denotes the number of documents retrieved for the given query.

So for the computation of the DCG, we sum over all positions in the ranking, taking the result of the ground truth score of the element at this position multiplied the importance of the position. Therefore, the DCG is a weighted sum over the labels with decreasing weights. It is easy to see that the DCG is maximized if the upper ranks are occupied by events with high scores. As the DCG is defined dependent on the number of query results k, it needs to be normalized in order to compute a meaningful average across many queries. This normalization is done by dividing the achieved DCG by the maximally achievable DCG for the given document set (i.e., when the documents are sorted in a descending manner according to their ground truth labels). This measure is then called NDCG.

Regarding the engineering aspect, Carman & Ibrahim [10] propose the following approach towards learning-to-rank:

1. Retrieve all documents.

2. Sort all documents according to their BM25 score.

3. Keep only the top n documents.
4. Compute other features for each document (e.g., BM25 score for the document title).
5. Normalize all features.
6. Label them with their relevance labels.

Step 2 and 3 basically use the BM25 score as pivot for a first filtering step by throwing away all documents with a low BM25 score. This helps to make sure that obviously irrelevant documents get filtered out early on and don't play a major role in the later labeling process. However, this approach requires that the pivot feature is very expressive. This is assumed to be the case for the tf-idf based BM25 measure which can be considered a standard in the Information Retrieval community [29].

Carman & Ibrahim [10] also investigate the influence of three hyperparameters on the performance of random forests in this regression-based learning to rank setting: sample size, number of negative examples, and optimization function. Their conclusions are the following: Reducing the sample size for the individual trees in the forest helps to decorrelate them. This can result in improvements of both training runtime and performance. Reducing the number of negative examples in the training set improves training time, but at the same time causes performance to slightly degrade. Finally, replacing the MSE optimization criterion with an NDCG optimization criterion for training the random forest results in slight performance increases but makes training much more complex. Note that this replacement of the optimization function makes the approach "listwise" as it optimizes a function defined on the complete ranking and no longer a function defined only on single documents.

Duan et al. [16] give an example application of the learning to rank scheme described so far. Their goal is to rank tweets for a given query. They use an SVM-based learning-to-rank approach using four ordered categories (bad, fair, good, excellent). They distinguish three types of features:

- **Content relevance**: length of the tweet, BM25 scores, etc.
- **Twitter specific**: number of hashtags used, number of URLs used, etc.
- **Account authority**: information about the author of the tweet (e.g., number of followers).

They picked 20 example queries and labeled 500 tweets per query, yielding a total data set of 10,000 labeled tweets. For evaluation, they used NDCG in a five-fold cross-validation.

Although the objective of their system is different from ours, their learning-to-rank approach does have some parallels to the ranking of news events: The objects being ranked are tweets which have a maximum size of 140 characters. Also the news events being ranked in the NewsTeller system are "small" documents, i.e., they can usually be described by one short sentence. This small document size differs from most learning-to-rank approaches that are concerned with the ranking of longer documents, like websites in the

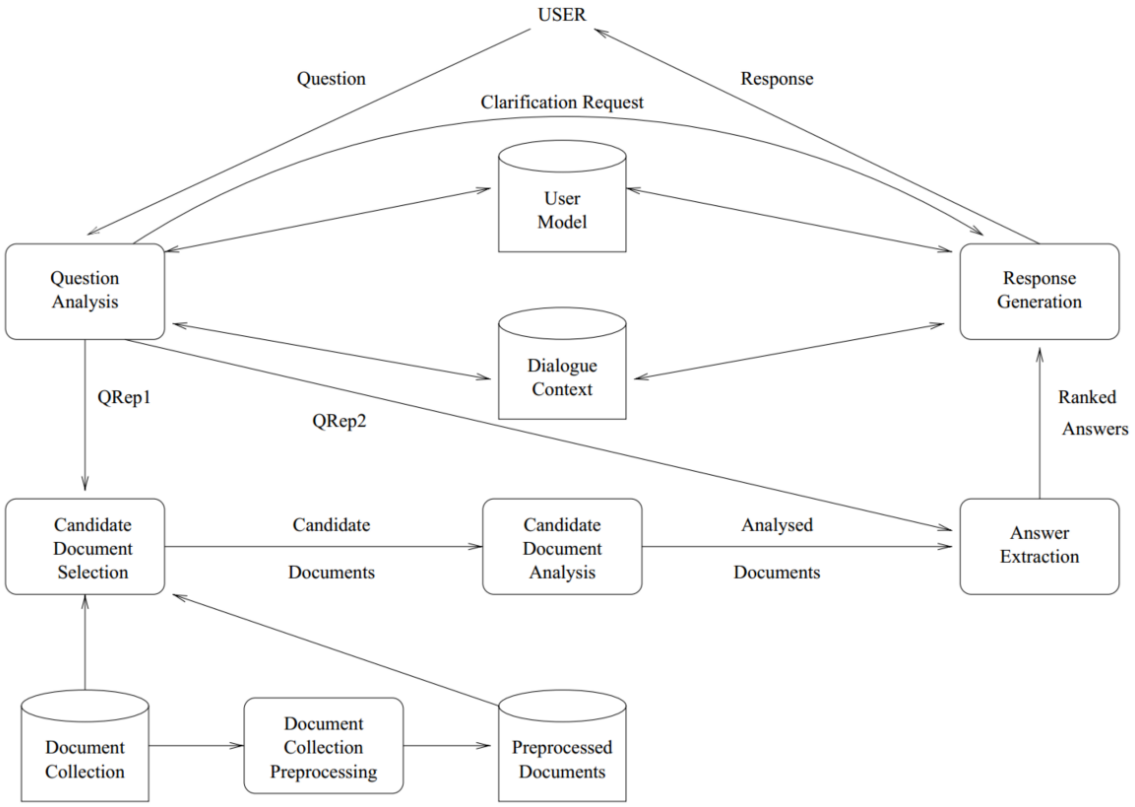

Figure 2.7.: Generic Question Answering system architecture (taken from Hirschman & Gaizauskas [24]).

context of a web search engine. Regarding potential features, however, only the ones from the "content relevance" category are applicable to the work presented in this thesis as the two other categories are problem-specific. Moreover, Duan et al. do not make use of any user model as is done in this thesis.

2.5. Question Answering

The goal of question answering (QA) is to give concise answers to questions formulated in natural language. It is therefore different from information retrieval as presented in Section 2.4.

Hirschman & Gaizauskas [24] describe a generic architecture for the QA task. It is shown in Figure 2.7 and consists of five main steps:

- **Question Analysis:** analyzes the question posed by the user with respect to the expected semantic type of the answer (e.g., number, person, or date) and other constraints the answer needs to fulfill. This process can potentially be influenced by a user model and/or the dialog context.
- **Candidate Document Selection:** finds a small set of documents that likely contain the correct answer. Usually this is done by using information retrieval. The

underlying document collection being used may be preprocessed (e.g., by performing named entity recognition).

- **Candidate Document Analysis:** performs a detailed analysis of the candidate documents (named entity recognition, parsing, etc.).
- **Answer Extraction:** extracts candidate answers from the analyzed documents and ranks them based on the constraints extracted during question analysis.
- **Response Generation:** creates a textual response based on the answer ranked highest. Can be potentially influenced by a user model and/or the dialog context.

It is possible to map the NewsTeller system into this generic QA architecture:

The underlying document collection is the set of news articles. It is preprocessed by the NewsReader NLP pipeline which extracts entities, events, and their relationships. The user utterance can be interpreted as question. It is analyzed by the social dialog system which extracts keywords from it and hands them to the NewsTeller system. The candidate document selection corresponds to searching for potentially relevant events in the KnowledgeStore. The subsequent candidate document analysis is missing in the NewsTeller system as the analysis of the news articles has already been done during preprocessing. The answer extraction step roughly corresponds to the ranking of events performed in the NewsTeller system which is however not only influenced by the keywords from the user query but also by the conversation history and the user model. The final response generation corresponds to extracting the sentence in which the selected event is mentioned.

Although this mapping might appear promising on the first look, it leaves out an important aspect: The goal of question answering is to find the correct answer to a factoid question. The NewsTeller system, however, tries to propose interesting small talk topics that match both the user's utterance and his/her interests. Since the "question" in this context is not factoid in nature (e.g., there is no expected answer type), there is no "correct" answer the NewsTeller system could find – there are only more relevant and less relevant answers.

One example for a QA system is the Ephyra framework developed by Schlaefer et al. [43, 44, 45]. In their system, processing takes place in three steps:

- **Query Formation**: corresponds roughly to the question analysis step in the generic architecture.
- **Search**: contains candidate document selection, candidate document analysis and a part of answer extraction.
- **Answer Selection**: corresponds to the scoring part of the answer extraction step.

During query formation, the question string from the user is analyzed and three types of information are extracted: target (the entity of interest), property (the specific attribute the user is interested in) and context (further information helping to nail down the concrete information need). For instance, consider the question "How many calories are

there in a Big Mac?". The target of this question would be "calories" the property would be "number" and the context would be "Big Mac". The extraction done during question analysis is based on learned patterns. Based on the extracted information, different queries are generated for the "search" step.

During search, both unstructured sources (like Google or Yahoo! web searches) and (semi-)structured sources (e.g., the CIA World Factbook or Wikipedia) are used to find answer candidates.

During answer selection, different filters are applied to score the answer candidates and to select the most promising one. Filtering is done for example by matching learned answer patterns against the answer candidate, taking into account target, property and context as extracted during query formation.

In his Bachelor's thesis [27], Kaiser extended the Ephyra system into an interactive QA system. He combined three approaches: named entity disambiguation (based on Wikipedia disambiguation sites), anaphora resolution (based on the co-reference resolution module from the Stanford CoreNLP toolkit) and context tracking (based on the contexts given to Ephyra in previous turns). Using these three techniques, the questions are modified before handing them over to the Ephyra system and some additional scoring is performed on the answers being returned. This way, the interactive QA system is able to deal with follow-up questions by the user and thus to have an information-seeking dialog with the user.

This information-seeking dialog is somewhat similar to the approach taken in this thesis for small talk: The common idea is to respond to a user's utterance using a short natural language response and to keep open the possibility of follow-up questions. However, this follow-up capability itself is not in the scope of this thesis but is left for future development.

2.6. Recommendation Systems

2.6.1. Overview

Recommendation systems (also called recommender systems) try to present relevant items to a user based on this user's profile. Lops et al. [30] and Pazzani & Billsus [40] both give a good overview over this research area.

In information retrieval, the underlying assumption is that the user has a well-defined information need which is represented by the user query. In the area of recommendation systems the user however usually has a rather vague information need which could be expressed as "provide me with interesting items" (where "interesting" is a rather fuzzy term) [7]. Since there is usually no explicit user query in this setting, a user profile is used for judging the relevance of different items.

Most recommendation systems present a list of recommendations to the user, hence also the order of presentation needs to be determined by the system. The user then selects interesting items to retrieve more information about them. Based on this interaction, the user profile can be updated.

One can in general distinguish two approaches to recommendation systems:

- **Collaborative Approach**: The system recommends items that other users with a similar profile have liked in the past.
- **Content-based Approach**: The system recommends items whose description match the user's profile.

We will focus on the content-based approach here, as the collaborative approach is not applicable to the system devised in this thesis (we cannot assume to have a sufficient amount of users and ratings which is referred to as the "cold start problem").

In the content-based approach, there is a need to represent the items and their attributes in a structured way (e.g., as database entries) in order to match them to the user profile. This works well for items like movies or books (with attributes like genre, author, actors, etc.) but is more challenging for items like websites or news articles. In this case, one needs to convert the unstructured text into some structured representation by extracting informative features (e.g., tf-idf scores). The user profile on the other hand usually contains the user's preferences (i.e., a description of the types of items the user might be interested in) and the history of the user's interactions with the system. This history can for example be used to prevent the system to recommend news articles that the user has already read. On the other hand, the history can also be used as training set to infer the user's preferences using machine learning techniques.

The field of recommendation systems is relevant to the project presented in this thesis, as it shares the property of selecting relevant items (in this case news events) based on a user model. However, in contrast to most recommendation systems, the NewsTeller system also takes into account a user query.

One example for a content-based recommendation system with an approach related to the system developed in this thesis is given by Mooney & Roy [37]. They present their work on a book recommendation system in a library setting. They argue that collaborative filtering is not applicable in this case, as it requires large numbers of both users and ratings which are hardly obtainable in a library setting. Books are rated on a ten-point Likert scale which is subsequently translated in a binary classification problem (with books belonging to the positive class if their rating is at least six). For each book, various pieces of information are extracted from the amazon.com website (e.g., title, author, synopsis, comments) and stored as bag-of-words. Their system is based on a naive Bayes classifier. For training, they use about 3,200 books being rated by four annotators. For evaluation, they use a ten-fold cross-validation. They look at the top k entries in the resulting ranking and compute the *Precision@k* metric introduced in Section 2.4 as well as the average user rating among these top k entries.

Also the problem explored in this thesis naturally lends itself towards the content-based approach, as only a small number of users and ratings are available. Also in our approach, the ranking data set was created by different annotators. It was evaluated with variants

of the *Precision@k* metric based on a leave-one-out procedure (which is similar to cross-validation). However, instead of solving a binary classification problem, we framed the selection of relevant events as a regression problem on the class labels of multiple ordered categories.

In the following two subsections, we will take a closer look at the ways of representing a user's preferences (Section 2.6.2) and at recommendation systems being used in the news domain (Section 2.6.3).

2.6.2. User Modeling

In general, a user profile (or user model) is a representation of a user regarding aspects important for the given application. According to Gauch et al. [18], it can consist of demographic information (name, age, country of citizenship, etc.) but also of other information like interests and preferences. For recommendation systems, especially the latter part is important as it can be used as a description of potentially relevant items. Information about a user can be collected in an explicit or an implicit way (e.g., asking the user to fill out an questionnaire vs. deducting information from usage logs).

Gauch et al. [18] also give an overview of different approaches to modeling a user profile. They distinguish three types of representation:

- **Keyword Profile**: a bag of keywords, each with its associated weight. This is the most simple and most commonly used profile representation.
- **Semantic Network Profile**: a graph with nodes corresponding to keywords and edges corresponding to co-occurrence of keywords in documents interesting to the user.
- **Concept Profile**: a graph with nodes corresponding to concepts and edges corresponding to relationships between these concepts.

In this thesis, a simple keyword profile is used for representing a user's interests. The system devised in this thesis assumes that this model already exists, hence no information about the user profile will be collected.

Diaz et al. [12, 14] further distinguish between a long-term and a short-term user model for their digital newspaper application.

The long-term model is used to represent the long-term interests of the user and is therefore assumed to remain constant. It consists of weights for different news categories (like "sports" or "economy") and a set of freely chosen keywords. Information for this long-term model is gathered in an explicit way by asking users to rate different news categories on a four-point Likert scale and to give keywords they are interested in.

The short-term model, on the other hand, is based on user feedback for recently received documents, hence on implicitly collected information.

Also Billsus & Pazzani [7] distinguish between a long-term and a short-term model for their news access system. They use both explicit user feedback (e.g., allowing the user to mark documents as "interesting", "not interesting", and "I already knew this') and implicit user feedback (e.g., based on how much of a story the user has read).

Their short-term model uses a "nearest neighbor" approach: New news stories to be recommended are picked based on the cosine-similarity of their tf-idf vectors with the vectors of the n most recently recommended news stories. In order to be picked, a new story must be close enough to one of the recently read stories (indicating a related topic) but not be too close to any of them (indicating a potential duplicate). The purpose of this short-term model is to keep track of "threads" of current events the user is following. This is used to provide follow-up stories referring to recently read articles.

The long-term model, in contrast, models the general user interest independent of currently followed threads. For each category of news articles, Billsus & Pazzani identified a set of stable keywords (by selecting keywords appearing often in the top n tf-idf words for documents from this category). They then trained a naive Bayes classifier to estimate the probability of the document being interesting to the user given the keywords appearing in it.

In their approach, the short-term model is used by default to retrieve a relevant document. Only if it fails, the long-term model is used as a fallback solution. Hence, they prefer following a currently open news thread over starting a new one.

In this work, only a long-term user model is used, although one can think of the user query as representation of short term interests.

2.6.3. Adaptive News Access

One application of recommendation systems is the area of adaptive news access. Billsus & Pazzani [8] give an overview over this topic. They distinguish four different types of adaptive news access:

- **News Content Personalization**: recommending news stories based on the user's interests.
- **Adaptive News Navigation**: re-configuring the user interface to facilitate the use of sections frequently accessed by the user.
- **Contextual News Access**: providing news based on the content currently being viewed by the user (e.g., emails or websites).
- **News Aggregation**: aggregating and classifying news content from different providers.

In order to evaluate an adaptive news access system, one usually compares the performance and/or user experience for two versions of the system: static (i.e., not using the user model) and adaptive (i.e., using the user model). Recommendation systems are usually used in the area of news content personalization, therefore the remainder of this section will focus on this subarea.

A news content personalization system needs to take into account that the user has multiple interests that change over time. Moreover, it needs to consider notions like novelty when recommending news (e.g., not recommending the same story over and over again). Furthermore, it needs to avoid tunnel vision, i.e., breaking news should always have a good chance to be recommended to the user.

Billsus & Pazzani describe a system for adaptive news presentation [7]. The goal of their system is to display news stories that are likely to be interesting to the user. It uses a wide range of news channels (top stories, politics, world, business, technology, sports, science, health, and entertainment). As discussed in Section 2.6.2, they employ a user model consisting of a short-term and a long-term model to estimate the relevance of news stories to a user's profile.

Diaz et al. [13, 12, 14] describe a similar application in a digital newspaper scenario. However, in addition to selecting news stories interesting to the user, they also provide short summaries of these news articles to help the user judge the relevance of the articles before reading them. Their user model (also consisting of a long-term and a short-term model) has been presented in Section 2.6.2 and is used both for judging the relevance of news articles and for creating personalized summaries.

Both the work of Billsus & Pazzani and the work of Diaz et al. are related to the project described in this thesis. However, instead of presenting a certain number of complete news articles to the user, the goal of this thesis is to present a single news event. Moreover, a user query is taken into account in addition to a user model (although, as stated earlier, this can be thought of as a short-term model of the user's interests).

Lv et al. [31] put their focus on transitions between related news stories and define a relatedness measure for news stories based on four heuristics: similarity, novelty, connection clarity and transition smoothness. Based on a corpus of news articles labeled by newspaper editors, they used machine learning to combine these four heuristics into a single relatedness measure. The application scenario they have in mind is the presentation of related news stories on an online news site, similar to the "news threads" of Billsus & Pazzani.

Although interesting, their results are not really applicable to the work presented in this thesis as the transition between topics is not taken into account in the first version of the NewsTeller system. It might however be an interesting direction for future development.

2.7. Summarization

Automated summarization of documents has been of research interest for several decades. Usually, the goal is to achieve a high compression rate (i.e., a small ratio of summary length vs. source text length) while still preserving the main pieces of information from the original document. In [26], Jones gives a basic model of the summarization process consisting of three steps. It is shown in Figure 2.8.

This model of the summarization process can also be applied to the work presented in this thesis: The interpretation step is done by the NewsReader system (transforming a

Figure 2.8.: Model of Summarization according to Jones [26].

set of news articles into an RDF graph), whereas transformation (transforming this large RDF graph into an small RDF subgraph about a single event) and generation (extracting the sentence mentioning the selected event from the original news article) are handled by the NewsTeller system.

Hahn & Mani [20] give a list of dimensions according to which different summarization approaches can be distinguished:

- **Extraction vs. Abstraction**: In extraction-based summarization, snippets from the original document are used to create the summary. In abstraction-based summarization, new formulations are created based on some internal semantic representation.
- **Indicative vs. Informative vs. Critical**: An indicative summary is used to judge the usefulness of the original document and can be thought of as a pointer to it. An informative summary is supposed to completely replace the original document. A critical summary goes one step further and also includes an opinion or a judgment of the original text.
- **Knowledge-rich vs. Knowledge-poor**: A knowledge-rich approach to summarization uses domain-specific background knowledge in addition to the source text. A knowledge-poor approach does not make use of any additional resources other than the source text.
- **User-focused vs. Generic**: A user-focused summary takes into account knowledge about the user (e.g., interests or background knowledge) to bias the summarization process. A generic summary does not use any information about the user. User-focused summarization is often used in information retrieval where for each retrieved document a short indicative summary is created which is biased by the user query [50].
- **Single-document vs. Multiple-document**: Single-document summarization deals with a single input document whereas multi-document summarization compiles a set of multiple documents into a single summary.

The work presented in this thesis can be classified in the following way:

- *Hybrid between abstraction-based and extraction-based*: It makes use of the abstract event representation in the KnowledgeStore, but returns an extracted sentence.

- *Informative*: The user will not have any access to the original document.
- *Knowledge-rich*: The system uses knowledge encoded in the KnowledgeStore that was not part of the original news article (e.g., information from DBpedia).
- *User-focused*: It takes into account the user model and the user query.
- *Multi-document*: The event being talked about in the summary might be mentioned in different source documents.

Jones argues in [26] that context factors are important for each summarization task. She distinguishes three groups of context factors characterizing a summarization task:

- **Input Factors**: information about the original text (i.e., its structure, expected readers, etc.).
- **Purpose Factors**: information about how the summary will be used (i.e., in which situation, by which audience, for which purpose).
- **Output Factors**: information about the desired summary (i.e., the compression rate, style, etc.).

In this work, we assume the source to be a set of news articles which are not targeted at a specific audience, that the summary will be used within a social dialog system doing small talk with a single specific user (which can be characterized by a user model) and that the summary only needs to cover a part of the information present in the original documents, giving a short informative summary.

Although the NewsTeller system can be viewed in the summarization context, the process of finding a relevant news event to talk about is far more important than the summarization aspect of the system.

3. System Architecture

3.1. Overall Architecture

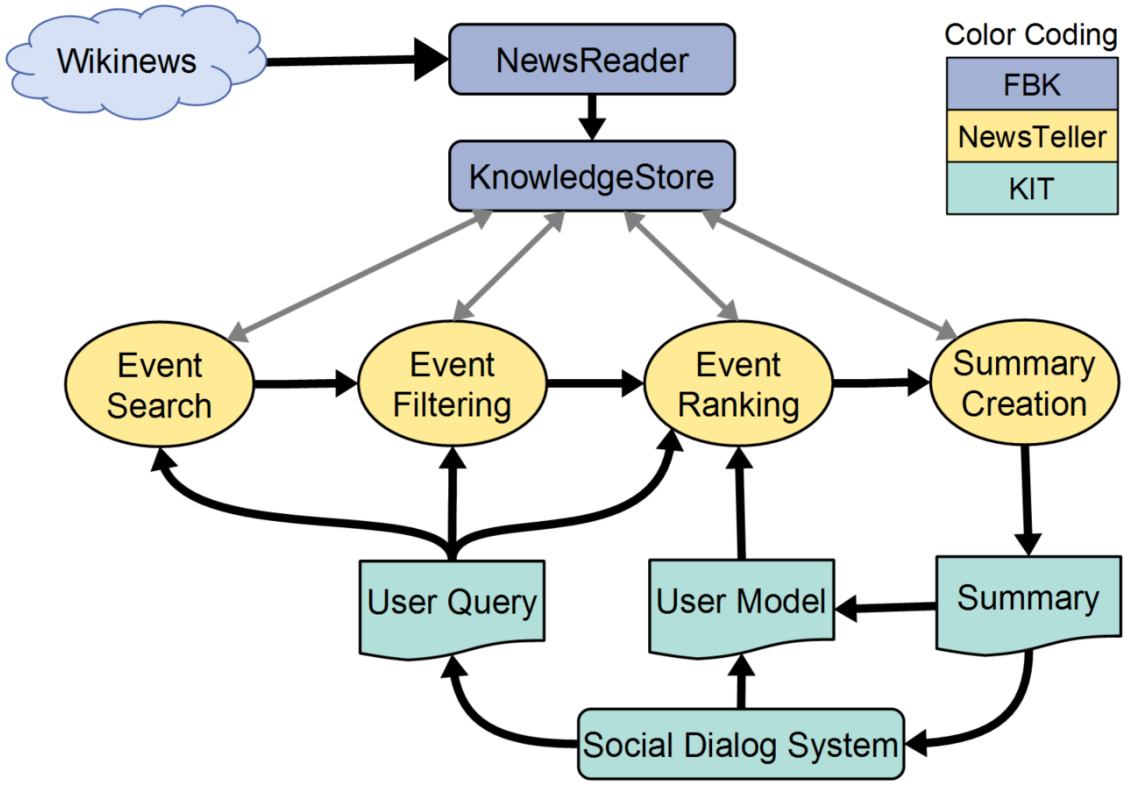

Figure 3.1.: Overall architecture of the NewsTeller system.

Figure 3.1 shows an overall architecture sketch for the NewsTeller system. A quick note about the color coding: Systems developed at FBK are shown in blue, systems developed at KIT in green. All components that were developed as part of this thesis are colored yellow.

Starting from the top of the picture, one can see that the NewsReader NLP pipeline [1, 55] is used to process news articles from the online news source Wikinews [57] and to extract events from them. The results of this extraction process (i.e., the extracted events along with intermediate information) are stored in the KnowledgeStore component [11]. The KnowledgeStore SPARQL and CRUD endpoints are used by various parts of the NewsTeller system to access events, their properties, and the relationships between them as well as mentions, news articles, and their respective properties.

Starting from the bottom of Figure 3.1, one can see the social dialog system developed at the KIT [47]. It takes care of extracting keywords from the user utterance and hands them as user query to the NewsTeller system. It takes the summary that is returned by the NewsTeller system and may use it as output to the user. Furthermore, it provides the NewsTeller system with a user model consisting of a set of keywords that represent the user's interests. Note that also the summaries of previous turns are stored in the user model in order to avoid presenting the same event multiple times.

The task of the NewsTeller system is to create a summary of a news event based on the user query and the user model (both provided by the social dialog system). It does so by using the contents of the KnowledgeStore (which has been populated by the NewsReader NLP pipeline).

The workflow of the NewsTeller system can be broken down into four steps that form a pipeline:

- **Event Search**: fetches a large number of potentially relevant news events from the KnowledgeStore, based on the user query.
- **Event Filtering**: filters the events found in the search step based on their expected usability (i.e., keeps only well-formed events that are suited for further processing). Uses the user query and information from the KnowledgeStore to define features for the filtering classifier.
- **Event Ranking**: ranks the events according to their expected relevance, based on both the user query and the user model. Uses the user query, the user model, and information from the KnowledgeStore to define features for the ranking regressor. Selects the most relevant event.
- **Summary Creation**: creates a summary sentence for the selected event based on a simple sentence extraction approach.

The NewsTeller system was implemented in Java. As both the KnowledgeStore and the ISL Social Dialog system are also implemented in Java, this seemed to be the natural choice. For machine learning tasks, the WEKA framework [21] was used[1]. Moreover, in order to keep the implementation modular and flexible, the dependency injection capability of the Spring framework was used[2]. See Appendix A.1 for a complete list of external resources used to implement the NewsTeller system.

The remainder of this chapter is structured as follows:

Section 3.2 gives some more details about the user model used within the NewsTeller system. Section 3.3 presents the initial task of finding potentially relevant events. Section 3.4 explains the necessity to filter the events found during search and the approach taken to solve this task. Section 3.5 describes the core problem of this thesis – ranking the events according to their estimated relevance – and the approaches for solving this

[1] See also `http://www.cs.waikato.ac.nz/ml/weka/`

[2] See `http://projects.spring.io/spring-framework`

problem. Section 3.6 is concerned with the creation of an output sentence for the selected event. Section 3.7 concludes this chapter by describing the techniques that were used to reduce the system's response time.

All descriptions in this chapter are given on a relatively high abstraction level, highlighting the underlying ideas and approaches. Further implementation details can be found in Appendix A.

One short remark about the terms used in the following sections: We will refer to both actors and places of an event as "entities" and to actors, places, and the event label as "constituents" of an event.

3.2. User Model

The relevance of an event does not only depend on the user query but also on the user's interests: The same event returned for the same query might be very relevant to one user, but practically irrelevant to another user.

Consider for instance two users: User A is mainly interested in economics, whereas user B is mainly interested in soccer. Suppose both of them formulate a query using the keyword "Bayern Munich". An event referring to the annual report of Bayern Munich would be highly relevant for user A but not very interesting to user B. However, an event about the latest soccer match of the club is certainly interesting to user B, but not to user A. And then, there are of course also events that are interesting to both of them (like the expensive transfer of a famous soccer player) or to none of them (some news about the basketball team of this club).

In order to discriminate between users' interests, a user model is therefore needed. The information from this user model is then used to bias the ranking results. It could potentially also be used to retrieve events in a proactive scenario where the system presents a news event to the user without an explicit user query. This possibility was however not explored in this thesis.

The user model used for this project is relatively simple: It consists of a bag of weighted keywords describing the user's interest and a list of events that have been presented to the user in earlier turns. In comparison to some of the user models mentioned in Section 2.6.2, there is no explicit distinction between a long-term and a short-term model. However, one could interpret the keywords from the user model as representation of the user's long-term interests and the keywords from the current query as representation of the user's short-term interests.

For the scope of this thesis, the keywords in the user model are expected to be already given and are not modified throughout the interaction. One possible extension of the system would of course be to infer new keywords or reweigh existing ones based on the interaction history. Moreover, for the sake of simplicity, all keywords have unit weight for all experiments conducted in this thesis. This constraint could be removed in future research.

3.3. Event Search

The first step in the processing pipeline is the search for potentially relevant news events in the content of the KnowledgeStore. This search step is solely based on the keywords from the user query. Information from the user model is not used in the search step. This might however be an interesting direction for future development.

In a first step, only single-keyword queries were considered. Based on this single keyword, originally three different SPARQL queries were used to find potentially relevant events: based on the event label, the participating actors, and the location of the event. However, it turned out that the processing time of the system could be greatly improved by merging these three queries into a single query (see Section 3.7). Only the final approach of using one global query will be further discussed in this section.

Figure 3.2 shows the SPARQL query that is being used to find events. It basically consists of four parts: Line 4 is the first semantic block of the query. It is followed by two blocks in curly braces (lines 6 and 7, and lines 11 to 17) which are connected by a `UNION` keyword. This means that at least one of them must match. Finally, the last line in the `WHERE` clause (i.e., line 19) constitutes the fourth semantic part of the query. Let us look at these four parts in more detail, considering the following example event:

"The Augustine volcano erupted on January 13 2006."

The first part (line 4) is self-explanatory: We only want to retrieve nodes from the RDF-graph that are in the category `sem:Event`, i.e., that are events.

The second part (lines 6 and 7) describes events that match the keyword through their event label. It is used to find the example event for the keyword "erupt". The label of the event is selected (line 6) and a simple check is done to ensure that this label matches the keyword (line 7). This is done by making use of the `bif:contains` relationship which matches if and only if the content of the variable to the left contains the characters specified to the right. Note that `*b*` is a placeholder which is replaced by the keyword before the query is fired.

The third part of the query (lines 11 to 17) describes events that match the keyword through one of their attached entities (i.e., actors or places). It is used to find the example event for the keyword "volcano". It works as follows: The entities attached to the event are selected (line 11). For each of them all nodes reachable by zero or more `rdf:type` links are retrieved (line 12), i.e., the node itself and all its hypernyms (parent nodes in the "is-a" hierarchy). For each of these nodes the corresponding label is selected (line 15) which must match the keyword (line 16). This means that this part of the query only matches events that have at least one entity that matches the keyword either by itself or via one of its ancestors in the 'is-a" hierarchy.

This hypernym structure is included in order to deal with abstract keywords like "comedian" where the entities actually mentioned in news articles are usually referred to by their name (e.g., "Ben Stiller") and not by their profession. The information that these

```
1  SELECT ?event
2  WHERE
3  {
4      ?event rdf:type sem:Event .
5      {
6          ?event rdfs:label ?label .
7          ?label bif:contains '*b*' .
8      }
9      UNION
10     {
11         ?event sem:hasActor|sem:hasPlace ?entity .
12         ?entity rdf:type* ?class .
13         GRAPH <http://www.newsreader-project.eu/modules/dbpedia-en>
14         {
15             ?class rdfs:label ?label .
16             ?label bif:contains '*b*' .
17         }
18     }
19     FILTER(REGEX(STR(?label),"*k*", "i"))
20 }
21 GROUP BY ?event
```

Figure 3.2.: The SPARQL query for the search step.

entities are in the category "comedian" is accessible via the category structure from DBpedia.

Note that with the `GRAPH <http://www.newsreader-project.eu/modules/dbpedia-en>` expression (line 13), the label matching is restricted to the DBpedia subgraph, i.e., the RDF triples extracted from DBpedia. This is necessary to ensure that "stable" entities are selected:

As the NewsReader pipeline is not perfect, it sometimes creates incorrect RDF triples including ones with incorrect `rdfs:label` information. This means that sometimes an incorrect label is attached to an entity. For example, the entity `dbpedia:Michael_Jackson` has (among others) the label "George Michael". This label was (erroneously) attached to the entity by the NewsReader and is thus part of the `http://www.newsreader-project.eu/instances` subgraph. If the query did not restrict the label matching to the DBpedia subgraph also this incorrect label would be taken into account. This can be a problem because an entity with a single incorrect label may participate in many events – then the query would return many irrelevant results. In the given example, when querying for "George Michael", all events in which the entity `dbpedia:Michael_Jackson` participates would be returned as a result – which is clearly not desired. By restricting the label matching to the DBpedia subgraph this problem is circumvented: It is assumed that all labels extracted from DBpedia are correct. Potentially erroneous labels attached by the NewsReader during its processing are therefore ignored. As most users will probably be

interested in well-known entities (which supposedly have a Wikipedia entry and thus are part of the DBpedia corpus), this solution seems to be not too restrictive.

The fourth part of the query (line 19) requires that the label which was found in the second or third part matches a regular expression (regex) derived from the keyword. This regex replaces the placeholder `*k*`. The regex match itself is case insensitive (indicated by the parameter `"i"`).

Both the regex match and the `bif:contains` checks are not performed on the original keyword, but on its stem (as derived by the snowball stemmer [9]). This is done to allow for more matches: For instance, given the keyword "election", its stem "elect" will be used for all search queries. Thus, also events labeled with "elect" can be found and not only ones that involve an entity labeled as "election". The `bif:contains` is used as a first rough filter (reducing query time as less potential matches need to be checked by the regex) whereas the regex match is a more precise filter applied in a second step. It is necessary to perform this second filtering step when the keyword consists of more than one word (e.g., "United States"): The `bif:contains` only checks if all parts of the keyword appear in the label, and the regex makes sure both that they appear in the correct order and that they immediately follow each other. As the regex match is however relatively slow, it makes sense to apply it only to events pre-filtered by the considerably faster `bif:contains`.

Overall, the workflow of the event search component can be described as follows:
Stem the keyword, insert its stem into the SPARQL query, send this query to the KnowledgeStore, and collect all results. This resulting set of event URIs is then passed on to the filtering component.

So far, we only considered the case of a single keyword. In order to deal with multiple keywords, the placeholders `*k*` and `*b*` in the SPARQL query from Figure 3.2 are replaced by more complex expressions that include the stems of all keywords from the user query. This proved to be considerably faster than executing multiple queries – even when executing them in parallel.

For some keywords (e.g., "Star Wars"), the `bif:contains` does not work properly. In these cases, the query returns an empty set of events. In order to deal with this problem, a fallback query is used which is identical to the query from Figure 3.2 except that lines 7 and 16 are missing. It is slower than the standard query and is thus only used if the standard query fails.

Some keywords can yield several thousands of potentially relevant events (e.g., "sport" yields 11,967 events, and "United States" yields 53,220 events). As the processing time of the subsequent components heavily depends on the number of events they need to handle, a random filtering is performed: If the total number of events collected in the search step is more than 1,000, only 1,000 of them are kept for further processing. These events are selected randomly. This is done to ensure a reasonable response time to user queries. The

general idea of this random filtering is that in the end the system only needs to return the description of *one* relevant event to the user. As 1,000 events can be expected to contain a sufficient amount of relevant events, it is acceptable for our purposes to only consider 1,000 events for further processing. Although we might throw away very good events at this point, the reasoning is that there is still a sufficient amount of them contained in the data that is passed on to subsequent components.

Of course, an obvious starting point for further work would be to replace this random selection with a more targeted selection based on a simple heuristic.

3.4. Event Filtering

Just like any NLP system, the NewsReader pipeline is not perfect. This means, that not all events extracted by the pipeline and stored in the KnowledgeStore are suitable for further processing. For example, some actors might be missing (e.g., a "contradict" event only with a `propbank:A0` link but without a `propbank:A1` link – we do not know who or what was contradicted), two events might accidentally be merged into one, or a named entity might get misrecognized (e.g., extracting the entity "Michael Jackson" from a text where a person named "Michael" is mentioned). Moreover, the stemming done in the search step might sometimes also introduce additional noise: When searching for "Hawking" (referring to the physicist Stephen Hawking), the stemmer returns the stem "Hawk" – which unfortunately matches the often-mentioned Hawk helicopter.

As the search step is "blind" in the sense that it simply returns everything it can find, a subsequent filtering step is necessary to eliminate events of low quality. One could of course skip this step and make sure that the low-quality events will be ranked last in the following ranking process. However, as the number of malformed events is surprisingly high (about 84 % in a data set of about 6,000 events), it seems like this approach might introduce too much noise into the ranking problem. Therefore, we decided to first apply a filter before ranking the events. This filtering is performed by a classifier.

The following subsections describe the data collected for the filtering task (Section 3.4.1), the features and classifiers being used (Section 3.4.2) and the results obtained on the data set (Section 3.4.3). For reasons of simplicity, only queries with one keyword are considered at first. Section 3.4.4 shows how the one-keyword approach generalizes to multiple keywords.

3.4.1. Data Set

In order to train and evaluate a classifier on the usability of the discovered events, a data set was created: A total number of 46 keywords were used to find news events and every found event was labeled as Usable or Not Usable. This data set contains 6,084 labeled events, of which 974 are Usable – which corresponds to 16.01%. Table 3.1 shows some queries from this filtering data set.

As it turned out during the creation of this data set, one can distinguish different reasons why an event should be considered Not Usable. The different reasons for non-

Query Characteristic	Query Keyword	Number of events	USABLE events
Query with most events	comedy	402	4 (11.19%)
Query with least events	GermanWings	5	2 (40.00%)
Query with highest fraction of USABLE events	erupt	211	120 (56.87%)
Query with lowest fraction of USABLE events	Chernobyl	346	6 (1.73%)
Average query	N/A	133	21 (16.01%)

Table 3.1.: Table showing the some queries from the filtering data set.

usability are shown in Table 3.2 and were used to further annotate all non-usable events. About 50% of the events which are considered as NOT USABLE fall into two or more of these categories.

When looking at Table 3.2, one can see that two different types of reasons are distinguished: syntactic and semantic reasons.

If an event is considered NOT USABLE due to some syntax issues, this means that the syntactic structure expressed in the RDF representation (i.e., actors, places, propbank arguments) does not correspond to the syntactic structure of the original sentence. For instance, the MISSING SUBJECT category indicates that the NewsReader pipeline failed to extract the subject of the event although it is contained in the original sentence.

On the other hand, if an event is considered NOT USABLE because of some semantic problems, this means that the semantics represented in the RDF graph do not correspond to the semantics inherent in the underlying sentence. An example would be the KEYWORD ENTITY CATEGORIZATION class: The entity that matched the query keyword (e.g., "Michael Jackson") does not appear in the original sentence. Usually, another entity is mentioned in the original sentence (e.g., a different person named "Michael") that was erroneously linked to the keyword-matching entity.

Note that although syntactic problems can impair further processing (when information in the RDF representation is e.g., missing or superfluous), semantic problems are much worse from the users' perspective: Users will not notice if syntactic problems occurred during the processing as they never deal with the RDF representation of an event. They will however immediately spot semantic errors when the system's output is not related to their initial query at all. Therefore, the filtering of semantically wrong events is more important than the filtering of events with syntactic errors. However, we trained a single classifier for filtering out both syntactically and semantically impaired events instead of training two specialized classifiers. We briefly investigated the latter approach but it turned out to be inferior to a joint classification.

Figure 3.3 visualizes the frequency of the different reasons for non-usability in the data set. See Appendix A.3.1 for a list of all queries used for creating the filtering data set.

Name	Description	% total	% only
Syntax			
Wrong Parse	The attachment of entities to events is incorrect (e.g., an event has actors that belong to a different event).	20.30%	6.24%
Overlapping Constituents	Some of the event constituents overlap in the text (e.g., an event with label "declare" and actor "to declare bankruptcy").	17.62%	3.17%
Missing Object	An object of the event was not extracted from the text (e.g., a "contradict" event misses its A1 argument).	17.40%	9.29%
Broken Entity	An entity was not correctly extracted: either with incorrect boundaries or as incorrect entity type (e.g., a place being extracted as actor).	10.31%	3.13%
Missing Subject	The subject of the event is not extracted from the text (e.g., a "beat" event without A0 argument).	7.25%	3.09%
Semantics			
Keyword Entity Categorization	The entity matching the keyword was incorrectly categorized or matched (e.g., "the accident" is encoded as an instance of `http://dbpedia.org/resource/Chernobyl_disaster` although it refers to a traffic accident).	36.04%	9.82%
No Event	The event label refers to something that is not an event (e.g., "spokesman").	30.25%	9.23%
Event Merge	Two separate events were accidentally merged into one.	15.92%	3.81%
Keyword Regex Mismatch	The entity matching the keyword stem is not related to the original keyword (e.g., "Black Hawk helicopter" matching the stem of "Hawking").	4.85%	0.72%
Other Entity Categorization	An entity different from the one matching the keyword was incorrectly categorized (e.g., a person named "Clint Brown" is matched to the entity `http://dbpedia.org/resource/Freddie_Brown_(cricketer)`).	3.87%	0.84%

Table 3.2.: Table showing the different classes of non-usable events. The column "% total" shows the percentage of non-usable events in the benchmark that fall into the respective class. The column "% only" indicates the percentage of non-usable events in the benchmark that only belong to the respective class and not to any of the other classes.

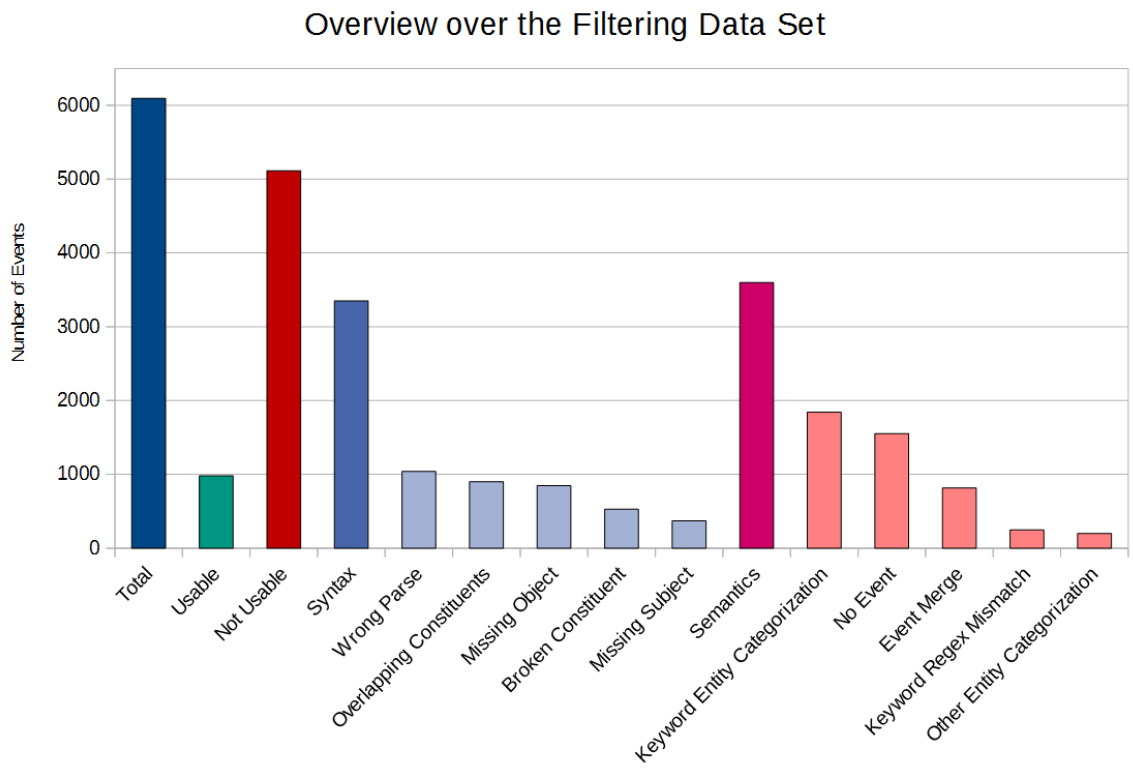

Figure 3.3.: Diagram showing the frequency of the different subclasses of the NOT USABLE class in the filtering data set.

3.4.2. Approach

In order to solve the filtering problem, we briefly entertained the idea of simply extracting complete subgraphs (i.e., the event plus a neighborhood of a fixed size) and performing structural learning on those graphs. This means that an event would be classified into the USABLE or NOT USABLE class based on the structure of its RDF graph.

There are basically two approaches to graph classification [51]: The first approach identifies frequently appearing subgraphs and uses their presence or absence in a specific graph as binary feature. The second approach uses kernel-based classification approaches. Kernel-based classification needs a kernel function that acts as a similarity measure for two given training examples. This graph-kernel is usually based on the size of the intersection graph (i.e., the largest common subgraph shared by both input graphs).

As one can see, both approaches require finding common subgraphs – either in the full training set (in order to use them as features) or for two given graphs (to define a graph-kernel). Most of the existing approaches for finding common subgraphs try to find exact matches – they assume that there is a relatively low number of different labels for both nodes and links and that all of them occur quite frequently in the data. This is a reasonable assumption when dealing with chemical molecules (where most of the graph classification research is coming from), but in the case of an RDF graph this assumption does not hold: Each node is identified with its unique URI, therefore the number of different labels is high and each label appears only rarely in the data. Hence, the existing subgraph

finding algorithms and therefore also the established graph classification algorithms cannot be used out-of-the-box in this context and would require significant modifications. Therefore, we did not investigate this approach any further.

Instead, we defined different features based on the query keywords, the RDF representation of the events, and the underlying news articles. These features ranged from rather simple ones, e.g., the number of `propbank:A1` links, to more complicated ones, e.g., counting the maximum number of words in the original sentence (e.g., "Michael Jackson, also known as 'King of Pop', died yesterday in his home.") that separate the anchor of the event (e.g., "died") from one of its entities (e.g., "Michael Jackson"). The features can be classified with respect to the three KnowledgeStore layers on which they are defined as well as with respect to the three categories "Query-Event", "Event" and "Query" (which correspond to the "query-document", "query-independent", and "document-independent" categories from Section 2.4). Some example features (including their classification according to the two dimensions mentioned above) are listed below:

- **A1**: counts the number of `propbank:A1` links. Uses only the entity layer of the KnowledgeStore and belongs to the "Event" category (the query keywords are not taken into account at all).
- **POS**: retrieves the mentions of the event and determines whether the they were tagged as verbs. Uses both the entity layer (to retrieve the mentions) and the mention layer (to retrieve the part of speech information). Is also an example of an "Event" feature.
- **appearKeywordLabelsInSentence**: retrieves the DBpedia labels of the entities matching the keyword stem and checks whether these labels appear in the sentence from which the event was extracted. Uses the entity layer to retrieve the labels and the resource layer to retrieve the original text. Makes use of the query keyword by looking only at entities matching this keyword, and can thus be classified as a "Query-Event" feature.

A list of all features being used in the final classifier with short explanations is given in Appendix A.3.2.

An initial set of features was developed based on general intuitions about the properties of a Usable event. However, these features were not yet sufficient to achieve good performance. Therefore, further feature engineering was performed, looking at each of the reasons for non-usability individually. Focusing on a specific part of the overall problem made it easier to identify features that could be helpful to distinguish the respective subclass of the Not Usable class from the overall Usable class.

As the feature engineering activity yielded a plethora of candidate features (over 90 in total), many of which were variants of each other, it was necessary to perform feature selection. The need for a much smaller number of features is based on two main arguments:

First and foremost, the well-known "curse of dimensionality" implies that the number of necessary training examples grows exponentially with the number of features. This

is necessary to make sure that the feature space is sufficiently covered in order to avoid overfitting. As the data set created for this classification task is limited in size (it consists of only about 6,000 training examples), the number of features used for training the classifier should be as low as possible to ensure that the classifier does not overfit the data.

The second motivation for a smaller set of features is based on runtime considerations: As the NewsTeller system is supposed to interact with users in an online fashion, processing time of the different components is a major constraint. Because the features cannot be extracted prior to the search step, and because the calculation of each feature value takes some time, reducing the number of features will also help to reduce the overall response time of the system.

Based on this second motivation, we decided not to use dimensionality reduction methods like Principal Component Analysis (PCA): Although dimensionality reduction helps to reduce the number of features for the classifier, it does not reduce the amount of features that need to be computed in the first place.

Regarding feature selection, one can distinguish three approaches:

- *Filter methods* compute some statistical metrics on a feature set (or a single feature). This statistical metric is then used as indicator for the quality of the feature set. An example would be computing the correlation between a feature and the class label.
- *Wrapper methods* make use of a classifier: They train the classifier using different subsets of the overall feature set and use the classifier's performance as indicator for the quality of the respective feature set.
- *Embedded methods* make also use of a classifier, but use classifier-internal information (e.g., weights associated with the different features) to judge the importance of different features. This is only applicable to some classifiers.

For each of the three approaches, there are different algorithms and variants that can be used. As each of them might return a different subset of features, we used a number of different feature selection algorithms in parallel. The underlying assumption is the following: If many different feature selection algorithms agree that a certain feature is important (i.e., belongs to their top k list of features), then this feature should be useful in general (as it was selected multiple times, based on different points of view). See Appendix A.3.3 for a list of feature selection algorithms that were used for the filtering problem. After this application of multiple automated feature selection methods, a wrapper-based approach was used to further reduce the number of features.

As noted earlier, only about 16% of the events belong to the Usable class. This high class skew makes the classification problem difficult, as a classifier will be heavily biased towards the majority class: A classifier that always predicts Not Usable will already have an accuracy of 84%. The problem of imbalanced data is a quite general one, and He and Garcia [23] provide a good introduction into the topic. To deal with this problem in the scope of this thesis, three different techniques were explored, namely undersampling, oversampling, and cost-sensitive classification:

- *Undersampling* tries to mitigate the class skew by throwing away examples from the majority class. This however comes at the cost of reducing the overall size of the data set.
- *Oversampling* approaches the problem by creating additional examples for the minority class – either by duplicating existing instances (which might lead to overfitting) or by interpolating between existing instances (which is computationally more expensive).
- *Cost-sensitive classification* does not modify the data set itself, but imposes a cost matrix on the classification problem: If the minority class is the positive class (as in the filtering problem), one can assign a higher cost to false negative classification errors than to false positive errors. The classifier can then be trained to minimize the overall cost instead of maximizing the accuracy.

Of course, it is also possible to combine these approaches.

For training a usability classifier, we followed two main approaches: training a global classifier (i.e., only on the coarse-grained Usable – Not Usable distinction) and training an ensemble of specialized classifiers (i.e., one classifier per non-usability reason). We will now describe each of these approaches in some more detail.

For training a global classifier, a set of eleven different classifiers (ranging from random forests and decision tables over logistic regression and Bayes nets to support vector machines and multi layer perceptrons) was applied to the problem. As it turned out, tree-based approaches (especially random forests) performed very well on the data. As random forests were consistently showing a very good performance across different feature sets and different oversampling/undersampling/cost-matrix configurations, they were selected for further optimization.

Moreover, based on the subclasses of the Not Usable class, specialized classifiers were trained: For each subclass of Not Usable, a classifier was trained that tries to distinguish Usable examples from examples of the respective subclass. These specialized classifiers were then combined in an ensemble to solve the overall usability classification problem. Each specialized classifier was expected to filter out a part of the non-usable events (the ones corresponding to the respective subclass). Of course, one can expect that due to the accumulation of misclassifications, the performance of the ensemble might be considerably worse than the performance of its individual classifiers. However, as about half of the non-usable events belong to at least two subclasses of the overall Not Usable class, it was hoped that this would somewhat mitigate the performance degradation. We also hoped that individual specialized classifiers would be able to capture the respective subproblems much better than an overall classifier would be able to capture the global problem. Therefore, the ensemble might even in the face of accumulating errors perform better than a global classifier.

There were two ways in which the individual classifiers were combined: One was based on accumulating the class probabilities from the individual classifiers and another one was

based on a hard filtering. The accumulation of class probabilities was done by taking the minimum, average, maximum, and the product, respectively. The hard filtering was based on the classification decision of the individual classifiers: An event was only classified as Usable by the ensemble if all individual classifiers agreed on this classification (basically implementing a logical AND function on the positive class). As soon as one classifier classified an event as Not Usable, the overall classification would also yield Not Usable.

This hard filtering was expected to yield inferior performance compared to the approach based on probabilities as the number of false negatives can be expected to be much higher. If a single classifier makes a false negative misclassification, the overall classifier will also make this false negative error. However, the hard filtering approach has a certain runtime advantage: If the first classifier outputs Not Usable, the classification result is already determined and the classification of the other classifiers does not need to be gathered. We reasoned that this might save some time compared to always using all classifiers to classify an event. Therefore, it was hoped that the hard filtering would yield a reasonable tradeoff between classification time and classification performance compared to the accumulation-based approaches. Out of the accumulation approaches, the minimum-based one is semantically closest to the hard filtering.

Note that the majority vote approach (which is often used for classifier ensembles) is not applicable to this problem: The different classifiers are specifically targeting different reasons for non-usability. Therefore, if for example the event is Not Usable because its label does not describe an event (e.g., "spokesman"), it is likely that only the No Event classifier will classify it as Not Usable; the other classifiers focus on other aspects which might be all fine, and might thus all output Usable. The majority vote rule would then yield Usable – which in this context does not seem very intuitive. Therefore, the majority vote was not used for the combination of classifier outputs.

For the specialized classifiers, feature selection was performed on the features specifically engineered for the respective subclass of Not Usable. The union of all these feature sets was used as feature set for the global classifier. After this, for both approaches a hyperparameter optimization was performed on both the classifier parameters (in case of the random forest e.g., the number of trees) and the respective cost matrix.

3.4.3. Results

When evaluating classifiers on an imbalanced data set, one should avoid the standard accuracy measure because it can be misleading: As already mentioned, for the usability data set being worked with, a simple "always no" baseline will be correct in most of the cases and reach an accuracy of 84%. An accuracy of over 80% looks like the problem is close to being solved, but always predicting the majority class is clearly not a satisfactory result. In the given application, this would mean that not a single event would be left after the filtering step – this is clearly undesirable. Therefore, three other performance metrics have been used. They are based on the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN), as well as on the accuracy metric:

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

The additional performance metrics are the following:

- F_1 **measure**: $F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$

  The F_1 measure (also called F_1 score) is the harmonic mean of $precision = \frac{TP}{TP+FP}$ and $recall = \frac{TP}{TP+FN}$ and is coming from an information retrieval background.

- **Balanced accuracy**: $BalancedAccuracy = 0.5 \cdot \frac{TP}{TP+FN} + 0.5 \cdot \frac{TN}{TN+FP}$

  The balanced accuracy is the unweighted average of the recall for the positive class and the recall for the negative class. By putting equal emphasis on both classes irrespective of their frequency in the data set, it is very insensitive to class skew.

- **Cohen's kappa**: $\kappa = \frac{p_0 - p_e}{1 - p_e}$

  Cohen's kappa adjusts the classifier's accuracy p_0 by taking into account the probability p_e of random agreement between the classifier and the ground truth. This probability of random agreement is computed based on the prior class probability in the data and the overall fraction of predictions made by the classifier for each class. An "always no" classifier in an imbalanced data problem will have a high probability of random agreement with the ground truth. Therefore, its kappa value will be small.

All three of these performance measures are thought to more accurately reflect classifier performance on an imbalanced data set than the standard accuracy measure.

Let us first take a look at the individual classifiers trained for the ensemble approach before comparing the performance of the overall ensemble with the global classifier. Table 3.3 shows the results obtained in a ten-fold cross-validation for each of the non-usability subclasses. In cases of imbalanced data, cost-sensitive classification was used to mitigate the class skew and improve classifier performance. Appendix A.3.2 lists the features used for each individual classifier.

As one can see in the "Classifier Type" column of Table 3.3, also for the specialized classifiers tree-based approaches often yielded the best performance.

It also seems that there are some relatively easy classification problems: For Overlapping Constituents, a threshold classifier on a single feature already yields 98% of balanced accuracy, and also the Event Merge and Keyword Regex Mismatch classifiers achieve more than 90% of balanced accuracy. This can be explained by the availability of expressive features: For instance, Overlapping Constituents can easily be detected by looking at the event constituents and simply checking their mentions for overlaps.

On the other hand, there seem to be also some relatively hard classification problems, namely Other Entity Categorization, Wrong Parse, and Broken Entity. For each of these three problems, the balanced accuracy is below 75% and also the rather low value of Cohen's kappa indicates that the classifiers do not perform well. One explanation for the low performance in these cases are the depth of the problems and the lack of expressive features for them: For instance, the Wrong Parse subclass contains events that have e.g., superfluous entities, which is probably caused by a bad parse of the sentence. However, parsing itself is a complex and deep problem and detecting parsing errors using only

Subclass	Classifier Type	Accuracy	Balanced Accuracy	Cohen's kappa	F_1 score
Semantics					
Keyword Entity Categorization	RF	0.8873	0.8674	0.7469	0.8313
No Event	RF	0.9131	0.8961	0.8121	0.8795
Event Merge	DT	0.9434	0.9463	0.8867	0.9463
Keyword Regex Mismatch	DT	0.9324	0.9137	0.8006	0.9568
Other Entity Categorization	AB	0.8729	0.7376	0.5131	0.9249
Syntax					
Wrong Parse	RF	0.7447	0.7457	0.4900	0.7439
Overlapping Constituents	TH	0.9803	0.9805	0.9605	0.9808
Missing Object	RF	0.8500	0.8490	0.6991	0.8584
Broken Entity	DT	0.7198	0.6399	0.3113	0.8076
Missing Subject	DT	0.8638	0.8368	0.6641	0.9051

Table 3.3.: Table showing the performance of the respective best classifier for the different subclasses of non-usable events. The abbreviations in the "Classifier Type" column have the following meanings: **R**andom**F**orest, **D**ecision **T**ree, **A**da**B**oost, and **TH**reshold.

shallow features (like the maximum number of words between an entity and the event itself in the original sentence) is apparently not possible. One could argue that the use of more complex features might help to boost performance in these cases. However, the computation of more complex features (e.g., re-parsing the sentence and comparing the resulting parse tree to the constituents of the event) requires more time. And as we already have argued earlier, total processing time is a considerable constraint in the setting considered in this thesis. Therefore, no further feature engineering was performed.

As the overall performance of the individual classifiers seems to be acceptable, they were combined into an ensemble as described in the previous section.

Table 3.4 compares the overall results for the Usable – Not Usable classification task that was obtained by the different approaches. For each classifier, cost-sensitive classification was used for tuning purposes. For the ensemble approaches based on probability aggregation, only the results for the best aggregation rule are reported, which is the aggregation by using the product of the probabilities. The global classifier being used is a random forest with 50 trees, each of which has a maximum depth of 20 nodes and uses a maximum of 15 features.

As a lower bound, also the performance of two simple baselines is given in the table. The "Random 50:50" baseline predicts both classes with equal probability, whereas the

Classifier	**Acc**	**BA**	κ	F_1	**Prec**	**Rec**
"Random 50:50" baseline	0.5000	0.5000	0.0000	0.2425	0.1601	0.5000
"Always No" baseline	0.8399	0.5000	0.0000	NaN	NaN	0.0000
Ensemble (Product)	0.2314	0.5933	0.0984	0.4315	0.2757	0.9919
Hard Filtering	0.8524	0.7497	0.4764	0.5649	0.5349	0.5986
Global Classifier	0.8794	0.7645	0.5411	0.6125	0.6304	0.5955

Table 3.4.: Table showing the overall classifier performance on the Usable – Not Usable problem for the different approaches. The columns show the following metrics (from left to right): **Acc**uracy, **B**alanced **A**ccuracy, Cohen's κ, F_1 score, **Prec**ision, and **Rec**all.

"Always No" baseline always predicts Not Usable. As mentioned before, the "Always No" baseline reaches a high value for accuracy, but does not perform well with respect to the other given metrics.

As one can see, the ensemble approach using an aggregation of probabilities performs notably worse than the other two classification approaches. However, it is still better than both baselines with respect to all metrics except accuracy. It reaches a very high recall, but its relatively low precision outweighs this by far. The hard filtering approach is definitely better, but still in all respects worse than the approach of using a global random forest. Even for this global classifier, the best trade-off between precision and recall that can be achieved is around 60% for each of these measures. This means that 59.55% of the positive events are classified as positive, and 63.04% of the events that are classified as positive are indeed positive examples.

Why does the global classifier preform better than the best combination of specialized classifiers? Different factors might play a role:

- *Accumulation of errors*: As stated earlier, the false negative errors made by the individual classifiers in the hard filtering approach accumulate. In order to mitigate this problem, the individual classifiers were optimized for high recall. As there is however always a trade-off between precision and recall, this causes their precision to drop. This means that more non-usable events are labeled as Usable by each of the classifiers – and these false positive errors also accumulate: On average, a non-usable event belongs to 1.64 subclasses. That is, there are on average 1.64 specialized classifiers that are supposed to identify it as Not Usable. If they all commit a relatively high number of false positives, a considerable amount of non-usable events might "slip through" and hence be classified as Usable.
  Therefore, the accumulation of both false negatives and false positives might impair the performance of the "hard filtering" approach.

- *More training data*: The global classifier is trained on the overall training set whereas the specialized classifiers are all trained on considerably smaller data sets (the subsets containing the positive class and the respective negative subclass). This dif-

ference in the amount of available training data might contribute to the observed difference in performance.

- *Random forest classifier*: The random forest classifier being used for the global approach is an ensemble of random decision trees. Therefore, it also has the advantages of an ensemble-based approach. However, it differs from the specialized classifier ensembles in two ways: Firstly, the ensemble members are not trained on a specific subproblem using specifically engineered features, but on a random subsample of the data set using a random subset of features. Secondly, the number of ensemble members is considerably higher (50 vs. 10). These two differences might allow the random forest to generalize better beyond the training data, as it explores more configurations of its ensemble members.

- *Feature synergy*: Each of the specialized classifiers is trained using a small set of one to six features. The global classifier however can make use of all features. Some of these features might be useful also for other subproblems than the one for which they were originally selected. The global classifier may be able to make use of these "synergy effects" whereas the specialized classifiers can only use their respective small feature sets.

A second question arises when looking at the results: Why does the hard filtering perform better than all ensemble approaches that are based on an aggregation of probabilities?

The answer seems to be less straightforward in this case. Perhaps the way in which the overall classification problem is divided into subproblems naturally lends itself towards a combination by a hard AND. Moreover, the individual classifiers might not be good at accurately predicting probabilities (which in turn might be due to the small size of the respective training sets), but still perform reasonably well in making a binary decision. More work would be required to investigate this issue.

Although better than its competitors, the overall performance of the global random forest classifier is still far from perfect. Again, our explanation for this mediocre performance is the fact that we try to solve a deep problem using only a limited set of shallow features. As the filtering problem is only a pre-processing step to improve the data quality for the ranking problem, we decided to accept the performance level achieved so far. The global random forest was selected as final classifier to be used in the system as it yielded both the best performance as well as a reasonable processing time. It was re-trained on the whole data set before being incorporated in the system.

In order to have some realistic training data for the ranking problem, the data set created for the filtering problem was filtered with the filtering classifier. However, we did not use the version that was re-trained on the whole data set because this would yield too optimistic results (as evaluation on the training set always does). Therefore, we used the classifications obtained during the cross-validation of the selected classifier to filter the data set. The results were both used to analyze the effect of the filtering on the data

Data Set Property	Before	After
Number of Events	6094	926
Percentage of Usable Events	16.01%	63.04%
Average Number of Events per Query	~130	~20
Average Number of Usable Events per Query	~20	~12
Average Number of Reasons for non-usable Events	1.64	1.18
Percentage of semantic reasons for non-usability	70.38%	37.35%
Percentage of syntactic reasons for non-usability	65.52%	80.59%

Table 3.5.: Table showing the effects of the filtering step as before-after comparison.

	Before	After
1.)	Keyword Entity Categorization (ca. 36%)	Wrong Parse (ca. 33%)
2.)	No Event (ca. 30%)%	Broken Entity (ca. 23%)
3.)	Wrong Parse (ca. 20%)	Missing Object (ca. 18%)

Table 3.6.: Table showing the three most frequent subclasses of the Not Usable class before and after applying the filtering.

set quality as well as for creating an initial data set for the ranking problem (see Section 3.5.1.1).

Table 3.5 shows a before-after comparison of the filtering data set. As one can see, the number of events is greatly reduced while the percentage of Usable events increases from 16.01% to 63.04%. After filtering, there are about 20 events per query left, 12 of which are Usable. The fact that the number of annotated reasons for non-usable events drops from 1.64 to 1.18 indicates that the classifier is successful at filtering out non-usable events that have more than one "problem". One can also see that the percentage of semantic reasons among the non-usable events is almost halved whereas the fraction of syntactic reasons increases. This indicates that the crucial semantic problems (when the retrieved event is incorrect on a semantic level) are being filtered out to a large degree. Moreover, when looking at the top three reasons for non-usability (see Table 3.6), also a shift from semantic to syntactic problems can be observed. It is also not surprising that Wrong Parse and Broken Entity, two of the difficult subclasses identified earlier in this section, are appearing in the top three after filtering.

Finally, one should also remark that two queries ("Rhine" and "Himalaya") that had a small number of events (15 and 16, respectively) are left without any events after filtering. Although they contained some Usable events (one and three, respectively), all events of these queries were classified as Not Usable. However, we think that this will not be a big issue in the final system, as most user queries are expected to yield a large number of events. Thus, still a considerable amount of them should "survive" the filtering.

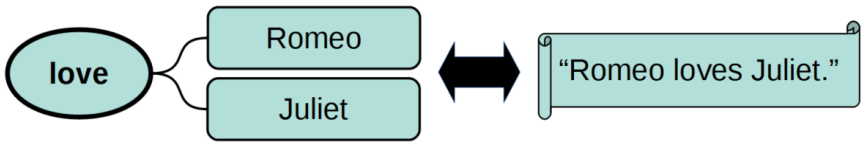

Figure 3.4.: Illustration of an event belonging to the Not Usable class for all queries involving “Rome” due to a Keyword Regex Mismatch.

3.4.4. Multiple Keywords

So far, we only considered queries containing a single keyword. However, the final system will also have to deal with queries containing multiple keywords. It is hence necessary to generalize the given filtering approach to multiple keywords.

Fortunately, there is a straightforward solution to this problem:

As we are not concerned with *ranking* the events, yet, but only with a basic sanity check, we do not need to consider interactions between keywords. The result of the search step for a query consisting of multiple keywords equals the union of the search results for each individual keyword (if we ignore the random filtering at the end of the search step). This set equality means that each of the events in the input of the filtering step was found by matching *at least one* keyword. So when for example computing the *keywordInText* feature, it is sufficient to check whether the labels of *any* of the keywords appears in the original news article. This means, that the value of this feature is computed for every single keyword individually, and the maximum of these value is taken (which corresponds to the logical “OR” operation).

This approach has two main advantages: Firstly, the output of the filtering step for multiple-keyword queries corresponds to the union of the filtering results for the respective single-keyword queries. Secondly (and more importantly), the classifier does not need to be re-trained and no additional data needs to be gathered – the single-keyword filter can be used “as is” without major modification.

However, one should note that this simple solution does not come without drawbacks: For example, consider the event depicted in Figure 3.4. Looking at the event itself, it looks like it should belong to the Usable class as it correctly reflects the underlying sentence. So for the query “Juliet”, for instance, this event should be considered Usable. However, when querying for the Italian capital “Rome”, due to the stemming and regex matching process, “Romeo” gets matched to the keyword “Rome” – which is semantically clearly incorrect. Therefore, for all queries involving “Rome”, this example event should be treated as Not Usable due to a Keyword Regex Mismatch. This can be detected for example by checking whether the original keyword appears in the sentence as its own word (as done by the *keywordInTextFeature*).

Consider now a query consisting of the keywords “Rome” and “Juliet” that yields the event from Figure 3.4. Clearly, with the “Juliet” part everything is fine – but there is a problem with the “Rome” part. By taking the maximum over all keywords, the *keywordInText* feature (which outputs a value of 1.0 for “Juliet” and a value of 0.0 for “Rome”) will return

a value of 1.0, indicating that everything is all right. In this case, the classifier cannot detect that this event should belong to the NOT USABLE class for the given query.

A straightforward remedy for this problem would of course be the use of the minimum instead of the maximum when aggregating over different keywords. This however would cause some other problems down the road: For instance, if there are no events involving both "Juliet" and "Rome", it might be a good strategy for the NewsTeller system to respond with an event involving only one of them. However, if all features during filtering are computed by taking the minimum across keywords, the resulting feature values would probably indicate to the classifier that all of the events for this query are NOT USABLE – we would then end up with an "always no" classification. This issue would arise due to the interaction of keywords introduced by taking the minimum, an interaction which should not yet be considered at this stage.

As we expect the described kind of errors to happen only rarely, we decided to implement the simple solution of taking the maximum, accepting that some non-usable events might pass the filtering without being detected. We think the advantage of not having to create a new data set, modified features and a new classifier outweigh the drawbacks of a slightly higher risk for false positives. It might however be a worthwhile target of future research to investigate how frequent the described types of errors are in practice and whether a more complex way of handling multiple keywords in the filtering step has considerable advantages.

3.5. Event Ranking

After filtering the events with respect to their well-formedness, one event must be picked that will be returned to the user. A simple baseline would be to pick a random one, but this does not seem to be satisfactory: Different events have a different degree of relevance to the user, and talking about more relevant events will supposedly lead to a higher user satisfaction. Therefore, it makes sense to sort the events based on their relevance. As the true relevance of unseen events is unknown, it must be estimated. This relevance estimation was carried out with the use of machine learning techniques.

As the interaction between different keywords is crucial for the ranking process (a relevant event should match as many of the given keywords as possible), this problem was not simplified by first looking at single-keyword queries. Instead, we directly worked on multiple-keyword queries in order to directly define features based on sets of keywords. The ranking functionality was developed in two main steps: first looking at multiple-keyword queries without taking into account information from the user model (Section 3.5.1), and later adding features based on the user's interests (Section 3.5.2).

3.5.1. Multiple Keywords

The first version of the ranking was developed assuming that there is only a set of query keywords but no information from the user model. As already mentioned above, machine learning was used to estimate the relevance of the found (and filtered) events. In order to

use machine learning, annotated data is necessary. Therefore, the events from the filtering data set were also annotated with their relevance rank (see Section 3.5.1.1).

As already discussed in Section 2.4, there are different approaches for the "learning to rank" problem. The pointwise approach can simply be implemented by doing regression on the relevance value and then ordering the events according to their estimated relevance. This approach was chosen because of its simplicity. Both the pairwise and the listwise approach require additional computations (e.g., for the pairwise approach: creating an overall preference order based on the predicted pairwise preferences) and were thus not applied.

We decided to select the event with the highest estimated relevance as output. We also investigated whether a randomized selection (with equal probabilities or probabilities based on the estimated relevance) on the top k elements of the resulting sorted list might yield better performance, but it turned out to be inferior to the much simpler "pick the best" approach.

3.5.1.1. Data Set

The data set for the first version of the ranking problem was obtained by using the filtering data set described in Section 3.4.1. As already mentioned at the end of Section 3.4.3, the usability classifications obtained during ten-fold crossvalidation were used to filter the events in the data set. For the ranking data set, only events that "survived" this crossvalidation filtering were considered. Re-using the already existing data set allowed us to quickly bootstrap the ranking mechanism. Moreover, it enabled us to do some end-to-end evaluation using a consistent data set.

We decided to use only pre-filtered events as basis for the ranking data set because also in the final system the ranking component will work on the output of the filtering component. By also using filtered training data, the distribution of the data set can be expected to be more similar to the distribution of data that the ranking subsystem will encounter "in the wild".

Each event was labeled as belonging to one of the following four ordered classes: Irrelevant (index 0), Partially Relevant (index 1), Relevant (index 2), and Very Relevant (index 3). We decided to omit the often used fifth class Perfect Match as we think the four-class differentiation is already sufficient for the given ranking problem.

Events were labeled by the author of this thesis. The labeling was done query-wise, considering only the events' relevance with respect to the query (i.e., from an objective point of view, as user interests did not play a role, yet). In addition to the 44 one-keyword queries that "survived" the filtering process, an additional set of 20 multi-keyword queries (18 queries consisting of two keywords and two queries with three keywords) was created by combining keywords from the one-keyword queries. For example, the query "volcano, erupt, Iceland" was created by combining the three queries "volcano", "erupt", and "Iceland". Thanks to the simple solution of dealing with multiple keywords in filtering (described in Section 3.4.4), the set of events to label for this three-keyword query equals the union of the events to label for each of the individual queries. Note that in such

	Number of Events	Irrelevant	Partially Relevant	Relevant	Very Relevant
One-keyword queries	926	416 (44.92%)	262 (28.29%)	193 (20.84%)	55 (5.94%)
Multiple-keyword queries	1238	909 (73.42%)	239 (19.31%)	69 (5.57%)	21 (1.70%)
Total	2164	1325 (61.23%)	501 (13.15%)	262 (12.11%)	76 (3.51%)

Table 3.7.: Table showing some properties of the bootstrap data set.

combined queries only events matching all (or most) query keywords are considered to be Relevant. In the example query of "volcano, erupt, Iceland", only events about a volcano erupting in Iceland are considered to be Relevant or Very Relevant. Events about the election of the Icelandic parliament are at most Partially Relevant – however, for the one-keyword query "Iceland" they might be considered Relevant or even Very Relevant. As one can see, when events are labeled in different contexts, they might be assigned different relevance labels.

Table 3.7 gives an overview over the resulting "bootstrap data set" and its label distribution. Appendix A.4.1 lists all queries used in this data set and their respective label distribution.

Although the number of events for multiple-keyword queries is higher than for one-keyword queries, the total amount of Relevant and Very Relevant events is considerably lower. Also the fraction of Irrelevant events is higher for multiple-keyword queries than for one-keyword queries. These observations however come as no surprise. Consider again the example of the "volcano, erupt, Iceland" query: As the set of events under consideration consists of the union of the events for the individual queries "volcano", "erupt" and "Iceland", one can easily imagine that many of the events are not Relevant to all three query keywords, but maybe only to one of them. Therefore, for the multiple-keyword query these events might be Irrelevant or (maximally) Partially Relevant, whereas for the individual queries they might be even Very Relevant.

3.5.1.2. Approach

In order to solve the ranking problem, we made use of the "learning to rank" approach that was described in Section 2.4. More specifically, we followed the point-wise approach using random regression forests. This particular approach was chosen because it was very straightforward to implement, and because the literature indicates [36, 10] that this simple approach can perform competitively compared to more complex state-of-the-art systems.

The random forests were trained not on the class indices, but on the $regressionValue$ which is defined as

$$regressionValue = 2^{classIndex} - 1$$

to put more emphasis on the RELEVANT and VERY RELEVANT classes (as is commonly done in the "learning to rank" setting, cf. Section 2.4). The quality metric being optimized during training was the mean squared error (MSE). However, as will be discussed in Section 3.5.1.3, other metrics based on the resulting ranking were used for evaluation. These ranking-related metrics were not used as optimization criterion during training as this would have required considerable changes to the underlying machine learning algorithm.

A large set of features was created consisting of a total number of 485 features. Note that most of these features are variants of each other differing only in with respect to some parameters, e.g., in the way of aggregating results across different keywords (taking the average, the minimum, or the maximum). These features can be categorized with respect to two dimensions: whether they are based on the RDF-representation or the source text, and with respect to the "Event"-"Query"-"Event-Query" categorization introduced in Section 3.4.2. Some features used in the final regressor are the following:

- **sentenceLengthCharMin**: looks at all the sentences in which the event was mentioned and counts their length in characters. Returns the minimum over all these sentences. This feature can be classified as being based on the source text and belonging to the "Event" category.
- **numberOfDummyEntities**: looks at all entities of the given event and counts how many of them are not part of the DBpedia subgraph and furthermore do not have an `rdf:type` parent. This is an RDF-based feature from the "Event" category.
- **BM25Sentence1.2**: calculates the BM25 score (using the $k = 1.2$ in the BM25 formula) of the query and the sentences of the given event with respect to *all* sentences of *all* events that are to be ranked. This is a source text based feature from the "Event-Query" category.
- **entityEmbeddings02ed**: compares the word embeddings of the query keywords to the word embeddings of the entities' DBpedia description texts. Aggregates the similarities by taking the maximum over the entities and then the average over the keywords. This is an RDF-based "Event-Query" feature.

A full list of features used in the final classifier along with a short description and a classification with respect to the two dimensions given above can be found in Appendix A.4.2.

A quick note about the features making use of word embeddings:

Word embeddings (also called "word vectors") are high-dimensional vector representations of words [34]. They are usually obtained by training a shallow neural network with one n-dimensional hidden layer on a prediction task (e.g., by using the word to predict its context). This can be done in an unsupervised way on large amounts of texts. The actual embeddings (i.e., the mapping from a word to its n-dimensional vector representation)

are then obtained by looking at the weights that connect the hidden layer to the one-hot vector representation of the respective word.

A nice property of word embeddings is the aspect of semantic clustering: Words that are semantically related tend to have similar vectors (using the cosine of their angle as a similarity measure). This is based on the tendency of similar words to occur in similar contexts. Moreover, vector addition can provide useful information. The famous example is that the following vector equation approximately holds true:

$$king - man + woman \approx queen$$

This property is used in the embeddings-based features: For instance, in the *entityEmbeddings02ed* feature mentioned above, the word vector of the keyword is compared to the sum over all word vectors of the words appearing in the entity description text (except for "stop words" like "a", "the", "because", "in", etc.[3]).

Word embeddings tend to work best when they are trained on a data set that is similar to the one on which they are used. The word embeddings used for the ranking features were acquired from the word2vec website[4]. They were trained on a part of the GoogleNews data set (which contains about 100 billion words) with a vocabulary size of three million words. The resulting vectors have 300 dimensions. More information can be found in Mikolov et al. [34].

Because of the sometimes very large number of variants for the different features, feature selection was performed in two stages: First, all features of the same type (e.g., all features based on the BM25 measure) were grouped together. Out of each of these groups of features, the top k features were selected (with k reaching from three to seven, depending on the number of feature parameters). After this, only the top features of each feature group were used for a global feature selection step. Again, different feature selection algorithms were used in parallel before applying additional wrapper-based fine-tuning. See Appendix A.4.3 for a list of all feature selection algorithms used for this regression task.

After feature selection, some hyperparameter optimization was performed. We systematically tried different values for the number of trees in the random forest, the maximum depth of these trees, and the number of randomly selected features used to construct each tree. Although Carman & Ibrahim [10] note that a smaller sample size can help to improve the overall performance, we did not optimize this hyperparameter. The reason for this is that the Weka ML framework that is being used for this thesis does not permit to vary this parameter. Therefore, major manual engineering would have been necessary which would have exceeded the scope of this thesis.

Finally, we experimented with a threshold mechanism on the selected classifier to decide when the system should use the highest-ranked event as output and when it should rather not return anything. The idea is that whenever the expected relevance of the

[3] The stop words used were partially based on a stop word list available under `http://anoncvs.postgresql.org/cvsweb.cgi/pgsql/src/backend/snowball/stopwords/`.

[4] `https://code.google.com/archive/p/word2vec/`

highest-ranked event is below this threshold, the system should not respond at all to the user. This is motivated by the reasoning that not responding might be better than responding with something completely irrelevant.

3.5.1.3. Results

As already mentioned before, the random forest regressor was trained using the mean squared error (MSE) as optimization metric. For evaluation purposes, the MSE can indicate how well the regression problem was solved. However, in our case the regression is only used as means to an end: to produce a ranking of news events. In order to judge whether the ranking problem has been solved to a sufficient degree, other metrics are needed. Another standard metric used in the evaluation of regression tasks is the correlation between the predicted values and the actual values. The correlation can give a first impression on whether events with higher ranks receive higher predictions than events with lower ranks. However, this information is still relatively coarse-grained.

Another step closer to a useful metric is the NDCG (Normalized Discounted Cumulative Gain) which was introduced in Section 2.4 as standard metric for "learning to rank" approaches. The NDCG is defined on a list of documents (i.e., the result of a query). It is a weighted sum over the ground truth labels with weights decreasing from the top to the bottom of the list. It is computed by calculating the DCG value achieved by the regressor and normalizing it by the maximum attainable DCG value for the given list of events. The DCG is calculated based on the following formula:

$$DCG@k = \sum_{r=1}^{k} G(\pi^{-1}(r)) \cdot \eta(r) \quad \text{with} \quad \eta(r) = \frac{1}{\log_2(r+1)}$$

The parameter k gives the length of the list, and r indicates the position in the list. The function $G(\pi^{-1}(r))$ returns the $regressionValue = 2^{classLabel} - 1$ for the event at position r in the resulting ranking.

The NDCG is always in the interval $[0, 1]$ and measures the quality the overall regression-based ordering in comparison to the optimal ranking (i.e., by descending relevance). It is well suited for Information Retrieval tasks where the user is presented with a list of results. In our case, however, the system will not return a list of results but only a single result, i.e., the top of the list.

We therefore defined three additional application-specific metrics. All of them are based on the top element of the ranked list. Let Q be the set of queries and $\alpha(q)$ be the ground truth label of the event with the highest regression value for query $q \in Q$ for the given regressor. Let further $|A|$ denote the cardinality of set A. Then, our metrics can be defined as follows:

- **Average value**: $AV = \frac{1}{|Q|} \sum_{q \in Q} \alpha(q)$

  This metric simply measures the average value of the top-ranked element. It ranges from zero to three and can be interpreted as the expected relevance of the average system response.

- **Precision greater than zero:** $P_{>0} = \frac{|\{q \in Q : \alpha(q) > 0\}|}{|Q|}$

  This metric measures the frequency with which the top element of the list is at least Partially Relevant. It corresponds to the question how often the selected event is an "acceptable" response to the user query.

- **Precision greater than one:** $P_{>1} = \frac{|\{q \in Q : \alpha(q) > 1\}|}{|Q|}$

  This metric measures the frequency with which the top element of the list is at least Relevant. It corresponds to the question how often the selected event is a "good" response to the user query.

Note that $P_{>0}$ and $P_{>1}$ are special cases of the *Precision@k* metric that was introduced in Section 2.4. One gets these metrics by setting $k = 1$ (only looking at the top of the list) and defining the positive class as $\{1, 2, 3\}$ for $P_{>0}$, and $\{2, 3\}$ for $P_{>1}$.

The three metrics defined above are all *absolute* in the sense that they are defined across all queries without taking into account differences between the queries. For instance, the $P_{>1}$ metric measures the percentage of Relevant and Very Relevant system responses with respect to *all* queries in the data set. However, the maximally achievable value for this metric depends on the underlying data set: If for example 20% of the queries contain only Irrelevant and Partially Relevant events, then the maximum $P_{>1}$ value achievable on this data set is 0.80.

As this effect limits the comparability between data sets, we also introduced normalized variants for each of the three metrics listed above. Let in addition to Q and $\alpha(q)$ (as defined above) $\beta(q)$ denote the highest ground truth label for any event in query q. Then, the normalized metrics can be defined as follows:

- **Average normalized value**: $ANV = \frac{1}{|Q|} \sum_{q \in Q} \frac{\alpha(q)}{\beta(q)}$

  This metric indicates how close the ground truth label of the top list element is in average to the maximum attainable for the given query. It ranges from 0 to 1.

- **Normalized Precision greater than zero:** $P^{norm}_{>0} = \frac{|\{q \in Q : \alpha(q) > 0\}|}{|\{q \in Q : \beta(q) > 0\}|}$

  This metric measures the frequency with which the top element of the list is at least Partially Relevant for all queries that have at least one event that is at least Partially Relevant. It indicates how well the system is able to avoid Irrelevant events when it has the chance to.

- **Normalized Precision greater than one:** $P^{norm}_{>1} = \frac{|\{q \in Q : \alpha(q) > 1\}|}{|\{q \in Q : \beta(q) > 1\}|}$

  This metric measures the frequency with which the top element of the list is at least Relevant for all queries that have at least one event that is at least Relevant. It indicates how well the system can select Relevant and Very Relevant events if they exist.

So in general, the absolute versions of the metrics give an idea of the overall system performance ("in what percentage of the cases is the response at least Relevant?") whereas

the normalized metrics show how well the system performs on a given data set in comparison to the best performance achievable on this data set. When comparing the system's performance on two different data sets, it is therefore advisable to use the normalized metrics.

Note that both the NDCG and the ANV metrics are undefined on queries for which all events are labeled as IRRELEVANT (as both times the denominator of the normalization equation is zero). We decided to simply ignore these queries when computing the respective metrics. Note that we also had at least two other alternatives: setting the values of the respective metrics for the "problematic queries" to zero or setting them to one.

Setting them to zero seemed to not truthfully reflect the semantics of both metrics which try to measure how well the regressor performs with respect to the maximum attainable. If all events of a query are labeled as IRRELEVANT, any ranking of them is identical to the best ranking achievable (as *all* rankings are equivalent). This would support using a value of one. Doing so would however artificially improve the average value of the respective metric on the data set.

Note that the decision of how to deal with these "problematic" queries does not affect the comparison of different regressors on the same data set: The respective metrics of all regressors will be treated identically on the same data set. The decision taken does however affect the comparison of the same regressor across different data sets. If data set A has no "problematic" queries, but data set B has a considerable amount of them, using a value of zero for NDCG and ANV on these "problematic" queries will bias the regressor performance on data set B to look worse with respect to these two metrics than on data set A. Using a value of one will achieve the opposite effect. We therefore decided to simply ignore the queries on which NDCG and ANV are not defined while aggregating their values across queries. This seems to be the most canonical way of handling this issue.

As both the NDCG and the three self-defined metrics are defined on a query level (they need to look at the resulting ordering of the query results), evaluation was performed in a leave-one-out manner on the query level. This means that we iterated over the data set, taking out one query at a time as test set and using the events of all other queries as training set. This procedure is related to x-fold cross-validation, but instead of randomly partitioning the data into x parts, the partitioning is done using always one element as test set. As in cross-validation, the results of all iterations are averaged to estimate the regressor performance.

We performed three steps of feature selection and compared the results of each of these steps to two baselines.

The first baseline is a standard baseline in regression tasks, the "average" baseline ("Avg"). It computes the average value from all data points in the training set and always predicts this average value on the test set. Note that because all events receive the same score under this baseline, the ordering of events is arbitrary. As this ordering cannot easily be predicted in advance, this is close in nature to a random ranking approach. In order to get a more accurate estimate of this "random" behavior, we computed only

the regression metrics RMSE and correlation based on the average predicted value and all other metrics based on 100 random orderings of the events for each query.

The second baseline uses linear regression on a single feature ("SF"). This feature was picked by selecting the feature with the highest NDCG value. It simply counts the number of keyword tokens that occur in the sentence from which the given event was extracted. Ordering the events based on this single feature should give a better baseline than always predicting the average value.

In the first step of feature selection, for each feature type (e.g., all features based on the BM25 measure that use different configurations), the top k features were selected by automatic feature selection ($k \in [3, 7]$, depending on the number of initial feature variants). This yielded a total number of 58 features on which a random forest with 100 trees and standard configuration with respect to all other hyperparameters was trained ("RF-58").

In the second step of feature selection, another round of automatic feature selection was performed on the set of 58 features, yielding a reduced set of 23 features. Again, a random forest with 100 trees and standard configuration of all other hyperparameters was trained on this feature set ("RF-23").

Finally, following a wrapper-based approach and optimizing the three self-defined metrics ANV, $P_{>0}^{norm}$, and $P_{>1}^{norm}$, the feature set was further reduced to 13 features. On this final set of features, hyperparameter optimization was performed. This resulted in a random forest consisting of 200 trees and standard values for all other hyperparameters ("RF-13").

The performance of a random forest regressor depends on the seed used to initialize its pseudo-random number generator. Different seeds will lead to different decisions during the construction of the individual trees (when determining the subset of training examples and the subset of features to use for the respective tree). As the iterative wrapper-based reduction of the feature set was already computationally demanding, we performed both the feature selection and the hyperparameter optimization using the same default seed of 1 to initialize the random forest. However, this causes the optimization to somewhat overfit: We end up selecting a feature set and a hyperparameter configuration that works particularly well on the given data set for the given fixed seed. Therefore, the metrics achieved at the end of the whole optimization process are likely to be overly optimistic. In order to get a more realistic expectation of the system's performance, we therefore trained ten different random forests using the selected features and the selected hyperparameter configuration, but ten different seeds. These seeds were generated using the first ten pseudo-random integers generated by the `Java util.Random` class when initialized with a seed of 1. We averaged the achieved performance over these ten random forests to get a more robust expectation ("RF-13 10-S"). Of course, it would have been desirable to already use different seeds while optimizing the classifier, but this would have increased the already relatively long runtime of the feature selection by a factor of ten.

Table 3.8 shows the results obtained in query-based leave-one-out evaluation of the different classifiers.

Reg	RMSE	Corr	NDCG	AV	ANV	$P_{>0}$	$P_{>0}^{norm}$	$P_{>1}$	$P_{>1}^{norm}$
Avg	1.5358	-0.2579	0.6361	0.8325	0.3309	0.5222	0.5222	0.2417	0.2667
SF	1.4640	0.2875	0.6931	1.1094	0.4583	0.6719	0.6719	0.3594	0.3966
RF-58	1.3703	0.4466	0.7603	1.4219	0.5755	0.8125	0.8125	0.4375	0.4828
RF-23	1.3762	0.4425	0.7526	1.4375	0.5781	0.8282	0.8282	0.4375	0.4828
RF-13	1.3713	0.4464	0.7856	1.7656	0.7005	0.9219	0.9219	0.5781	0.6379
RF-13 10-S	1.3754	0.4423	0.7708	1.6094	0.6440	0.8844	0.8844	0.5172	0.5707

Table 3.8.: Table showing the overall regressor performance for the different regressors. The column "Reg" lists the following regressors: average baseline (Avg), single feature baseline (SF), random forest with 58, 23, and 13 features, respectively (RF-58, RF-23, and RF-13), and random forest with 13 features averaged over ten seeds (RF-13 10-S).

As one can see, the "single feature" baseline is considerbaly better than the simple "average" baseline. It produces at least Partially Relevant results in about 67% of all cases ($P_{>0}$ metric) and "good" results (i.e., Relevant or Very Relevant) for about 36% of all queries ($P_{>1}$ metric).

The random forest using 58 features improves this performance considerably with respect to all metrics, avoiding Irrelevant events for 81% of the queries ($P_{>0}$ metric) and selecting Relevant or Very Relevant events for 43% of all queries ($P_{>1}$ metric).

Reducing the number of features to 23 slightly improves the AV and $P_{>0}$ metrics (as well as their normalized versions) and slightly hurts performance with respect to RMSE, correlation and NDCG.

The optimized regressor using only 13 features clearly has the best performance: Although the differences in RMSE, correlation, and NDCG are relatively small compared to the 58 feature regressor, the three "important" metrics AV, $P_{>0}$, and $P_{>1}$ (as well as their normalized versions) are greatly improved: On average, the first element in the list reaches 70% of the maximally attainable label (ANV metric), for 92% of the queries there is an event at the top of the list that is at least Partially Relevant ($P_{>0}$ metric), and in almost 58% of the cases the top ranked event is Relevant or Very Relevant ($P_{>1}$ metric).

However, as mentioned earlier, these numbers are overly optimistic: When looking at the average taken across ten different seeds, one can see only minor changes with respect to RMSE, correlation and NDCG. However, the *ANV* drops to 64% and $P_{>0}$ and $P_{>1}$ drop to 88% and 51%, respectively. These differences are notable, but expected. However, the averaged value over ten random forests with different seeds still beats all the baselines as well as the random forests trained on larger feature sets. So when generalizing to new unseen data sets, one can expect that the trained regressor will avoid Irrelevant events in 88% of the cases where it is possible to do so ($P_{>0}^{norm}$) and when given the chance, it will identify Relevant and Very Relevant events in 57% of the cases ($P_{>1}^{norm}$).

Given the large percentage of Irrelevant events in the data set (62.48%), the high value of $P_{>0}$ indicates that the proposed approach works very well at filtering out Irrelevant

events. Moreover, only 15.55% of the events in the data set belong to the classes Relevant and Very Relevant, but the trained regressor manages to find examples of them for about 52% of the queries. This indicates, that the proposed approach also works reasonably well at identifying "good" events. Although there is still some room for further improvement, we are therefore reasonably satisfied with the results achieved so far.

We also trained other off-the-shelf regressors (linear regression, multi-layer perceptrons, and RBF networks) on the given feature sets, but they were consistently outperformed by the random forest approach. This is in line with the literature, where random forests are considered to be a good low-cost alternative to regressors specifically designed for "learning to rank" problems [36, 10].

After obtaining our final regressor, we looked for potential thresholds. First, we collected the predictions made by the regressor during the leave-one-out evaluation. Again, we ran the random forest with ten different seeds (the same as above) to get a better picture of the results. For each query, we recorded the predicted relevance value and the ground truth relevance value of the highest-ranked event as well as the highest ground truth relevance value that was assigned to any event in the query. These triples were not averaged across the different seeds, but were simply collected. The reasoning was that this way we get more data points (640 instead of 64). Moreover, averaging these triples would also mean to average the ground truth values: One regressor might put an Irrelevant event on the top of the list for a given query, while another regressor might put a Partially Relevant event there – so the average would be 0.5. This would make the computation of metrics like $P_{>0}$ impossible. But it is exactly this metric that we are trying to optimize.. Therefore, we simply accumulated all the triples across the different seed configurations.

We then sorted this data by the predicted relevance value in ascending order and looked for potential thresholds. For each potential threshold, we analyzed what happens if all events below this threshold are not reported to the user. Thresholds were computed by taking the average predicted value of two neighboring triples and rounding to a precision of two decimal places. This rounding was done to smooth the resulting thresholds and to counteract a threshold overfitting (given the limited amount of triples we use).

The threshold is supposed to work as follows: For queries where the predicted relevance value of the highest-ranked event is lower than the threshold, no event is returned, i.e., the system comes back with "I'm sorry, there's nothing I can tell you about this topic."

For each potential threshold, we computed the metrics ANV, $P_{>0}$, $P_{>1}$, and $P^{norm}_{>1}$ for the case that this threshold was used as a cutoff. As the $P_{>0}$ and $P^{norm}_{>0}$ metrics are identical on the given data set, we only looked at the $P_{>0}$ metric. For all of these metrics, only the remaining data set was used for normalization. So for instance, $P_{>0}$ then represents the percentage of at least Partially Relevant system responses based on the queries where the system actually makes a response (i.e., after applying the threshold). As the idea of this thresholding is to exclude Irrelevant events, we are mainly interested in the improvements of $P_{>0}$. In addition to the metrics listed above, we also counted the percentage of Irrelevant events that were removed by the threshold (we want this number to be high) and the percentage of other events (i.e., ones that are at least Partially Rel-

Threshold	% ans	ANV	$P_{>0}$	$P_{>1}$	$P_{>1}^{norm}$	rem >0	rem 0	$P_{>0}$ all
0.00	1.0000	0.6440	0.8844	0.5172	0.5707	0.0000	0.0000	0.8859
0.74	0.9688	0.6648	0.9129	0.5339	0.5807	0.0000	0.2703	0.9156

Table 3.9.: Table showing the ranking performance for different thresholds. The column "% ans" shows how many queries the system answers and the columns "rem >0" and "rem 0" indicate how many responses with a ground truth label of >0 and 0 have been removed, respectively. All metrics are computed based on the events not filtered out by the threshold except for the last column ($P_{>0}$ all) which is based on all events.

evant) that were removed by the threshold (this number should be low). Finally, as we are also interested on how the threshold affects performance when taking a look at the *complete* data set (and not just the subset that is left after applying the threshold), we also computed for each threshold the $P_{>0}$ metric with respect to the whole data set. For the computation of the $P_{>0}$ metric, we treated queries to which the system did not respond due to the thresholding as if they had a relevance label of 0.5 – i.e., as additional category between Irrelevant and Partially Relevant. We expect the $P_{>0}$ metric to rise as Irrelevant outputs are removed by the threshold and to fall as non-irrelevant outputs are removed. We therefore seek to maximize this metric.

Table 3.9 shows the metrics of the regressor without a threshold (threshold 0.00) and when using the selected threshold of 0.74.

As one can see, a threshold of 0.74 causes the system to not return a response to 3.12% of the queries from the data set while slightly improving the other metrics. All triples falling below this threshold have an Irrelevant event ranked highest (as the "rem >0" column indicates). As can be seen in the "rem 0" column, about 27% of the Irrelevant system responses can be suppressed by using this threshold. We selected this threshold by requiring that the fraction of answered queries has to be at least 90% (i.e., we want to eliminate at most 10% of the system responses) and by looking for the maximum value of "$P_{>0}$ all".

The data from which the threshold was selected is relatively small: Although we have 640 data points, they are all generated based on the same 64 queries and only slightly differing random forests. Therefore, it is not clear whether the selected threshold is a good choice or too data set specific. We will try to validate this threshold on another data set in Section 3.5.2.2 to see how well it generalizes.

A final remark: Please note that using all events in the data set for selecting a threshold is not helpful because evaluating the effects of any threshold needs to be carried out on the highest-ranked events of the queries – i.e., on the set of 64 data points per regressor we used. It is therefore sufficient to select the threshold directly on these data points.

3.5.2. User Model

After having reached satisfactory results on the bootstrap data set generated by one person, we wanted to generalize this approach to different users. We assume that each user can be represented as a bag of keywords that indicate his or her interests. In order to both evaluate the generalization capability of the regressor obtained in the previous section and to improve its performance based on the knowledge about the user, a new data set was created.

In Section 3.5.2.1, we describe the creation of this "UM data set" which is then used in Section 3.5.2.2 to further validate the regressor obtained in the previous section. After this, we describe our approach for improving the performance of this regressor with features based on the user model (Section 3.5.2.3). Finally, in Section 3.5.2.4 we present the results obtained by this approach.

3.5.2.1. Data Set

In order to get realistic data, we collected ratings from eleven different potential users of the system (four female, seven male). Each of them was asked to describe his or her general interests with two to six keywords as well as to give a list of twelve queries they would make to the system (each consisting of at least one and at most three keywords). The questionnaire used for gathering this data can be found in Appendix A.4.1.

This information was then used to retrieve a set of news events. For each annotator, a total number 100 events were selected which were equally distributed across the queries. As the random forest ranking system that was trained on the bootstrap data set was not yet operational at the point of this data collection, the events were selected based on a simple heuristic: For each query, about half of the events were selected by using a single feature, the other half was selected randomly. Note that only events "surviving" the event filtering (Section 3.4) were considered. The feature being used for this heuristic is the same that was also used for the "single feature" baseline in Section 3.5.1. It counts the minimal number of keyword tokens from the query that appear in the sentences in which the event is mentioned.

This feature was used for the selection of half of the events as it proved to be a quite reliable baseline in early experiments on the boostrap data set. Moreover, it is easily interpretable: An event is good only if all of the query keywords appear in its underlying sentences. This feature is therefore expected to help selecting the events that match the query very well.

This simple feature is however only used for the selection of half of the events as it is clearly only a simple heuristic: It does not take into account a wide variety of other important aspects (e.g., semantic similarity, as between "beat" and "defeat" in a sports context). Basing the collected user data completely on this simple heuristic would probably introduce a very strong bias. Therefore, half of the events were selected randomly. We did not select all events randomly as this also seems to be not an optimal solution. In the bootstrap data set, about 62% of the events are labeled as IRRELEVANT. Selecting all events randomly would probably cause a large amount of the events given to the annotators for labeling to be IRRELEVANT. However, the aim of this new data set is to adjust the

system to different user interests, so only to perform some sort of fine-tuning on which events should be judged more relevant than others. Therefore, including a large number of events that are already objectively IRRELEVANT is not helpful.

Based on these considerations, we decided to follow the described mixed strategy of combining a single feature with a random selection.

After selecting for each query the events to be labeled, the sentences mentioning these events were extracted. If an event was mentioned in more than one sentence, one of them was picked randomly. These sentences were then returned to the annotators with the task to label them according to their relevance to both the query itself as well as the annotator's general interests. The task description can be found in Appendix A.4.1. For labeling, the same four classes were used as for the bootstrap data set.

The resulting "UM data set" contains a total number of 1,096 events for 101 queries, labeled by eleven different annotators with different interests. Three annotators used five keywords to describe their general interests, the remaining eight annotators gave six keywords. These numbers are surprisingly high given that annotators were asked to give between two and six keywords. This indicates that one might need several keywords to accurately describe a user's interest profile. However, given the small sample size, we cannot be too confident in this deduction. We will revisit this aspect when analyzing the results of the user study in Chapter 4. Some keywords were used by multiple annotators to describe their interests. This list is headed by "politics" (six annotators) and includes also "travel" (three annotators) as well as "football", "movies", "cars", and "Italy" (two annotators, respectively).

Eight annotators handed in a set of twelve queries as requested. One annotator created two additional queries (a total of 14) while two annotators only gave a set of five and six queries, respectively. Overall, we collected 121 queries: 22 with one keyword, 63 with two keywords, and 36 with three keywords. Twenty of these queries did not yield any events and were thus not used for labeling. Eleven one-keyword queries, seven two-keyword queries and two three-keyword queries were thus eliminated. This was largely due to very specific requests (for instance "word embeddings") that did not match any events in the KnowledgeStore. Of course, in some cases also the event filtering component might have eliminated all of the events found in the KnowledgeStore. In total, 101 out of 121 queries could be responded to by the system, a fraction of 83.47%. This number is relatively low when one considers using the NewsTeller as standalone system. However, as the NewsTeller system is going to be used as one component out of many in a social dialog system, it is not crucial that it always returns a result – if it does not, other components can take over.

So in the UM data set, there are 11 one-keyword queries, 56 two-keyword queries, and 34 three-keyword queries. The bootstrap data set, however, contained 44 one-keyword queries, 18 two-keyword queries, and two three-keyword queries. It therefore seems like users are in general interested in a combination of two or more keywords. Also this tendency will be revisited in Chapter 4. Table 3.10 shows these numbers in comparison to the bootstrap data set and Figure 3.5 illustrates the differences.

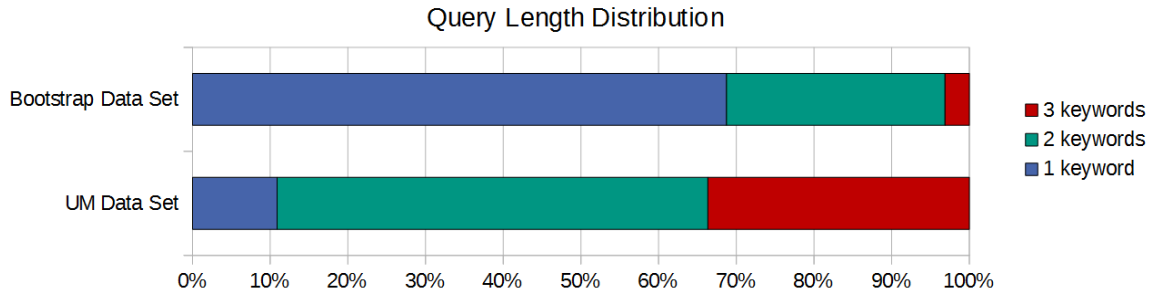

Figure 3.5.: Illustration of the query length in keywords for both the bootstrap data set and the UM data set.

Property	**Bootstrap data set**	**UM data set**
Number of events	2164	1096
IRRELEVANT	61.23%	41.70%
PARTIALLY RELEVANT	19.31%	34.12%
RELEVANT	5.57%	12.96%
VERY RELEVANT	3.51%	11.22%
average event score	*0.5652*	*0.9370*
Number of queries	64	101
one keyword	44	11
two keywords	18	56
three keywords	2	34
average number of keywords	*1.3438*	*2.2277*
top event IRRELEVANT	0 (0.00%)	10 (9.90%)
top event PARTIALLY RELEVANT	6 (9.38%)	24 (23.76%)
top event RELEVANT	19 (29.69%)	23 (22.77%)
top event VERY RELEVANT	39 (60.94%)	44 (43.56%)

Table 3.10.: Table comparing the UM data set to the bootstrap data set.

On average, each annotators labeled 9.18 queries with a minimum of five and a maximum of twelve queries. Almost all annotators labeled 100 events, two annotators only labeled 99 events and one annotator labeled 98 events. In the overall data set, 41.70% of the events were labeled as IRRELEVANT, 34.12% as PARTIALLY RELEVANT, 12.96% as RELEVANT and 11.22% as VERY RELEVANT. Table 3.10 compares these percentages to the ones of the bootstrap data set being used in Section 3.5.1 and Figure 3.6 illustrates them graphically. As one can see, they are quite different from the percentages in the bootstrap data set. The UM data set contains a larger percentage of both RELEVANT and VERY RELEVANT events. Moreover, less events are labeled as IRRELEVANT and more as PARTIALLY RELEVANT. The lower number of IRRELEVANT events indicates that using a simple feature for selecting 50% of the events worked as intended. Indeed, there is a general shift towards higher relevance ranks. However, one cannot exclude that the annotators in general might simply

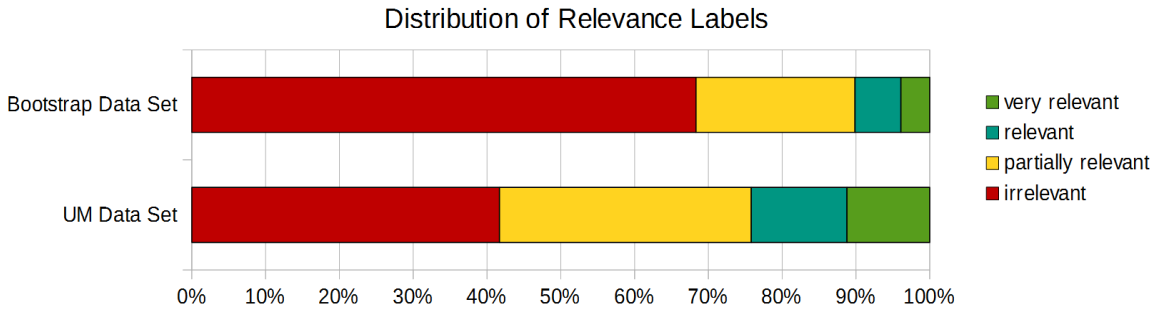

Figure 3.6.: Illustration of the distribution of relevance labels for both the bootstrap data set and the UM data set.

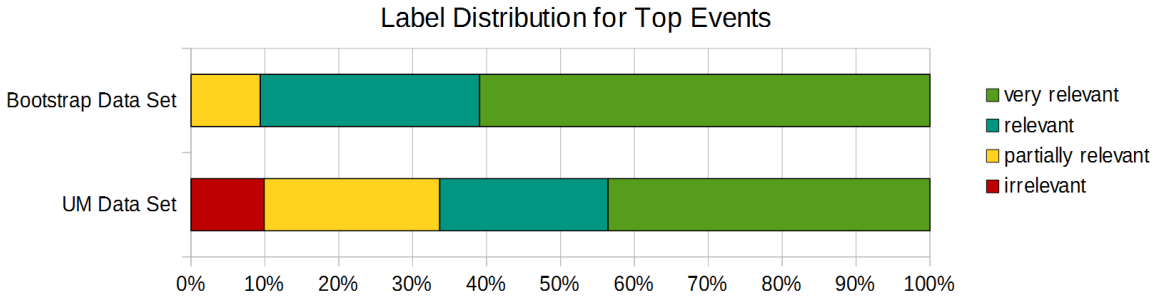

Figure 3.7.: Illustration of the distribution of the highest relevance label per query for both the bootstrap data set and the UM data set.

have been more "generous" in labeling the data than the annotator of the bootstrap data set.

When taking a closer look, one can see differences between the individual annotators. One of them, for example, rated about 7% of the events as Irrelevant and about 20% as Very Relevant, whereas another annotator labeled 67% as Irrelevant and only 4% as Very Relevant. There are at least two factors contributing to these big differences: On the one hand, the labels themselves are relatively fuzzy in their meaning and not well-defined. Therefore, different annotators might interpret them differently and thus have a different subjective perspective on what they would classify as relevant. On the other hand, the annotators labeled events for different queries. There was no fixed set of events that were rated by all of them, but every annotator labeled events for his/her queries. Therefore, if the event selection heuristic worked better on some queries than on others, the objective quality of the data provided to the different annotators would differ – depending on what type of queries they made. This would of course also influence their labeling behavior.

When considering only the event with the highest label per query, a different picture emerges: In the bootstrap data set, there are no queries with only Irrelevant events, 9.38% of queries with maximally Partially Relevant events, 29.69% of queries where the top event is Relevant, and a vast majority of 60.94% of queries that contain at least

one Very Relevant event. In the UM data set, however, about one out of every ten queries contain only Irrelevant events, 23.76% contain only Irrelevant and Partially Relevant events, 22.77% have a highest ranked event that is Relevant and only 43.56% of the queries contain a Very Relevant event. This is illustrated in Figure 3.7.

Although in general the UM data set contains a lower percentage of Irrelevant and a higher percentage of Very Relevant events than the bootstrap data set, this effect is apparently reversed when only looking at the highest label given to any event for a query. It seems that in the bootstrap data set, there is only a small overall fraction of Very Relevant events, but they are spread out across all queries in such a way that about 60% of the queries contain at least one such event. In the UM data set, this fraction of Very Relevant events is higher in general, but they seem to cluster in a smaller percentage of queries. On the other hand, there is a large fraction of queries that contain only Irrelevant events.

These differences between the two data sets are probably based on their respective "thoroughness": In the bootstrap data set, for each query *all* events are labeled. In the UM data set, however, for each query only about eleven events were labeled. With this relatively small sample size per query (especially for queries with hundreds or thousands of events), it is quite likely to produce "outlier samples", i.e., samples that significantly deviate with their label distribution from the underlying distribution for that query. For instance, it is sufficiently probable to draw a sample of only Irrelevant events. Moreover, due to the usage of the selection heuristic for 50% of the events to be rated, it might also be quite likely to draw a sample containing a large fraction of Relevant or Very Relevant events (given that the heuristic works as expected – which seems to be the case as argued above). The observed differences come therefore as no big surprise.

Finally, we also looked at the number of overlaps between the user interests and the user queries, i.e., the number of keywords that the annotators used both to describe their general interests and to formulate queries. We only looked at identical keywords, not at semantically closely related ones. On average, about 19% of the queries originally formulated by the annotators contain at least one of the keywords describing their interests. However, there seem to be different types of annotators: Three annotators used their interest-keywords in more than 60% of their queries, whereas the remaining eight annotators did not use them at all (six annotators) or only in one of their queries (two annotators). This is an interesting effect, which might be caused by the way the interests and queries were elicited: The annotators were first asked to describe their interests and immediately afterwards to formulate queries. This means, that they may have been biased towards re-using the already used keywords. This effect might have been reduced if the annotators were asked for their interests in a first session and for the queries in a second session some time later, or if the order of elicitation was reversed (first asking for concrete queries, then asking for interests). However, as only three out of eleven annotators were showing this effect, we did not analyze it any further.

Some final remarks: It seems that the instructions to "use correct capitalization" for the keywords and to separate the keywords within a query by commas were unclear to at least some annotators – they ended up capitalizing all words or not separating the keywords at

all. This had to be manually corrected before generating the events. There are two ways of dealing with this issue: either providing a better description of the required input format in the user study or writing a small preprocessing script that uses for example named entity recognition to separate different keywords. For the user study described in Chapter 4, we chose the former approach. Note that in the final usage of the system, the keyword extraction component of the social dialog system is assumed to give only well-formed input to the NewsTeller system.

Moreover, many queries were formulated in a "search engine style", i.e., as if the annotators were searching for information on the Internet using a search engine. This means that annotators were not necessarily querying for news – thus, in some cases there did not exist any matching news events. Again, a more elaborate description of the expected type of queries can mitigate this problem in the user study. However, with respect to the final use of the system (proposing news-related small talk topics based on the last user utterance), one can imagine for most of thsee "search engine style" queries that the respective keywords are mentioned in a small talk conversation. Therefore, this effect might not be that important.

Finally, some of the queries were very specific (e.g., "word embeddings" or "Jon Snow, Khaleesi, Season 6") and none of their keywords was mentioned in any news article. Although this might decrease user satisfaction when using the NewsTeller system as standalone system, in its envisioned usage context, this should not be a problem: If there is nothing to say about this topic from a news perspective, then other modules of the dialog system will be used to continue the conversation.

3.5.2.2. Re-evaluation of Previous Regressor

After having collected data from potential users, this new data set was used to investigate the generalization properties of the approach devised in Section 3.5.1. Table 3.11 shows the results.

The first part of the table shows the results obtained on the bootstrap data set (as reported in Section 3.5.1.3) and is presented here for comparison. The second part of the table shows results obtained on the UM data set, again for both baselines and the random forest. All metrics were again computed in a query-based leave-one-out fashion. The third part of the table shows the results of training the random forest on the bootstrap data set and testing it on the UM data set. Its performance in this setting can tell something about its capacity to generalize. Note that for all random forest results reported in the table, ten different seeds were used and the results were averaged across these seeds. Now let us look at the results in more detail.

First of all, let us compare the results that were obtained on each data set individually by using a query-based leave-one-out procedure (i.e., the first two parts of the table). The higher RMSE on the UM data set for all approaches comes as no surprise, given that the data is more uniformly distributed than in the bootstrap data set, where the IRRELEVANT events had a vast majority. In both cases, the "average" baseline performs worst, the "single feature" baseline performs better, and the random forest performs best.

Reg	RMSE	Corr	NDCG	AV	ANV	$P_{>0}$	$P^{norm}_{>0}$	$P_{>1}$	$P^{norm}_{>1}$
Bootstrap data set									
Avg	1.5358	-0.2579	0.6361	0.8325	0.3309	0.5222	0.5222	0.2417	0.2667
SF	1.4639	0.2875	0.6931	1.1094	0.4583	0.6719	0.6719	0.3594	0.3966
RF	1.3754	0.4423	0.7708	1.6094	0.6440	0.8844	0.8844	0.5172	0.5707
UM data set									
Avg	2.1805	-0.5608	0.6375	0.9443	0.4551	0.5882	0.6601	0.2486	0.3693
SF	2.0648	0.3119	0.6692	1.0891	0.5259	0.6436	0.7222	0.2970	0.4412
RF	2.0173	0.3806	0.7111	1.3248	0.6409	0.7109	0.7978	0.4000	0.5941
Transfer condition									
RF	2.2503	0.4037	0.7750	1.2812	0.6100	0.6931	0.7778	0.3881	0.5765

Table 3.11.: Table showing a performance comparison on both data sets. The regressors shown in column "Reg" are the average baseline (Avg), the single feature baseline (SF) and the random forest regressor (RF) from Section 3.5.1.

The ANV, $P^{norm}_{>0}$, and $P^{norm}_{>1}$ metrics are higher for both baselines on the new data set. We assume that this effect is also based in the underlying label distribution.

For example, let us consider the $P^{norm}_{>0}$ metric: In the UM data set, the proportion of Irrelevant events is considerably smaller than in the bootstrap data set. This also means that the average query will contain a smaller fraction of Irrelevant events. Thus, the probability of putting an Irrelevant event on top of the list during random shuffling is smaller. In the case of random shuffling, the $P^{norm}_{>0}$ metric expresses the expected probability of putting an event that is at least Partially Relevant on top of the list, given that the query does contain at least one such event. As one can see, this probability is higher on the UM data set than on the bootstrap data set. Therefore, the $P^{norm}_{>0}$ metric should be higher on the UM data set than on the bootstrap data set. Similar arguments apply also to the ANV and $P^{norm}_{>1}$ metrics. For the "single baseline" feature, another factor might also play a role: As the same feature that was used to select about 50% of the data set is now used to predict the relevance labels, it has at least in theory an advantage on the UM data set in comparison to the bootstrap data set.

The random forest approach performs worse on the new data set than on the old data set. It still beats both baselines, but especially the $P^{norm}_{>0}$ metric drops notably from 88% to 79%. This indicates that the features selected on the bootstrap data set do not capture the regularities in the UM data set as well as they did on the bootstrap data set. This comes as no big surprise, as the data sets are different with respect to both the presence/absence of user interests in the rating process and the label distribution. Surprisingly, the $P^{norm}_{>1}$ metric improves by two points absolute on the new data set in comparison to the old data set. This may be partially caused by the same reasons responsible for the improvement of the two baselines. However, it is not quite clear why these reasons should only apply to the $P^{norm}_{>1}$ and not to the other two metrics. As the improvement is only minor, a part of it might also be attributed to random noise.

Note that only the correlation, NDCG, ANV, $P_{>0}^{norm}$, and $P_{>1}^{norm}$ metrics can be safely compared across different data sets. All other metrics are data set specific and were therefore not considered in the above analysis.

Now let us take a look at the "transfer" condition where the random forest was trained on the bootstrap data set and was tested on the UM data set.

As one can see in the table, it has an inferior, but nevertheless comparable performance to the random forest that was trained and evaluated on the UM data set in a leave-one-out manner: It is better with respect to correlation and NDCG, but is slightly worse with respect to the other metrics. Although it has a higher RMSE than any of the baselines, it outperforms all of them with respect to all other metrics – confirming that RMSE is not a reliable performance indicator for this application. Although the two data sets are different in some important aspects like the distribution of labels, the regressor generalizes reasonably well. Although it performs definitely worse than on the bootstrap data set, it manages to produce results comparable to a leave-one-out on the UM data set without using any information about the UM data set during training. It seems that the performance loss with respect to the previous results on the bootstrap data set are mainly due to characteristics of the UM data set that are not captured well by the old features. One can of course argue that the leave-one-out results on the UM data set are not much better than the results of the "transfer" condition because the UM data set is only about half as big as the bootstrap data set. The reasoning would be that the much smaller amount of training data in combination with the same number of features leads to overfitting – which in turn would cause bad results of the random forest regressor on the UM data set in a leave-one-out evaluation. We cannot rule this out completely, but random forests are in general known to be relatively resistant against overfitting (see e.g., [36]). Moreover, the observed performance drop with respect to the bootstrap data set is very similar to the "transfer" condition which indicates that there might be similar underlying reasons.

Overall, we can conclude that the given approach generalizes reasonably well to the new data set. Although both data sets are quite different in some regards, the results achieved are satisfactory. There is however a significant performance drop, especially with respect to the $P_{>0}^{norm}$ metric. Our working hypothesis is that this might be caused by the subjectivity of the relevance rating in the new data set – which in turn can hopefully be at least partially captures by using information from the user model. This will be the subject of the following sections.

Before we move on to the UM-based approach, let us take a look at the threshold. In Section 3.5.1.3, we estimated a threshold for when to not respond to the user query. The selected threshold was 0.79. In the "transfer" condition, we extracted the (*predicted value*, *actual value*, *maximal value*) triples for the top element of each query for each of the ten random forests being used. This yielded a total number of 1010 triples. We both evaluated the effects of setting the threshold to 0.79 and identified an optimal threshold for the given triples by maximizing the $P_{>0}^{norm}$ with respect to all queries as explained in Section 3.5.1.3.

Table 3.12 shows the results. Due to space constraints, only the normalized versions of all metrics are shown. The table compares three thresholds: 0.00 (which is equivalent to not having a threshold), 0.49 (which is the optimal threshold for the given triples)

Threshold	% ans	ANV	$P^{norm}_{>0}$	$P^{norm}_{>1}$	rem >0	rem 0	$P^{norm}_{>0}$ all
0.00	1.0000	0.6100	0.7778	0.5765	0.0000	0.0000	0.7789
0.49	0.9198	0.6429	0.8215	0.8546	0.0271	0.2000	0.8267
0.79	0.5842	0.7225	0.8815	0.6863	0.3200	0.6323	0.7467

Table 3.12.: Table showing the ranking performance for different thresholds based on applying the old regressor to the new data set. The column "% ans" shows how many queries the system answers and the columns "rem >0" and "rem 0" indicate how many responses with a ground truth label of >0 and 0 have been removed, respectively. All metrics are computed based on the events not filtered out by the threshold except for the last column ($P^{norm}_{>0}$ all) which is based on all events.

and 0.79 (which is the optimal threshold on the old data set). Just by the fact that the optimal threshold is lower than the one obtained before, one can already guess that the old threshold eliminates a too large proportion of the responses. As one can see in the table, with the old threshold of 0.79, only 58.42% of the queries are answered by the system, i.e., more than 40% of the queries are ignored by the system. This number is clearly far too high. Although the three metrics ANV, $P^{norm}_{>0}$ and $P^{norm}_{>1}$ greatly improve with respect to the 0.00 baseline, the cost outweighs the benefits. This threshold admittedly manages to eliminate the majority of IRRELEVANT responses (63.23%), but it also eliminates almost a third of at least PARTIALLY RELEVANT responses (32.00%). The "$P^{norm}_{>0}$ all" metric gives another clue that this threshold might not be suitable: It is lower than for the threshold 0.00, indicating that the beneficiary effects of removing IRRELEVANT system responses are outweighed by the damage done due to removing also a large amount of non-irrelevant system responses.

The best threshold of 0.49 on the given triples is much more modest: Only about 8% of the responses are excluded and there are some slight but notable improvements with respect to all metrics computed on the remaining responses. About 20% of the IRRELEVANT system responses are removed and only less than 3% of the other responses are excluded which seems to be a reasonable tradeoff. Note that this threshold of 0.49 does not yield any noticeable improvements on the bootstrap data set – there, it is too small.

It seems like this thresholding approach does not generalize very well across the two data sets. In fact, its generalization is much worse than the one of the regressor itself. It is not quite clear what might be the reasons for this. We assume that the differences between the data sets play a major role but we were not able to determine which of the data set characteristics is mainly responsible for this poor generalization. It might be an interesting direction for further research to investigate this issue, for instance by using artificially generated data sets. This is however outside of the scope of this thesis.

3.5.2.3. Approach

After the re-evaluation of the regressor optimized on the bootstrap data set, we now turn to the question of how to use information from the user model (i.e., the user's interests) to improve the performance of the system. As the random forest approach has proved to be fruitful, we want to continue using it. Therefore, only the features and the hyperparameters change.

As now the information about the user's interests is also available for defining features, the "Event"-"Query"-"Query-Event" categorization is extended with "Interest", "Interest-Event", and "Interest-Query" categories.

The features we considered for the new regression approach can be divided into three groups:

1. Already existing features that were defined in Section 3.5.1.2 and that do not make use of user model information. All of them belong to the categories "Event", "Query", or "Query-Event".

2. Copies of already existing features from the "Query" and "Query-Event" categories that were modified to use the user's interests instead of the user query. They belong to the categories "Interest" and "Interest-Event".

3. One completely new feature that uses word embeddings to compare the keywords from the user query to the keywords representing the user interests. It belongs to the category "Interest-Query".

We investigated two approaches for selecting a new feature set: One was based on improving the feature set of the "old" regressor by adding and removing features, and one was based on doing a global feature selection. For both approaches, the union of the bootstrap data set and the UM data set was used to optimize the feature set. As we already argued, the two data sets differ in some way, so using both of them for feature selection seems to make sense as they might cover complementary aspects of real-world data. For instance, the bootstrap data set is more "thorough" in the sense that it provides labels for *all* events retrieved for a given query, whereas the UM data set provides information about user interests. We will now describe both feature selection approaches in some more detail. Please note that just as in Section 3.5.1.2, we pre-filtered the features by only keeping the most useful feature variants of each feature type before applying the actual overall feature selection described below.

The first approach starts with the feature set obtained on the bootstrap data set, i.e., a subset of feature group one. This feature set contains 13 features. We iteratively add five features from the union of feature groups two and three in a wrapper-based approach, obtaining a set of 18 features. After this, we iteratively remove five features from this feature set (again wrapper-based) to arrive at a new feature set of 13 features. This whole procedure is implemented using a beam search, i.e., always using the top six feature sets of the current iteration to produce candidate feature sets for the next iteration. This "add, then remove" approach is based on the assumption that the original feature set is already optimal or near-optimal with respect to feature group one. As shown in the previous

section, it seems to generalize not perfectly, but still in a satisfactory manner to the UM data set. Based on this initial feature set, we want to find the five best UM-based features. As a feature set with 18 features is however probably too large, we reduce it again by removing features. Note that we can either remove newly added features or features from the original set. We decided to aim for a feature set of 13 features to achieve a better comparability to the previous results. Otherwise beating the no-UM baseline might be simply a matter of having a larger feature set and not of solving the problem in a better way.

The second feature selection approach was based on a global optimization: Using different feature selection algorithms on the union of all three feature groups, a candidate feature set was obtained. We then used a wrapper-based beam search approach to iteratively eliminate features from this feature set until the resulting set only contained 15 features. Given the larger amount of training data (about 3,100 training examples in the union of both data sets compared to about 2,100 instances in the bootstrap data set), we allowed for two more features than before. In order to have a comparable baseline, we planned to apply the same procedure to the feature group one (i.e., only non-UM features) for reaching a total number of 15 features. However, as it turned out, this was not necessary as the feature set obtained using the "add, then remove" approach was superior to the one obtained using the elimination approach.

For both feature selection approaches, we used a wrapper of the random forest regressor for the individual iterations. Again, as in Section 3.5.1.2, a random forest with 100 trees and otherwise default hyperparameter values was used. However, this time the leave-one-out procedure was not performed on the query level, but on the user level. This decision was based on the reasoning that we are mainly interested in how well the system can generalize to new users for which only the abstract user model (i.e., the bag of keywords representing the interests) is given, but no concrete ratings. Moreover, the user-based leave-one-out has a lower computational demand: There are only eleven folds (one for each user) on the UM data set instead of 101 folds (one for each query).

For each fold of the leave-one-out procedure, the bootstrap data set was included in the training set. Evaluation was however only performed on the UM data set. This was done based on the following reasoning: As stated before, we wanted to utilize both data sets as they probably contain complementary information. Moreover, we decided to perform the leave-one-out procedure on the user-level as this seems to be a conceptually more sound approach. However, this leaves the problem of how to incorporate the bootstrap data set.

Viewing it as an additional 12th user does not seem very appealing as this would introduce a sharp imbalance with respect to both the number of queries per user and the number of events per user: Every user labeled about 100 events for about nine queries, but the bootstrap data set contains about 2,100 events for 64 queries.

Another option would be splitting up the bootstrap data set into several virtual users. However, this introduces the question of how to split the data. One could split the data into six users (each one with about ten queries and about 350 events) or into 21 users (each one with about three queries and about 100 events) or any number between these two extremes. However, the number of events per query in the bootstrap data set is not

evenly distributed, therefore partitioning the data set into virtual users might be difficult. As it can be seen, this splitting approach would introduce some complications.

Although it would be interesting to see how well the system can deal with the cold-start problem (i.e., having no information about a new user – in our case an empty user model), this is outside of the scope of this thesis: We assume that a user model is given. As both approaches of incorporating the bootstrap data set into the leave-one-out procedure as virtual user(s) do not seem very appealing, we simply always use it as additional training data.

As the baseline random forest was only optimized on the bootstrap data set, it is somewhat questionable to compare it to the random forests that were optimized on the overall data set. We therefore also applied the "add, then remove" strategy to the old feature set, using the exact same procedure as described above, but only features from the feature group one (i.e., not taking into account the user model at all). This was done to obtain a better baseline. This baseline was in turn used to again apply the "add, then remove" strategy with the user model features (feature groups two and three) to see if we can further improve the regressor's performance.

For the final evaluation of the selected features and hyperparameters, again the performance of ten random forests initialized with different seeds was averaged to get a realistic expectation that is less affected by overfitting effects.

3.5.2.4. Results

As already stated in the previous section, the leave-one-out procedure was not performed on the query level, but on the user level. All of the metrics defined in Section 3.5.1.3 for the no-UM case are however still applicable. As they are averaged across all queries (and thus also across all users), they can however only give information about the average system behavior. This is very useful information, but we are not only interested in a system that performs well in average, but we want that system to also have acceptable performance in the worst case. We therefore also computed the minimum value of ANV, $P^{norm}_{>0}$, and $P^{norm}_{>1}$ across all users (i.e., we computed these metrics on a user-level and took the minimum of each of them across all users). These additional metrics help to make sure that the system's performance is acceptable for all users (in addition to being good on average).

For the optimization procedures described in the previous section (both feature selection and hyperparameter optimization), the following six metrics were used: ANV, $P^{norm}_{>0}$, $P^{norm}_{>1}$, "min ANV", "min $P^{norm}_{>0}$", and "min $P^{norm}_{>1}$". Due to spatial constraints, only the normalized metrics are shown in this section. Their respective absolute versions are given in Appendix A.4.4.

Table 3.13 shows the average results for the different approaches, and Table 3.14 shows the minimum performance across all users for the given approaches. All metrics were computed by doing a user-level leave-one-out evaluation on the UM data set using the bootstrap data set as additional training data. Both tables are divided into three parts: The uppermost part shows the two simple baselines already known from earlier sections. The middle part of the table shows two regression approaches that do not make use of

Regressor	RMSE	Corr	NDCG	ANV	$P^{norm}_{>0}$	$P^{norm}_{>1}$
Average baseline	2.2277	-0.2015	0.7137	0.4730	0.6778	0.3706
Single feature baseline	2.2094	0.3114	0.7436	0.5259	0.7222	0.4416
no-UM baseline	2.1025	0.3699	0.7804	0.6185	0.7899	0.5397
no-UM addRemove	2.0656	0.3715	0.8132	0.6944	0.8600	0.6294
UM addRemove	2.0609	0.3326	0.8078	0.7074	0.8511	0.6868
UM fromScratch	2.0586	0.3409	0.8147	0.7020	0.8567	0.6544
UM addRemove v2	2.0296	0.3715	0.8333	0.7400	0.8667	0.7191

Table 3.13.: Table showing the average performance of different regressors on the UM data set in a user-based leave-one-out evaluation.

Regressor	min ANV	min $P^{norm}_{>0}$	min $P^{norm}_{>1}$
Average baseline	0.2263	0.3982	0.0654
Single feature baseline	0.3125	0.5000	0.0000
no-UM baseline	0.4342	0.5665	0.1943
no-UM addRemove	0.5134	0.6345	0.4114
UM addRemove	0.5342	0.6807	0.5095
UM fromScratch	0.5569	0.6462	0.4740
UM addRemove v2	0.6036	0.7125	0.5481

Table 3.14.: Table showing the minimum performance across users of different regressors on the UM data set in a user-based leave-one-out evaluation.

user model information: the regressor obtained in Section 3.5.1 ("no-UM baseline") and an improved version obtained by executing the "add, then remove" feature selection strategy described in the previous section ("no-UM addRemove"). The third part of the table finally shows the results obtained by also taking into account information from the user model. It contains three variants, which were obtained by using the "add, then remove" strategy on the feature set of the "no-UM baseline" ("UM addRemove"), using a global feature selection procedure by iterative elimination ("UM fromScratch"), and by using the "add, then remove" strategy on the "no-UM addRemove" regressor ("UM addRemove v2").

The results again confirm that the "no-UM baseline" outperforms both of the simple baselines with respect to all metrics. Note however that all three of them perform very poorly with respect to the "min $P^{norm}_{>1}$" metric. This means that for the two simple baselines, there is at least one user for which the respective baseline *never* manages to present an event that is at least RELEVANT when given the chance. Also for the "no-UM baseline", this only works for about one fifth of the queries in the worst case. The poor performance of the "no-UM baseline" regressor seems to be based on a suboptimal feature set – after allowing to feature set to be adjusted to the new data set via the "add, then remove" approach, performance improves consistently across all metrics. This indicates that the feature selection performed in Section 3.5.1 might have been overfitting on the bootstrap data set by selecting features that work especially well on this particular data set but that

do not generalize well to other data sets. By re-selecting a part of the feature set, we of course face the risk that the new feature set now overfits the new data set. We will investigate this issue further down in the text. For now let us note that the average performance of the regressor can be satisfactory (with $P_{>0}^{norm}$ of 86.00% and $P_{>1}^{norm}$ of 62.94%) even without using information from the user model. Its performance in the worst case is however not quite satisfactory: For some user, only 63.45% of the system responses are not IRRELEVANT, and for some (potentially other) user the system responses are at least RELEVANT in only 41.14% of the cases.

Let us now look at the third part of the tables which is concerned with the approaches taking into account the users' interests. The "UM addRemove" approach clearly outperforms the the "no-UM baseline" based on which it was created on almost all metrics (except for correlation). It has a performance comparable with the "no-UM addRemove" regressor – being notably worse on correlation, slightly worse with respect to NDCG and $P_{>0}^{norm}$, but slightly better on ANV and "min ANV", and considerably better with respect to $P_{>1}^{norm}$, "min $P_{>0}^{norm}$", and "min $P_{>1}^{norm}$". This indicates that the taking into account information from the user model helps to identify RELEVANT and VERY RELEVANT events (due to the considerably better values of $P_{>1}^{norm}$ and "min $P_{>1}^{norm}$)". Moreover, using user model based features seems to help improving worst case performance overall. These results are therefore definitely encouraging.

The second approach, "UM fromScratch" yields comparable performance. It is outperformed by the "UM addRemove" approach on the metrics $P_{>1}^{norm}$, "min $P_{>0}^{norm}$", and "min $P_{>1}^{norm}$", while yielding better performance with respect to the NDCG and "min ANV" metrics. One should keep in mind that the "UM fromScratch" regressor uses a total number of 15 features whereas the "UM addRemove" only uses 13 features. It is a bit surprising that using a larger feature set that has been globally optimized by iterative elimination does not outperform the simpler "add, then remove" strategy. This might be caused by two different reasons: On the one hand, using the feature set of the "no-UM baseline" as a starting point provides the "add, then remove" algorithm with a good initialization – whereas the "UM fromScratch" must (as the name indicates) completely start from scratch. The set of candidate features used in this elimination-only approach is based on the output of different feature selection algorithms and might provide a worse initialization than the feature set which was optimized on the bootstrap data set. On the other hand, one could argue that the difference is caused by overfitting due to the higher number of features. We investigated this hypothesis by looking at the performance of the "add, then remove" procedure when selecting a final number of 15 features as well as the performance of the "elimination only" procedure when selecting a final number of 13 features. As it turns out, the "add, then remove" approach performs slightly better with 13 features compared to 15 features, whereas the "elimination only" approach performs slightly better when using 15 features compared to the 13 feature case. Therefore, the number of features can play only a minor role in this performance difference. We conclude that the better initialization probably makes the difference.

Based on this conclusion we applied the "add, then remove" procedure once more – this time to the "no-UM addRemove" feature set, adding user model based features. As this feature set is already yielding considerably better performance the the "no-UM base-

Regressor	ANV	$P^{norm}_{>0}$	$P^{norm}_{>1}$
Average baseline	0.6835	0.6934	0.0000
Single feature baseline	0.5942	0.6923	0.0000
no-UM baseline	0.7020	0.7181	0.3540
no-UM addRemove	0.7393	0.7378	0.6537
UM addRemove	0.7552	0.7998	0.7419
UM fromScratch	0.7932	0.7543	0.7243
UM addRemove v2	0.8157	0.8221	0.7622

Table 3.15.: Table showing the performance difference of the regressors between worst case and average case.

line", we hoped that it would provide an even better initialization. The performance of the resulting regressor is shown as "UM addRemove v2". As one can see, our suspicion was confirmed: The "UM addRemove v2" regressor outperforms both the "no-UM addRemove" and the "UM addRemove" regressors with respect to all metrics. This undermines two of our points: Firstly, that a good initialization is important for the success of the "add, then remove" procedure (as using a better feature set as starting point yields a better result in the end), and secondly, that one can still gain from user model based features even with respect to an optimized UM-less regressor.

In general, looking at the best result we were able to achieve (the "UM addRemove v2" regressor), we can make the following observations: On average, the system manages to avoid Irrelevant events for about 86.67% of the queries, given that there is at least one event that is not Irrelevant ($P^{norm}_{>0}$ metric). If for a query there exists at least one Relevant or Very Relevant event, the regressor is able to identify it on average in 71.91% of the cases ($P^{norm}_{>1}$ metric). Moreover, the system achieves on average about 74.00% of the attainable maximum with respect to event relevance (ANV metric). These numbers are definitely not perfect, but they are promising nevertheless and sufficient for this first version of the system.

When looking at the "worst case" across users, these numbers expectably drop. However, the system still avoids Irrelevant events for 71.25% of the queries where this is possible ("min $P^{norm}_{>0}$" metric), identifies Relevant and Very Relevant events for 54.81% of the queries ("min $P^{norm}_{>1}$" metric), and still reaches 60.36% of the attainable relevance maximum ("min ANV" metric). Although these numbers are worse, they are still acceptable for a worst-case performance: Even in the worst interest-query combinations observed in the data set, more than two thirds of the system responses are at least Partially Relevant and more than half of them are Relevant or Very Relevant (always normalized to the number of queries where this is possible).

To further illustrate the performance degradation in the worst case when compared to the average case, we computed the fraction of the average performance that was achieved

Regressor	RMSE	Corr	NDCG	ANV	$P_{>0}^{norm}$	$P_{>1}^{norm}$
Average baseline	2.2249	-0.5559	0.6375	0.4551	0.6601	0.3693
Single feature baseline	2.2024	0.3183	0.6692	0.5259	0.7222	0.4412
no-UM baseline	2.0969	0.3683	0.6977	0.5919	0.7544	0.5206
no-UM addRemove	2.0202	0.4143	0.7276	0.6706	0.8467	0.5838
UM addRemove	2.0142	0.3821	0.7185	0.6591	0.8311	0.6176
UM fromScratch	1.9950	0.4233	0.7288	0.6748	0.8289	0.6191
UM addRemove v2	1.9516	0.4471	0.7362	0.7067	0.8544	0.6985

Table 3.16.: Table showing the performance of different regressors on the UM data set in a query-based leave-one-out evaluation.

in the worst case for the three metrics ANV, $P_{>0}^{norm}$, and $P_{>1}^{norm}$. Table 3.15 shows these comparisons. All of the entries are in the interval [0,1], where higher values indicate better performance in the worst case, i.e., a smaller "gap" between average case and worst case. This translates to less performance variance across users.

As one can see, the difference is especially large for the three baselines with respect to the $P_{>1}^{norm}$ metric (as was to be expected, as they also performed very poorly in absolute). In general, there seems to be a (weak) tendency for the percentages with respect to ANV and $P_{>0}^{norm}$ to behave similarly. For all approaches using user model based features, also the percentage with respect to $P_{>1}^{norm}$ is in the same order of magnitude. This again indicates that user model based features are especially helpful for identifying Relevant and Very Relevant features as already observed before.

For the overall best regressor "UM addRemove v2", all the fractions are above 75% which seems to be an acceptable difference.

In addition to the evaluation on the user-level (which simulates performance on previously unseen users), we also evaluated the given approaches using a leave-one-out procedure on the query level to see if any interesting effects emerge. As there are 101 queries, but only eleven users, this took considerably more time (each regressor had to be trained 101 times instead of only eleven times). Table 3.16 shows the results obtained. Two striking differences can be found when comparing these numbers to the ones from Table 3.13: RMSE and correlation improve, but all other metrics deteriorate. This is an interesting effect: On the one hand, the additional training data helps to solve the regression problem in a better way (as indicated by improving RMSE and correlation), but on the other hand this seems to hurt the ranking performance. Not only the three self-defined metrics concerned with the highest-ranked event decrease but also the NDCG does so. This indicates that the overall ranking performance is impaired, not only the top of the list. The performance order of the different approaches stays more or less unchanged with respect to the user-level evaluation except for the "UM addRemove" and "UM fromScratch" regressors – this time the "UM fromScratch" approach slightly outperforms the "UM addRemove" approach.

Regressor	RMSE	Corr	NDCG	ANV	$P^{norm}_{>0}$	$P^{norm}_{>1}$
Bootstrap data set						
no-UM baseline	1.3754	0.4423	0.7708	0.6440	0.8844	0.5707
no-UM addRemove	1.3738	0.4431	0.7622	0.5909	0.8250	0.5121
Transfer condition						
no-UM baseline	2.2503	0.4037	0.7750	0.6100	0.7778	0.5765
no-UM addRemove	2.2409	0.4152	0.7889	0.6407	0.8056	0.6015

Table 3.17.: Table comparing the two no-UM regressors both on the bootstrap data set using query-based leave-one-out and in the transfer condition.

So the relative performance of the different approaches when compared to each other stays largely unchanged. All of them improve with respect to the regression problem but get worse with respect to the ranking problem. We hypothesize that the improvements with respect to the regression problem are due to an increased amount of training data. The performance drop with respect to the ranking problem might be due to a natural tradeoff between performance in the two problems. In the user-based evaluation, the regressors were optimized to yield good performance with respect to the ranking problem, i.e., feature selection and hyperparameter optimization were carried out on the respective metrics. The regressors are therefore optimized to yield good performance in the ranking problem in user-based leave-one-out evaluation. Applying these regressors in a query-based leave-one-out evaluation simply means applying them in a scenario for which they were not optimized. This different scenario might have a different relationship between regression performance and ranking performance. Thus, what was optimal with respect to the prior scenario might not be optimal in this new scenario.

Because we were able to improve the "no-UM baseline" on the UM data set by modifying the feature set, we wanted to check how these changes affect performance on the bootstrap data set. We hypothesized that the old regressor was not performing well on the UM data set because its feature set and hyperparameters were overfitting the bootstrap data set. But maybe the new regressor now overfits the UM data set. We therefore evaluated the modified regressor on the bootstrap data set using a leave-one-out procedure on the query level. Moreover, we applied it in the "transfer" condition. Table 3.17 compares the results obtained to those reported in Section 3.5.1.3.

On the bootstrap data set, both regressors perform comparable with respect to RMSE and correlation, i.e., they both solve the regression problem with a similar quality. The "no-UM addRemove" regressor is however falling behind with respect to all other metrics, i.e., it is less effective in the ranking problem. While the performance differences are small with respect to correlation and NDCG, they are considerable for the remaining metrics.

When looking at the transfer condition, the "no-UM addRemove" approach performs better than the "no-UM baseline" approach. Compared to its performance on the bootstrap data set, the $P^{norm}_{>0}$ metric decreases only slightly, whereas ANV and $P^{norm}_{>1}$ even improve.

So it looks like the new feature set performs competitively on the bootstrap data set with respect to the regression problem. It however is definitely inferior with respect to the ranking problem. In the transfer condition, where the regressor is trained on the bootstrap data set and applied to the UM data set, the new feature set however outperforms the old approach. We therefore conclude that the old feature set was probably overfitting the bootstrap data set (due to its sharp performance drop) whereas the new feature set might be slightly overfitting the new data set.

We also evaluated the generalization of the threshold for the "no-UM addRemove" regressor. On the boostrap data set, a threshold of 0.92 was estimated. In the transfer condition, a threshold of 0.44 would have been optimal. Applying the threshold of 0.92 would have caused the system to respond only to 45.54% of the queries – which is even less than observed for the "no-UM baseline" in Section 3.5.1.3. This seems to indicate that the generalization ability of the threshold is not related to the generalization ability of the overall performance of the system: The "no-UM addRemove" regressor performs similar on the bootstrap data set and in the transfer condition, but its threshold does not generalize at all. We therefore think that one should be very cautious with picking the threshold.

Finally, we looked at potential thresholds for the "no-UM addRemove" and "UM addRemove v2" approaches as the respective best-performing approaches of their categories. Thresholds were estimated based on the user-level leave-one-out predictions from the experiments described above, again using ten runs with different seeds. For the UM data set, the threshold mechanism is more important than for the bootstrap data set as there are now several queries where the annotators have labeled all events as IRRELEVANT. Not responding to these queries at all would therefore be a wise strategy. Based on our experience with the poor generalization of thresholds across data sets, we are leaning towards picking smaller thresholds – it seems to be a better choice to miss out on optimizing the metrics than to respond only to half of the requests. This holds true at least with respect to the user study we conducted later on. As component of the social dialog system, the bias is the other way around: It is better to only present news events if one is completely sure that their quality is high – in any other case the other components of the dialog system can take over.

Table 3.18 shows two potential thresholds for the "no-UM addRemove" approach. The threshold of 0.86 was found by maximizing the "$P^{norm}_{>0}$ all" metric, the threshold of 1.10 by maximizing the $P^{norm}_{>0}$ metric. For both thresholds, the constraint was that the proportion of answered queries must be at least 90%. As one can see, there seems to be a rough tradeoff between "rem 0" and "rem >0": It seems like there is no clear separation between IRRELEVANT events and other events as one cannot remove many IRRELEVANT events without also removing a considerable portion of "good" events. Manual inspection showed that the value of "removed >0" is only zero for the threshold 0.00, i.e., the event with the lowest score in the given triples is not an IRRELEVANT one. It seems that when comparing the two potential thresholds, 0.86 might be the better choice. It only yields small improvements but has a better ratio of "removed 0" to "removed >0". Moreover,

Threshold	% ans	ANV	$P^{norm}_{>0}$	$P^{norm}_{>1}$	rem >0	rem 0	$P^{norm}_{>0}$ all
0.00	1.0000	0.5997	0.8367	0.6206	0.0000	0.0000	0.8367
0.86	0.9782	0.6064	0.8386	0.6270	0.0133	0.0467	0.8389
1.10	0.9079	0.6112	0.8494	0.6603	0.0784	0.1323	0.8089

Table 3.18.: Table showing the "no-UM addRemove" ranking performance for different thresholds. The column "% ans" shows how many queries the system answers and the columns "rem >0" and "rem 0" indicate how many responses with a ground truth label of >0 and 0 have been removed, respectively. All metrics are computed based on the events not filtered out by the threshold except for the last column ($P^{norm}_{>0}$ all) which is based on all events.

Threshold	% ans	ANV	$P^{norm}_{>0}$	$P^{norm}_{>1}$	rem >0	rem 0	$P^{norm}_{>0}$ all
0.00	1.0000	0.6475	0.8611	0.7147	0.0000	0.0000	0.8622
0.97	0.9673	0.6694	0.8827	0.7147	0.0000	0.1404	0.8989
1.12	0.9297	0.6880	0.9011	0.7254	0.0129	0.2596	0.9178

Table 3.19.: Table showing the "UM addRemove v2" ranking performance for different thresholds. The column "% ans" shows how many queries the system answers and the columns "rem >0" and "rem 0" indicate how many responses with a ground truth label of >0 and 0 have been removed, respectively. All metrics are computed based on the events not filtered out by the threshold except for the last column ($P^{norm}_{>0}$ all) which is based on all events.

motivated by the results of threshold generalization in the transfer condition, it might be better to be conservative and pick a lower threshold.

Table 3.19 shows two thresholds obtained for the "UM addRemove v2" approach. The threshold of 0.97 was obtained by looking for the highest threshold for which only IRRELEVANT events are removed, the threshold of 1.12 was picked based on the maximization of the "$P_{>0}$ all" metric. Again, only thresholds suppressing at most 10% of the responses were taken into consideration. As one can see, the tradeoff between "rem >0" and "rem 0" is much better in this case: One can remove 14% of the IRRELEVANT events without removing a single other event. This percentage can be increased to almost 26% which "costs" only the removal of 1.29% of other events. This indicates that the "UM addRemove v2" system manages much better to separate out IRRELEVANT events than the "no-UM addRemove" system. This is a bit surprising, as both approaches have a similar performance with respect to the $P^{norm}_{>0}$ metric. Based on the observed difference, the thresholding can help to further increase the performance gap between the two approaches. With respect to the question which threshold should be selected for the "UM addRemove v2" approach, things are not quite clear: The threshold of 1.12 seems tempting as it causes a great performance boost with only minor costs. However, the 0.97 threshold might be the "safer" alternative with respect to generalization. Just as argued above, we tend here towards

Regressor	**% ans**	**AV**	$P_{>0}$	$P_{>1}$	$P_{>0}$ **all**	$P_{>1}$ **all**
no-UM addRemove	0.9782	1.4342	0.7520	0.4271	0.7475	0.4178
UM addRemove v2	0.9673	1.5763	0.7932	0.4974	0.8010	0.4812

Table 3.20.: Table showing the absolute metrics for the regressors when using the two selected thresholds.

a conservative threshold for the user study evaluation. Therefore, we selected the 0.97 threshold for the final system.

Note that in both tables the threshold of 0.00 corresponds to using no threshold. The numbers shown in Tables 3.18 and 3.19 for this "no threshold" condition slightly differ from the numbers reported in Table 3.13 as in the latter case the metrics were aggregated on the user-level and averaged across different seeds whereas for the threshold estimation we computed the metrics based on all queries for all seeds.

Table 3.20 shows the absolute metrics for both regressors when being used with their respective selected threshold. Both systems respond to more than 95% of the queries. Note that this response rate is based on queries for which some events survive the filtering process – the overall system might respond less frequently as there might be also queries that yield no search results.

The AV, $P_{>0}$, and $P_{>1}$ metrics were computed for the case that the system responds to the user query. If it does so, one can expect the average system response to be about halfway between Partially Relevant and Relevant as indicated by the AV metric. For 75% and 79% percent of the queries, respectively, the regressors find an event that is at least Partially Relevant, and in 42% and 49% of the cases, respectively, the regressors output a Relevant or Very Relevant event.

The last two columns of Table 3.20 show the $P_{>0}$ and $P_{>1}$ values computed on *all* queries (i.e., also the ones to which the system does not respond). For the computation of these metrics, not responding to a query was judged to be better than responding with an Irrelevant event, but worse than responding with a Partially Relevant event. It was treated as an imaginary relevance class No Repsonse with a class index of 0.5. As one can see, both precision metrics yield only slightly different values when computed on all events. In general, the values are a bit lower (except $P_{>0}$ for the UM regressor) but the differences are relatively small.

So in general, when looking at the overall ranking problem, we can conclude the following: Performance is satisfactory with up to 80% of $P_{>0}$, up to 50% of $P_{>1}$, and up to an average regression value of 1.57. So for up to 80% of all queries, the system can find an event that is at least Partially Relevant, and for almost 50% of the queries the retrieved event will be Relevant or Very Relevant. However, one can see that there is still clear room for improvement. This improvement could be obtained by different means: One could define additional features, use larger feature sets for the regressors, train them on larger training sets, or change their optimization metric from a regression-based function like MSE to a ranking-based one. Also a more complex way of estimating thresholds might result in increased confidence in the thresholds' generalization ability and maybe

in improved performance. For the scope of this thesis, however, we are satisfied with the results achieved so far.

3.6. Summary Creation

After the most relevant news event has been selected, the NewsTeller system needs to create a short summary sentence about it. This is done by sentence extraction, i.e., the NewsTeller system simply extracts the sentence in which the event was mentioned. We originally planned to compare the performance of this simple approach with a more sophisticated approach using natural language generation (NLG) based on the RDF representation of the selected event. However, due to time constraints we were unfortunately not able to explore this path.

Each event in the KnowledgeStore is linked to one or more mentions that denote in which news article and at which exact characters within this article the given event was mentioned. The sentence extraction approach simply picks one of the mentions, accesses the corresponding original news article, and extracts the sentence in which the event was mentioned. Note that if an event has multiple mentions, one of them is selected randomly.

The actual extraction of the sentence based on the selected mention is done by starting at the character range denoted in the mention and extending this range both to the left and to the right until a sentence delimiter (e.g., period, exclamation mark, question mark) is found. As this simple strategy can fail with respect to abbreviations like "U.S." (where the periods do not indicate the end of a sentence, but simply an abbreviation), the sentence splitter of the Stanford CoreNLP toolkit [32] was used in a second step to improve the quality of the results: After selecting a candidate sentence with the simple approach, the CoreNLP sentence splitter is used on the original news article to obtain a list of sentences. For each of these sentences, we check whether it contains the extracted candidate sentence. If this is the case, the candidate sentence is replaced by the corresponding sentence retrieved from CoreNLP. The selected sentence from the original article is then returned as a summary.

Although this sentence extraction approach might be a useful baseline, it is clearly not optimal: If an event has multiple mentions, this means that the event has been composed of information from different sources. A sentence extracted from only one of these sources might not contain all information about the event, thus not being complete. Moreover, it might happen that a sentence in a news article mentions more than one event. Then the extracted sentence might also contain irrelevant information. Furthermore, the random selection of the mention to be used can yield suboptimal results as there is no guarantee that the “best” mention is selected.

Finally, cross-sentence anaphora in the original news article can cause some problems. Let us assume that the original article contains the following fictional paragraph:

> Angela Merkel visited Washington today. She talked to president Obama for two hours about the financial crisis.

If the "talk" event is selected as output, the system will return only the second sentence. Unfortunately, without the context of the first sentence, it is not clear who the pronoun "she" is referring to. Therefore, the user might think that this system response is inappropriate, as crucial information (*who* talked to Obama?) is missing.

Unfortunately, all these weaknesses of the current summary creation approach could not be worked on due to temporal constraints. It seems however that there are some low-hanging fruit for future development: For example, instead of selecting a mention randomly, one could use a simple heuristic. Moreover, in order to deal with cross-sentence anaphora, one could make use of coreference resolution tools. Finally, it might be also worthwhile to implement a simple NLG-based approach and to compare it to the current extraction-based approach.

3.7. Processing Time

In order to convincingly fulfill its role as social actor, a system must respond to the user's input within a reasonable amount of time – otherwise it will be perceived as not very intelligent. Also the NewsTeller system needs to fulfill certain real-time constraints, as it will be used by a social dialog system. This proved to be a major technical challenge for two reasons:

Firstly, the KnowledgeStore is accessed very frequently, e.g., for defining features in both the filtering and the ranking step. As each query to the KnowledgeStore is an I/O operation and as it takes some time for the KnowledgeStore to compute the result for each query, the time spent waiting for responses from the KnowledgeStore makes up a considerable fraction of the overall processing time.

Secondly, the calculation of some of the features used for filtering and ranking is computationally expensive. This further increases the total processing time for a query.

In order to deal with these processing time issues, we used three techniques: bulk queries, parallelization, and limiting the amount of events being processed. We will now briefly describe where, how, and why these three techniques were used.

The **bulk queries** approach targets the time spent waiting during interaction with the KnowledgeStore.

Originally, the NewsTeller system fired a separate query for each piece of information that was needed anywhere in the code. So during feature computation, for each event and for each feature, a separate query to the KnowledgeStore was fired. Although the individual execution time of these queries was very fast (less than 20 ms in most cases), their execution time adds up: For instance, if there are 1,000 events and 10 features that need to be computed for each event, and if each of these features makes a query taking 20 ms, then this results in 10,000 small queries to the KnowledgeStore. They have a total execution time of 20 seconds if executed sequentially. Parallelizing the queries helps to reduce this total amount of time but there is still a big communication overhead associated with opening and closing the respective connections.

The usage of bulk queries helped to considerably reduce this communication overhead. The idea adopted here is the following one: Instead of firing many small queries while computing the features, a small amount of large queries is executed *before* computing the features. The results of these "bulk queries" are stored in a cache. Then, during feature computation only a simple lookup is necessary. Each bulk query contains aspects used by different features (e.g., the number of `propbank:A1` links and a list of all actors participating in the event). Moreover, each bulk query retrieves these aspects not only for one event but for a large amount of events at once. This way, the total number of queries made to the KnowledgeStore could be greatly reduced and thereby also the communication overhead. This helped for instance to reduce the runtime of the filtering step for the query "Merkel" from almost two minutes to about 20 seconds.

This "bulk query" approach is also used in the initial search step. Originally, three different queries were used to find events based on their labels, actors, and places, respectively. These queries were executed once for each keyword in the user query. Again, using one large query that unifies the three individual queries and is able to deal with multiple keywords at once helped to reduce the overall time required for the search step. The transition from three different queries to one global query improved the time required for the search step from about eight seconds per user query to about three seconds per user query (computed over all user queries in the filtering data set). Using one big query, but dealing with the different keywords sequentially took 1,258 seconds (almost 21 minutes) for all user queries in the two ranking data sets. By also using only a single bulk query for all keywords, we were able to reduce this time to 893 seconds (almost 15 minutes).

The general idea of pre-computing and buffering intermediate results was also used in other places, e.g., for extracting the sentence of a given mention from the original news article.

Parallelization was mainly used to speed up the computation of features.

Especially some features used for the filtering classifier are computationally expensive. For example, some of them compute the bag-of-words overlap between the original news article and the description texts of DBpedia entities. The time required to compute these features could be greatly reduced by parallelizing the necessary comparisons (e.g., by using one thread per word in the description text).

Moreover, the feature computation was parallelized globally: After having executed all bulk queries, a new thread is started for each event to compute its feature values, using one child-thread per feature.

Also the remaining few bulk-queries to the KnowledgeStore were executed in parallel using independent sessions with the KnowledgeStore instance. This parallelization helped to further reduce the time required for executing the queries, but only to a certain extent: The KnowledgeStore processes some queries in a sequential manner even if they are executed in parallel.

Finally, **limiting the amount of events being processed** turned out to be a useful way to limit the overall processing time.

As already noted in Section 3.3, some queries yield tens of thousands of events. As the processing of each event takes a certain minimum amount of time, we can expect

the overall runtime to grow at least linearly with the number of events. In practice, this growth might be even considerably superlinear. Therefore, in order to limit the processing time, the number of events being processed needs to be limited. We found that limiting the number of events to 1,000 might be a reasonable choice. After filtering these 1,000 events (which is the pipeline stage taking up most of the processing time), one can expect to still have about 160 events left to rank. The hope is that among these 160 events, the ranking can identify at least one RELEVANT event. Of course, selecting these 1,000 events randomly introduces the risk of keeping only IRRELEVANT events for further processing while discarding potentially existing RELEVANT events. However, this seems to be a risk that needs to be taken in order to avoid astronomical response times for queries containing very general keywords. Moreover, as already argued in Section 3.3, we estimate this risk to be rather small.

Although we improved the code using the three techniques described above, the overall processing time per query can still be very high – e.g., about 75 seconds for the query "Barack Obama, election, United States". This reaction time is of course still way too high in the context of a social dialog system: Any user would expect the system to respond within a few seconds. Unfortunately, it seems like we already picked most of the low-hanging fruit with respect to runtime improvements. In order to make the system respond within a reasonable amount of time, a thorough refactoring of the code might be necessary, focusing specifically on runtime optimization. This mandatory next step is however outside of the scope of this thesis. The response time of the system is quite high but still tolerable for conducting a user study if every user only makes few queries to the system. We therefore used the system in its current form and postponed runtime optimization to future work.

We can already think of the following ways for improving the runtime further:

As downloading the original news articles from the KnowledgeStore is one of the most time consuming query types, buffering all news articles locally might yield some additional performance improvements.

Moreover, using a KnowledgeStore instance which contains only news articles from the last few days and which is periodically updated (by processing new articles with the NewsReader system and removing information about old articles) would decrease the overall amount of content in the KnowledgeStore. This would presumably improve the average query execution time as less RDF triples need to be considered (especially during the initial search step). The smaller amount of RDF triples also implies that there would be less matching events per query which could potentially make the approach of limiting the number of events for further processing obsolete. Finally, this would also have the nice side-effect of making system responses more relevant with respect to temporal recency – an aspect which has not been dealt with explicitly in this thesis.

A third way for optimizing the overall processing time would be the re-formulation of the feature computation by using inherently parallel algorithms. This might require some major design effort, but we are confident that especially for the computationally demanding features some significant runtime improvements can be achieved by algorithmic means.

4. Evaluation

After having validated the system on the given data sets, it is also important to evaluate how well the system generalizes to other users and other queries. Due to the relatively small size of the data sets used for training the classifier and the regressors, it is especially important to analyze whether they are overfitting on these data sets. We therefore conducted a user study to evaluate the system as a whole in an online scenario. We chose to evaluate the system as a standalone system and not as part of the social dialog system. One can therefore categorize this evaluation approach as "intrinsic". This evaluation approach was chosen because it allows the system to be evaluated in more depth. An extrinsic evaluation of the NewsTeller's effect on the overall social dialog system would of course be a reasonable next step.

The following sections present the user study in more detail: Section 4.1 describes the design of the user study and the general setup. Section 4.2 presents the results obtained in the study and Section 4.3 discusses their implications.

4.1. User Study Design

The user study was designed with two questions in mind:

- How well does the ranking approach work in general?
- How much can information about the user's interests help to improve performance?

In order to evaluate the quality of the ranking approach in general, we compare it to a random baseline. In order to analyze the impact of user model information, we compare two versions of the NewsTeller system: One version performs the ranking without any access to the user's interests (the "no-UM addRemove" regressor from Section 3.5.2.4), whereas the other one takes these interests into account (the "UM addRemove v2" regressor from Section 3.5.2.4). We therefore compare three system configurations: Random, Ranking-noUM, and Ranking-UM.

We decided to conduct this user study as online survey in order to increase the number of potential participants and to simplify the process of conducting the study. In the survey, participants were first asked to indicate two to six keywords representing their general long-term interest. Then, they were asked to formulate a total number of five queries to the system, each consisting of up to three keywords. After being entered by the participants, each query was sent to the NewsTeller system and the responses of the three system configurations were generated and displayed to the participants. Participants were then asked to rate the relevance of each system response with respect to both

their query and their general interests. Relevance was measured on the scale Irrelevant – Partially Relevant – Relevant – Very Relevant. An additional option System Did Not Respond was also given and participants were asked to select it if the system did not respond to their query. In addition to this relevance judgment, participants were asked to select the system response they judged to be the best one. Moreover, at the end of the survey, participants were asked to select the system with the overall best performance.

As one can see, we tried to mimic the initial data collection as closely as possible by asking for the same number of keywords and by using the same relevance scale. This was done for two reasons: On the one hand, by using the same number of keywords, the interests and queries elicited in this user study are expected to have similar characteristics as the ones collected earlier for the UM data set. This way, the data collected in the user study should be comparable to the one used for training – which makes performance comparisons meaningful in the first place. On the other hand, using the same relevance scale as in the training set allows for the application of the AV, $P_{>0}$, and $P_{>1}$ metrics that were used to evaluate the system's performance on the data sets.

The additional question about the best response to the query was included based on two motivations: Firstly, we assumed throughout the thesis that a "good" response to a user query has to be relevant. By comparing the participants' feedback to the "best response" question with their relevance ratings, we can analyze whether there is indeed a strong correlation of relevance and "goodness". Secondly, it might be the case that a participant gives a highest identical relevance rating to two system responses. By asking for the best response, one might capture differences in response quality that are not based on relevance.

Finally, by asking for the overall best system at the end of the survey, we try to get a "big picture" of system performance across all queries. Responses to this question might yield additional insight when compared to the results obtained for the individual queries.

We decided to let the participants interact with all three system configurations simultaneously instead of having three separate, sequential interactions. We used this simultaneous approach as it has several advantages when compared to the sequential approach:

In the sequential approach, participants first write some queries which are responded to by the first system, then some more queries which are handled by the second system, and finally a third set of queries which are dealt with by the third system. Each query to any of the systems takes some considerable amount of processing time. As we aimed for a maximum duration of 15 minutes for the complete survey, this means that only a limited amount of queries can be processed. In the sequential approach, this total number of queries would have to be divided between the three system configurations. This means that only a small amount of data could be collected per participant and system. Moreover, it is quite likely that the participants would formulate different queries for the different systems which reduces the comparability of the ratings.

If however participants interact with all three system configurations simultaneously, each system can be applied to every query. This yields more data points per system and also makes the retrieved ratings more comparable across systems. Furthermore, participants can directly compare the responses of the three system configurations and indicate

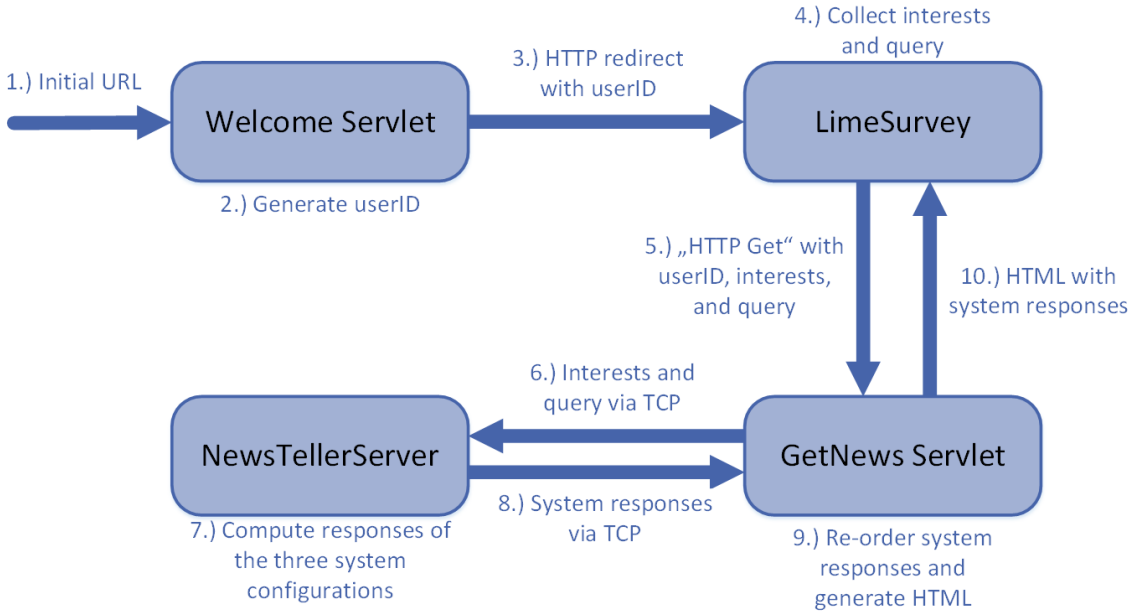

Figure 4.1.: Deployment of the user study.

which one they think is the best one. This can give additional insight when comparing the three system configurations.

In order to reduce effects based on the order in which the system responses are displayed to the participants, we counterbalanced the system order across all participants. System configurations were labeled as “System A”, “System B”, and “System C” during the survey. Based on an ID assigned to the participant at the beginning of the survey, the mapping of these three labels to the underlying system configurations was chosen. For instance, one participant might be presented with the systems in the order Random – Ranking-UM – Ranking-noUM while another participant might see them in the order Ranking-noUM – Random – Ranking-UM. By looping through the six different system permutations across all participants, we tried to make sure that all permutations appear (almost) equally often.

When designing this user study, we decided on purpose to give the participants as much freedom as possible in formulating their queries. By doing so, we hoped to get more realistic data about potential queries to the system. Statistics about the types of queries likely to happen “in the wild” could then be used to select the types of news feeds for filling the respective KnowledgeStore instance. This analysis is however out of the scope of this thesis.

Before we move on to the results obtained in the user study, let us quickly describe the deployment setup. Figure 4.1 illustrates how the NewsTeller system was connected to the user study web frontend.

Participants were provided with a link to the Welcome servlet (implemented in Java and running on a Tomcat server). This Welcome servlet assigned IDs to the participants based on a counter, i.e., the first participant was assigned ID 0, the second participant ID 1, the third participant ID 2, and so on. Participants were then redirected to the actual survey and their respective ID was forwarded as GET parameter in the survey URL. The survey itself was implemented using LimeSurvey[1]. After some introductory explanations, participants were asked to enter their general interests and to formulate queries. The system responses were displayed in a frame within the survey. The URL of this frame was generated on the fly based on the participant's ID, the indicated interests, and the respective query. This URL linked to the GetNews servlet which in turn forwarded both the query and the user interests to the NewsTellerServer application over a TCP connection. The NewsTellerServer executed the three system configurations for the given interests and query and returned the result to the GetNews servlet. The GetNews servlet then re-ordered the system responsed based on the participant's ID and returned the created HTML content. This was then displayed in the frame of the survey page. As the execution of the three system configurations was computationally demanding, the response time was up to three minutes. Participants were informed about this in the survey and were asked to be patient and to not reload the page.

See Appendix B.1 for screenshots of the different screens of the survey.

4.2. Results

In total, 130 participants began the survey. However, only 48 participants finished it which corresponds to about 36.92%. A part of this relatively low fraction of participants actually finishing the survey can be explained by restarts: One can sometimes observe two entries with the same age, gender and NLP experience as well as very similar interest keywords. Usually, the first of these entries is incomplete whereas the second one is complete. We interpret this as participants beginning the survey and later starting again from scratch. Due to the observed restarting effects, we only analyzed the 48 full responses in order to avoid duplications in our data.

Out of the 48 remaining participants, 29 were male and 19 were female. Their age ranged from 17 to 57 with a median of 26 and a mean of 29.02 (standard deviation: 9.4812). Most participants (30 of 48) were in their twenties, i.e,. between 20 and 30 years old. A majority of 31 participants indicated that they had no prior experience with natural language processing. On average, participants spent about 25 minutes with the survey – ten minutes longer than originally intended when the survey was designed.

On average, each participant listed 4.33 interest keywords. This number is considerably lower than the average number of 5.73 interest keywords observed in the UM data set. In total, 170 different interests were used. The most popular ones include:

[1] See `https://www.limesurvey.org/`

- music (16 times)
- sport (5 times)
- politics (4 times)
- football (3 times)
- cooking (3 times)
- Italy (3 times)
- soccer (3 times)

Note that "politics", "football", and "Italy" were also among the most popular interest keywords in the UM data set. Moreover, the most popular interest keyword "music" was used as an example keyword in the explanatory text of the survey. The other two example keywords "Trento" and "John Doe" do not appear in the list of popular interest keywords. This indicates that participants were somewhat biased by the examples given in the explanatory texts but that this bias was not too strong.

As each of the 48 participants formulated five queries, there are 240 queries in total. They contain a total number of 464 different keywords. The most popular keywords being used in the queries were the following:

- music (8 times)
- election (7 times)
- Italy (6 times)
- concert (4 times)
- economy (4 times)
- new (4 times)
- football (4 times)
- Paris (4 times)

Note that the keyword "election" was used as an example keyword in the explanatory text for creating a query. The other two example keywords "Trento" and "John Doe" do not appear in the list of popular query keywords. Again, participants seem to be slightly biased by the explanatory text.

On average, each query contained 2.35 keywords. Figure 4.2 illustrates the distribution of the query length (measured in number of keywords). It compares the query length distribution observed in the user study to the query length distribution observed in the UM data set where annotators used on average 2.23 keywords per query. A χ^2 significance test with a significance level of $\alpha = 0.05$ showed that the difference between these two

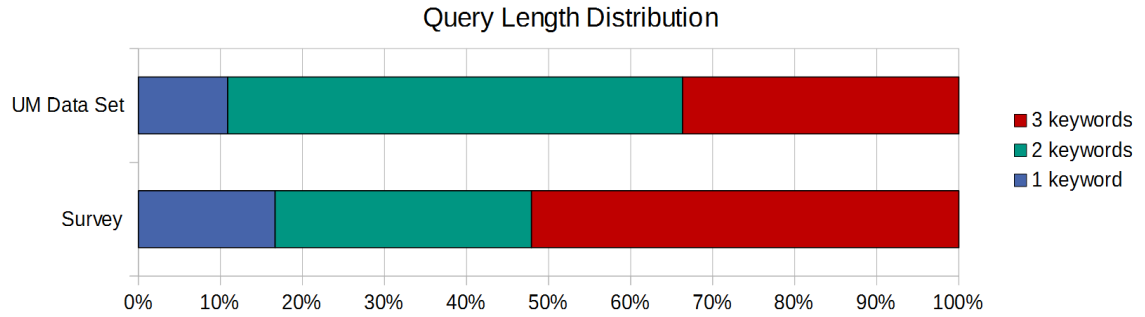

Figure 4.2.: Figure illustrating the distribution of different query lengths in the survey and in the UM data set.

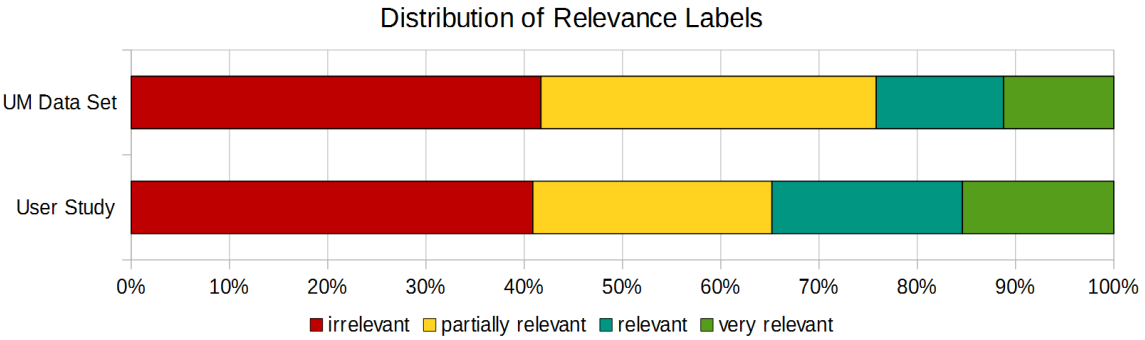

Figure 4.3.: Distribution of relevance labels for both the UM data set and the data collected in the user study.

distributions is significant ($\chi^2 = 56.88$, $p = 4.45 \cdot 10^{-13}$).

Just like in the UM data set, also in the user study we observed that some users tended to reuse their interest keywords in their queries. Again, we only looked at literal matches, not at semantically closely related words. In total, 33 of the 240 queries were constructed using at least one keyword from the user interests. This corresponds to 13%, a number slightly lower than the one observed in the UM data set (which was 19%). Most of the participants (23 of them) did not reuse their interest keywords to formulate queries. A smaller number of 15 participants reused their interest queries for one query, three participants for two queries, and four participants for three queries. This is largely in line with the observations from the UM data set, where three of eleven annotators re-used their interest keywords in more than 60% of the queries and where the remainder of annotators did so only maximally in one of their queries. As the number of queries generated per person differs greatly in the two settings (12 queries per annotator for the UM data set vs. five queries per participant in the user study), it did not seem to be meaningful to conduct a significance test as the numbers are not really comparable.

Figure 4.3 compares the distribution of relevance labels between the UM data set and the user study . We did not distinguish individual system configurations and ignored the

System A	System B	System C	Frequency
Random	Ranking-noUM	Ranking-UM	10
Ranking-UM	Random	Ranking-noUM	4
Ranking-noUM	Ranking-UM	Random	6
Random	Ranking-UM	Ranking-noUM	11
Ranking-UM	Ranking-noUM	Random	8
Ranking-noUM	Random	Ranking-UM	9

Table 4.1.: Table showing the six different permutations along with their respective frequency among the 48 full responses.

Configuration	System A	System B	System C
Random	21	13	14
Ranking-noUM	15	18	15
Ranking-UM	12	17	19

Table 4.2.: Table showing the results of the counter-balancing.

System Did Not Respond category as it does not appear in the UM data set. One can see that the proportion of Partially Relevant system responses is notably smaller in the data collected in the survey and that at the same time the proportion of Relevant and Very Relevant responses is slightly larger. The difference between the two distributions proved to be statistically significant in a χ^2 test ($\chi^2 = 39.82, p = 1.16 \cdot 10^{-8}$). One could see this as indication that the average of the three configurations tested in the user study selects more Relevant and Very Relevant events and less Partially Relevant events than the simple heuristic used for selecting the events in the UM data set. This comparison is however questionable in the sense that the annotators of the UM data set labeled about 11 system responses per query whereas the participants in the user study labeled only three system responses per query (one per system configuration). Moreover, in the UM data set, 50% of the events were selected randomly, whereas in the user study only one third of the system responses was based on a random event selection. We therefore do not analyze this comparison in any further detail.

As argued in the previous section, we presented the three system configurations to each participant in a different order, trying to average out ordering effects. Due to the high number of incomplete responses, this counter-balancing did however not work perfectly. Table 4.1 shows the six different system permutations and their respective frequency in the 48 complete responses. Note that an optimal counter-balancing would have assigned 8 participants to each of the permutations. Table 4.2 shows the overall results of this counter-balancing. It reads as follows: "The Random system was presented as *System A* to 21 participants, as *System B* to 13 participants, and as *System C* to 14 participants." Note that in a perfectly balanced setting, each configuration would have been presented under each name exactly 16 times. As one can see, although the distribution across permuta-

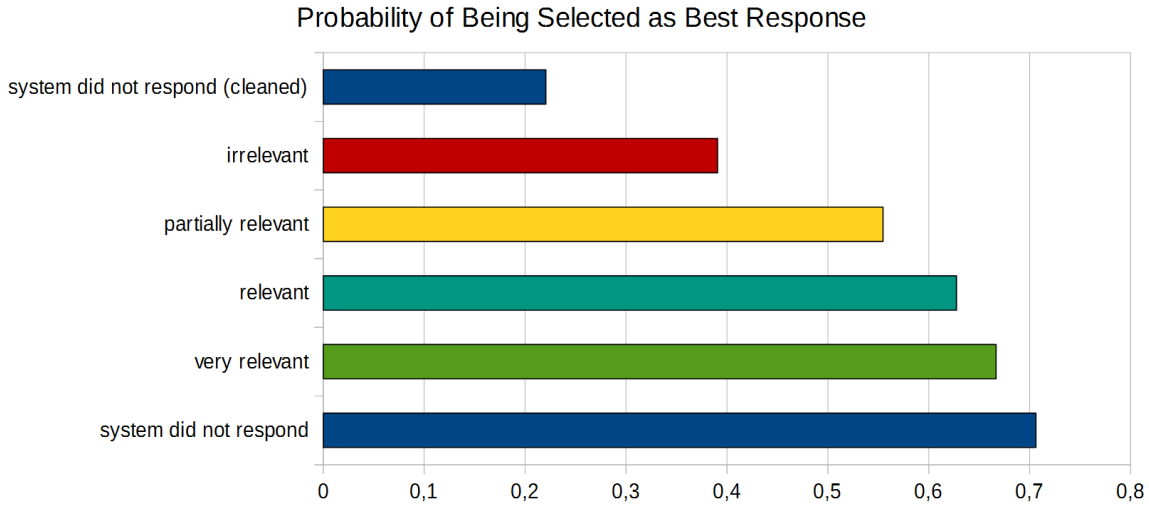

Figure 4.4.: Probabilities of being selected as "best response" given the relevance label.

tions in Table 4.1 is skewed, the overall counter-balancing still works reasonably well.

Before evaluating the system configurations themselves, we investigated whether our assumption holds true that a relevant response is perceived as a good response. For each relevance label, we computed the conditional probability that a system response with the given label is selected as best response. Figure 4.4 illustrates these probabilities. By visual inspection, one can observe a certain trend: The higher the relevance value of a system response, the higher is its probability of being selected as best response. Note that the gap is especially large between Irrelevant and Partially Relevant responses and relatively small between Relevant and Very Relevant responses. The System Did Not Respond class has the highest probability of being selected as best response. This seems to be due to the fact that if all three system configurations do not respond to the user query, the participants are still required to select a best response. As this happens quite frequently (for 42 of 240 queries, i.e., 17.50% of all queries), the probability of the System Did Not Respond class is artificially increased. If we remove the 42 queries for which all system configurations do not produce a response, this probability drops even below the probability for Irrelevant responses. This updated probability is shown as "System Did Not Respond (cleaned)" in Figure 4.4.

The overall trend of increasing probabilities for increasing relevance values seems to be confirmed by a high correlation value of $\rho = 0.9534$. Note that we did not take into account the System Did Not Respond class as it does not have a canonical class index. It is also excluded from the statistical χ^2 tests we conducted to further analyze the differences between the relevance labels. Table 4.3 shows the results of these significance tests. As one can see, all differences between the Irrelevant class and any other class are statistically significant. Also the difference between Partially Relevant and Very Relevant responses is significant. The differences between Partially Relevant and Relevant responses as well as between Relevant and Very Relevant responses are however not statistically significant. As the class borders are in these cases not well defined, this is not

Comparison	χ^2	p
Irrelevant – Partially Relevant	14.46	0.0001
Irrelevant – Relevant	24.02	$9.55 \cdot 10^{-7}$
Irrelevant – Very Relevant	25.92	$3.57 \cdot 10^{-7}$
Partially Relevant – Relevant	2.19	0.1392
Partially Relevant – Very Relevant	4.11	0.0426
Relevant – Very Relevant	0.53	0.4654

Table 4.3.: Table showing the results of various χ^2 tests on the differences observed in Figure 4.4.

very surprising. Overall, it seems like the assumption that relevant responses are good responses seems to be valid. However, one should keep in mind that participants were asked to pick the best response immediately after having labeled the responses with respect to their relevance. The participants might thus have been biased to mainly focus on relevance when selecting the best response.

In order to validate our assumption that not responding to a query is better than responding with an Irrelevant event, we looked at all queries where all of the systems responded with either an Irrelevant event or not at all. We only considered queries where not all system responses were labeled identically. As it turns out, there are only 23 such queries. Therefore, the following results should be taken with caution.

Whenever the participants had to chose between Irrelevant responses and non-responses, they selected an Irrelevant response as best response in 60.86% of the cases, and a System Did Not Respond in 43.47% of the cases. These numbers add up to a bit more than 100%, indicating that participants sometimes selected both types of responses as best responses. The observed differences are not statistically significant ($\chi^2 = 2.92, p = 0.0875$). It thus seems like the underlying assumption of "no response is better than an Irrelevant response" does not necessarily hold true – in fact, we observe a weak tendency in the opposite direction.

After having analyzed the participants, interests, queries, and the permutations, let us now turn to the actual ratings of the system responses. Figure 4.5 illustrates the distribution of relevance labels for all three of the system configurations. One can see that both ranking-based approaches have a lower number of Irrelevant responses and a higher number of System Did Not Respond than the Random baseline. We attribute this to the thresholding mechanism used for both ranking approaches. The Random baseline responded to 198 of 240 queries. If the Random baseline responds with an event and one of the ranking-based configurations does not, this means that there exists at least one event that survived the filtering step – but that the thresholding prevented the top event from being output. Out of the 198 queries that yielded a non-empty set of events, the Ranking-noUM configuration did not respond to 40 queries which corresponds to about 20.20%. The Ranking-UM configuration did not respond to 28 of these 198 queries which corresponds to about 14.14%. In both cases, the percentage of eliminated responses is con-

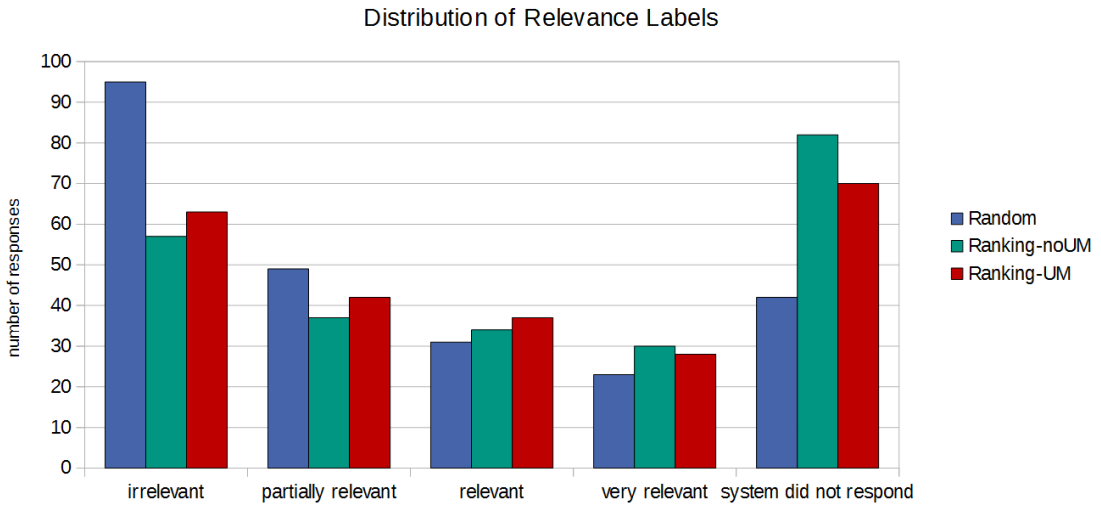

Figure 4.5.: Distribution of relevance labels for the three system configurations.

Configuration	**AV**	$P_{>0}$	$P_{>1}$
RANDOM	0.9090	0.6042	0.2250
RANKING-NOUM	1.2342	0.7625	0.2667
RANKING-UM	1.1765	0.7375	0.2708

Table 4.4.: Table showing the metrics AV, $P_{>0}$, and $P_{>1}$ for the three system configurations.

siderably higher than estimated on the UM data set (where both numbers were around 3-5%).

Table 4.4 shows the three metrics AV, $P_{>0}$, and $P_{>1}$ for the three system configurations. Surprisingly, the RANKING-NOUM configuration achieves the highest values for both the AV and the $P_{>0}$ metric. For the $P_{>1}$ metric, the RANKING-UM approach performs best. In general, both ranking approaches perform better than the RANDOM baseline. Note that the values for all three of the metrics are somewhat lower than on the UM data set. But as argued before, the given metrics are not well suited for comparisons across different data sets. One would need their normalized counterparts – which are unfortunately not computable on the ratings from the user study. For instance, to compute the ANV metric, one needs to know the highest ground truth label for any event for the given query. In the current setting, however, the participants only labeled the responses returned by the three system configurations, therefore this highest ground truth label is unknown. Hence, we unfortunately cannot directly compare these results to the ones obtained in Section 3.5 in a meaningful way.

Although not necessarily interpretable, there are however some interesting observations we can make: Whereas on the UM data set, the RANKING-UM approach outperformed the RANKING-NOUM approach with respect to all metrics, their relationship seems to be reversed in the user study. Moreover, the values of the $P_{>1}$ metric are considerably

lower than on the UM data set where we were able to achieve values of up to 0.4812 for this metric.

In order to see which of the observed effects are significant, we performed several χ^2 significance tests with a significance level of $\alpha = 0.05$. With respect to the overall label distribution shown in Figure 4.5, we found that the differences between the RANDOM baseline and the RANKING-NOUM approach are statistically significant ($\chi^2 = 58.65, p = 5.56 \cdot 10^{-12}$). Also the differences between the RANDOM baseline and the RANKING-UM configuration are statistically significant ($\chi^2 = 32.69, p = 1.38 \cdot 10^{-6}$). The difference between the two regression-based configurations RANKING-NOUM and RANKING-UM are however not statistically significant ($\chi^2 = 3.46, p = 0.4838$).

As the SYSTEM DID NOT RESPOND class is ignored for the computation of the AV metric, we removed all of the non-responses from the respective distributions and performed statistical tests on the remaining data points to investigate the differences with respect to the AV metric. The statistical tests indicated that the differences between the RANDOM baseline and the RANKING-NOUM configuration are significant ($\chi^2 = 15.63, p = 0.0013$). Also the differences between the RANDOM baseline and the RANKING-UM configuration showed to be significant ($\chi^2 = 11.73, p = 0.0084$). The observed slight differences between the two regression approaches however were not statistically significant ($\chi^2 = 0.74, p = 0.8643$). The high value of p indicates that the two observed label distributions are likely to be generated by the same underlying process. This means that the H_0 hypothesis is very probable which states that there is no difference between the underlying distributions.

With respect to the $P_{>0}$ metric, a similar picture emerges: Again, both regression-based approaches are significantly better than the RANDOM baseline ($\chi^2 = 25.16, p = 5.28 \cdot 10^{-7}$ for RANKING-NOUM, and $\chi^2 = 17.84, p = 2.40 \cdot 10^{-5}$ for RANKING-UM). However, between the two regression-based approaches, no significant difference could be found ($\chi^2 = 0.83, p = 0.3628$).

When looking at the $P_{>1}$ metric, things are less clear: None of the differences could be shown to significant. There seems to be a weak tendency for both the RANKING-NOUM ($\chi^2 = 2.39, p = 0.1222$) as well as the RANKING-UM approach ($\chi^2 = 2.89, p = 0.0891$) to be better than the RANDOM baseline, however none of these differences is significant. The observed difference between the two ranking approaches seems to be mainly caused by random noise ($\chi^2 = 0.02, p = 0.8839$), as the p value of almost 90% is very large.

Note that for the computation of both the $P_{>0}$ and the $P_{>1}$ metric, the SYSTEM DID NOT RESPOND class was assigned an artificial class index of 0.5 (cf. Sections 3.5.1.3 and 3.5.2.4).

For each of their queries, participants were also asked to indicate which system gave the best response. Multiple selections were allowed if two or more systems gave an identical best response. Participants selected the RANDOM configuration in 52.50% of the cases, the RANKING-NOUM configuration for 57.50% of the queries, and the RANKING-UM approach in 60.83% of the cases. Figure 4.6 illustrates these numbers. As one can easily see, these three percentages add up to more than 100%. This indicates that participants on average selected more than one system as the one with the best response. This is however no surprise, given that for 42 of 240 queries, all three systems did not respond. In this case, participants typically selected all three systems for the best response.

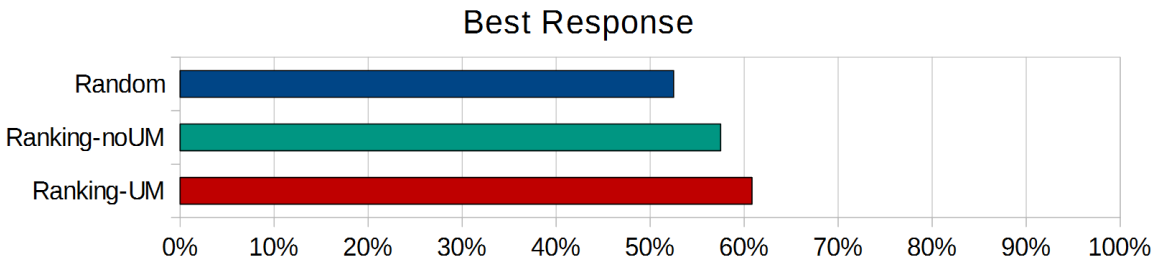

Figure 4.6.: Distribution of "best response" ratings.

Some χ^2 significance tests showed that only the difference between the RANDOM baseline and the RANKING-UM configurations is statistically significant ($\chi^2 = 6.68, p = 0.0097$). The differences between the RANDOM baseline and the RANKING-NOUM configuration ($\chi^2 = 2.40, p = 0.1209$), and between the two regression-based approaches ($\chi^2 = 1.09, p = 0.2962$) are not significant. This indicates that participants chose the response of the RANKING-UM configuration significantly more often as best response than the response of the RANDOM baseline. It is however not quite clear how to place the RANKING-NOUM configuration: Its value lies between the two extremes, but the differences to neither of them is statistically significant. It could therefore have the same underlying distribution as any of the other two systems.

It is a bit surprising that with respect to the "best response", the UM-based approach seems to be the best whereas with respect to the overall label distribution, it seems to be slightly inferior to the RANKING-NOUM configuration. However, as none of these differences between the two ranking-based systems are statistically significant, they may be (at least partially) attributed to random noise.

The last question in the survey asked participants to select the system with the overall best performance. Figure 4.7 illustrates the result. Each participant had to pick exactly one system. Almost half of the participants (22 of 48, i.e., 45.83%) selected the RANKING-UM configuration, 11 participants (i.e., 22.92%) selected the RANKING-NOUM configuration, and the remaining 15 participants (i.e., 31.25%) the RANDOM configuration. It is surprising that the RANKING-UM configuration was selected by twice as many participants as the RANKING-NOUM configuration although we could not find any statistically significant differences between the two approaches in any of our previous analyses. A χ^2 test showed that the observed distribution is not significantly different from a uniform distribution across the three system configurations ($\chi^2 = 3.88, p = 0.1441$). Again, we can only observe a weak tendency, but no significant effects.

In order to investigate whether there were any learning effects, we also compared the label distribution observed for the first two queries to the label distribution observed for the last two queries. We did not differentiate among the different system configurations in this case. Figure 4.8 compares the two distributions. They correspond to an AV of 0.9617 for the first two queries and an AV of 1.2271 for the last two queries. A χ^2 significance test showed that this difference in the AV metric is statistically significant ($\chi^2 = 16.46, p = 0.0009$), i.e., that participants gave significantly higher relevance ratings

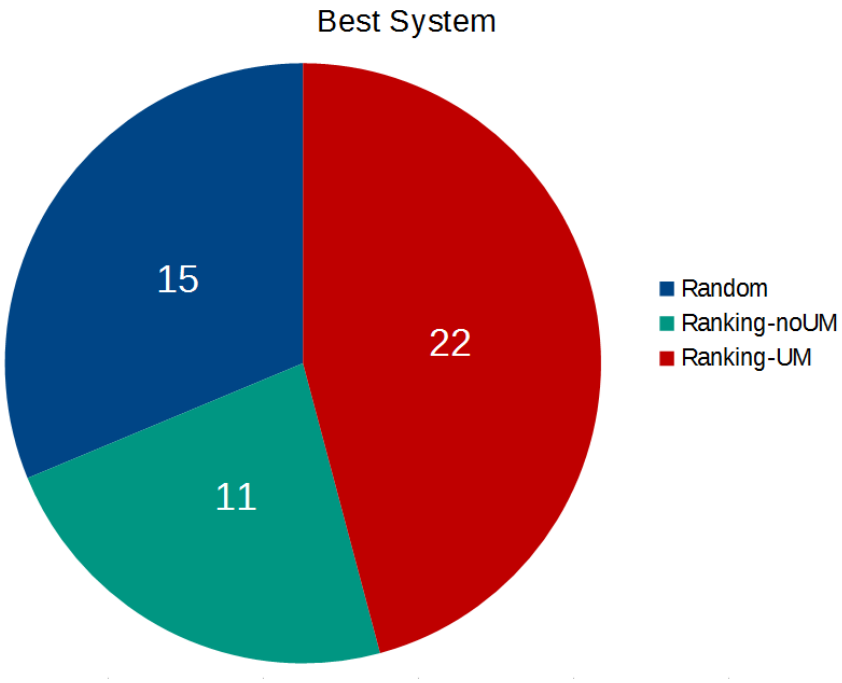

Figure 4.7.: Distribution of "best system" ratings.

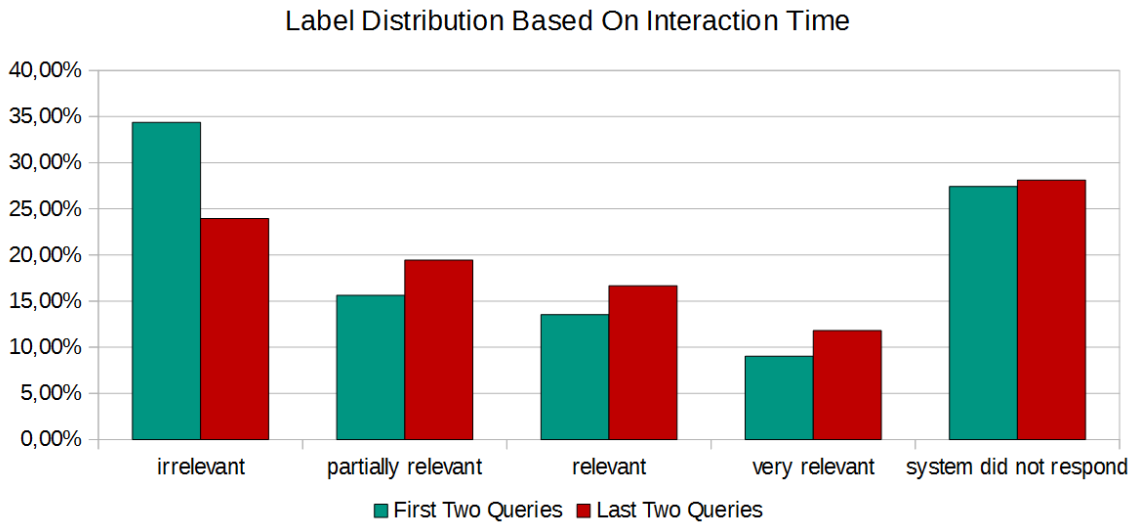

Figure 4.8.: Distribution of relevance labels for all systems for the first two and the last two queries, respectively.

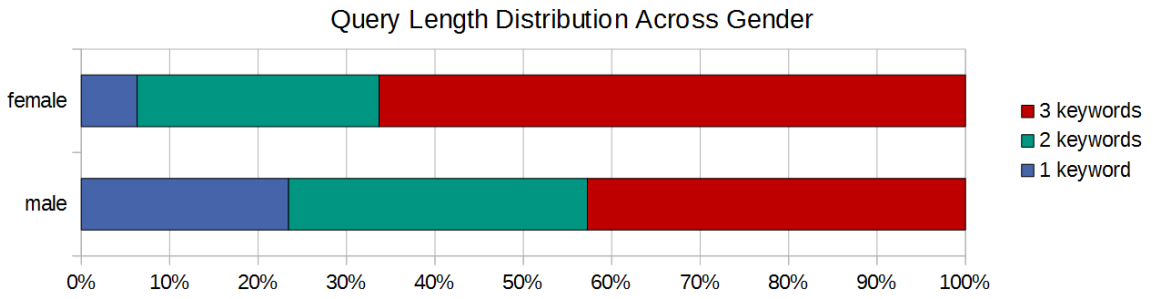

Figure 4.9.: Distribution of query length in number of keywords differentiated by gender.

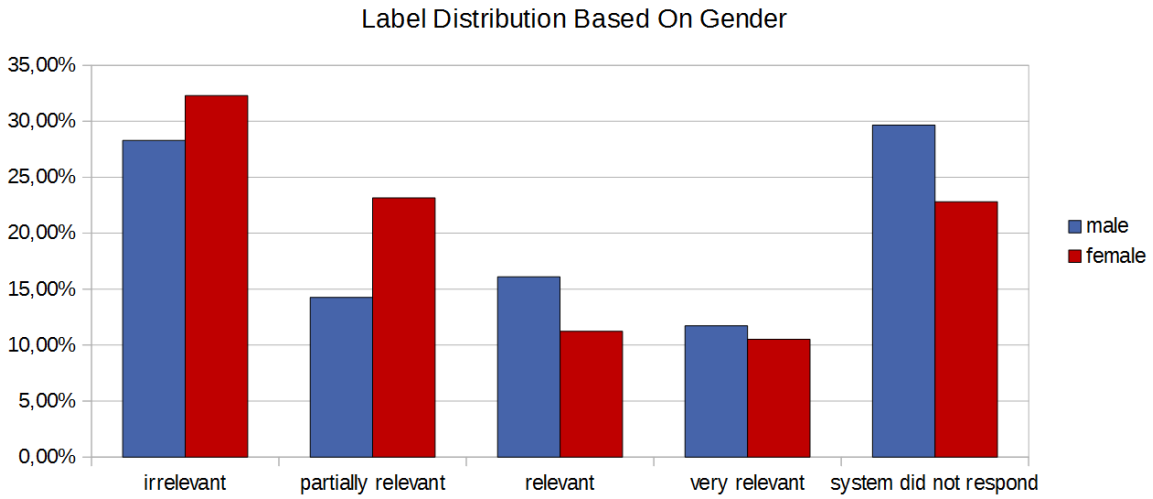

Figure 4.10.: Distribution of relevance labels for all systems differentiated by gender.

towards the end of the survey. It is however not quite clear why they did so. Maybe this is because participants learned how to formulate queries which are likely to produce good system responses. It could however also be the case that participants simply lowered their expectations in the course of the survey. As the proportion of System Did Not Respond is almost identical, the latter explanation seems to be more likely – if participants had learned how to formulate good queries, one would expect the percentage of non-responses to decrease.

We also briefly looked at differences along the gender dimension. Only the interesting effects are mentioned here, all other analyses can be found in Appendix B.2.1.

Figure 4.9 illustrates the distribution of query lengths (measured in number of keywords) for male and female participants. On average, male participants used 2.19 keywords per query whereas female participants used 2.60 keywords per query. This difference proved to be statistically significant ($\chi^2 = 25.38, p = 3.08 \cdot 10^{-6}$) which indicates that female participants tended to use more keywords per query than male participants.

Moreover, we analyzed differences in the overall label distribution aggregated over all system configurations. Figure 4.10 shows the respective distributions of relevance labels for male and for female participants. They correspond to an AV of 1.16 for male partic-

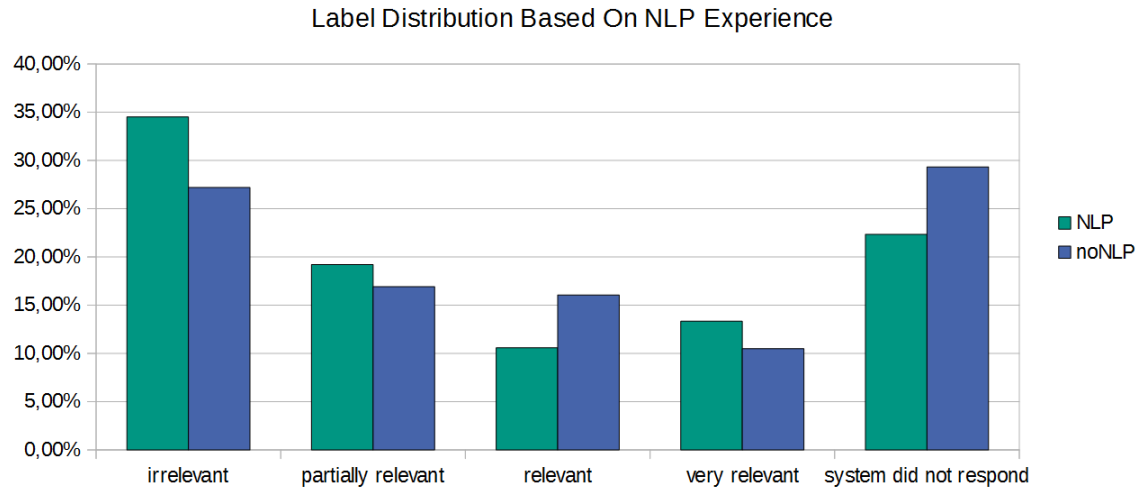

Figure 4.11.: Distribution of relevance labels for all systems differentiated by NLP experience.

ipants and an AV of 1.00 for female participants. The observed differences with respect to the AV metric are statistically significant ($\chi^2 = 18.32, p = 0.0004$). This indicates that male participants gave on average higher ratings than female participants.

Finally, we also looked at differences with respect to the participants' experience in the NLP field. Again, we only report interesting differences. All other results can be found in Appendix B.2.2.

Figure 4.11 illustrates the overall rating behavior of participants with prior experience in the NLP field ("NLP") and participants without any knowledge about NLP ("noNLP"). Participants with a background in NLP gave an average rating of 1.0354 whereas participants without any knowledge in the NLP field gave an average rating of 1.1280. This difference showed to be statistically significant ($\chi^2 = 24.51, p = 1.96 \cdot 10^{-5}$) indicating that users familiar with NLP research might be less easy to satisfy than naïve users. The lower number of non-responses for the "NLP" group might indicate that people with an NLP background have a better intuition about how to formulate queries which are likely to succeed.

Some participants gave us feedback after having participated in the user study. They criticized that there was no feedback form at the end of the survey. Moreover, with respect to the question for the best response of a system, they suggested to add another option "none of the systems gave a relevant response". Some participants also reported that they had difficulties to understand the task to be performed and the type of input they were supposed to make. For instance, the name "Jon Doe" was used in the examples that explained how to formulate interests and queries. In American culture, it refers to an unknown person (somewhat similar to the German "Max Mustermann"). To some participants, this term was unknown, hence they got confused. Furthermore, some participants were not sure what types of keywords were allowed. All of this feedback indicates that

the design of this user study was not perfect. This might have biased the results of this user study.

4.3. Discussion

After having analyzed the results of the user study, it is now time to discuss their implications.

First of all, we observed that there are some differences between the data collected for the UM data set and the data collected in the user study: In the user study, the average number of interest keywords per participant is considerably lower than in the UM data set. Moreover, participants used more keywords per query than the annotators in the UM data set. When analyzing the most popular keywords used for interests and in queries, we found that study participants were somewhat biased by the examples given in the explanatory text. This did not happen in the collection of the UM data set, as no examples were given to the annotators participating in this data collection. All of these observations indicate that the input to the NewsTeller system differed to a certain extent between training and evaluation.

The significant differences with respect to the relevance label distributions of the three systems seems to be largely due to the thresholding used with both ranking approaches. A visual inspection of Figure 4.5 can give a first hint in this direction: Both ranking-based systems have a considerably higher number of System Did Not Respond and a considerably lower number of Irrelevant responses. Also the results with respect to the $P_{>0}$ and $P_{>1}$ metrics support this hypothesis: For the computation of the $P_{>0}$ metric, System Did Not Respond is counted to the positive class (as it is considered to be better than Irrelevant). Here, both ranking-based approaches perform significantly better than the Random baseline. For the computation of the $P_{>1}$ metric, however, both Irrelevant and System Did Not Respond belong to the negative class (together with Partially Relevant). Here, no significant effects could be observed. We therefore suppose that the differences observed in the overall distribution are mainly due to differences with respect to the Irrelevant and System Did Not Respond classes – which are likely caused by the thresholding.

The results with respect to the AV metric come as a little surprise: Although both ranking-based approaches significantly outperform the baseline, there is no significant difference between them. On the UM data set, the Ranking-UM approach outperformed the Ranking-noUM approach with respect to this metric. Although we did not use any tests of statistical significance for the results obtained on the UM data set, the difference between an AV of 1.4342 for the Ranking-noUM approach and an AV of 1.5763 for the Ranking-UM approach is definitely notable. In the user study, however, the Ranking-noUM configuration reaches a slightly higher AV than the Ranking-UM configuration. The results from the user study indicate that both ranking-based systems are significantly better than the baseline with respect to the AV metric, but that there is not necessarily

any performance difference between them.

The same observation holds true for the $P_{>0}$ metric: Both ranking-based approaches are significantly better than the baseline, but there is no significant difference between them. Based on the results obtained on the UM data set, this result was to be expected: Also there, both ranking-based approaches outperformed the baseline with respect to the $P_{>0}$ metric but achieved almost identical values for this metric.

What comes as a much greater surprise are the results with respect to the $P_{>1}$ metric: In the user study, no significant differences between any two system configurations could be observed. On the UM data set, in contrast, we observed large performance differences between the Random baseline and the Ranking-noUM system, as well as between the Ranking-noUM and the Ranking-UM systems. The results obtained on the UM data set seemed to indicate that the ranking helps not only to avoid Irrelevant events but also to select Relevant and Very Relevant ones. Moreover, the old results suggested that using information about the users' general interests can further help to identify Relevant and Very Relevant events. When looking at the results of the user study, however, it looks like both ranking approaches do not perform significantly better at identifying Relevant and Very Relevant events than a random selection.

The fact that the response of the Ranking-UM system was selected significantly more often as best response than the response of the Random baseline seems to indicate that the ranking approach indeed does work better than a random selection. This effect cannot be explained with the thresholding alone, as the results indicated that participants did not prefer System Did Not Respond to Irrelevant responses. We therefore hypothesize that the significant difference between the Ranking-UM approach and the Random baseline means that the answer quality of the Ranking-UM approach in general is better than the answer quality of the Random baseline.

Also with regarding the "best system", the results seem to indicate that the Ranking-UM system worked best. However, the observed distribution does not significantly differ from a uniform distribution, therefore one cannot interpret too much into these results. It also seems counter-intuitive that participants preferred the Random baseline to the Ranking-noUM configuration which outperformed the former approach significantly with respect to the AV and $P_{>0}$ metrics. One explanation for this observed (insignificant) effect would be the following: Participants were asked to evaluate the overall performance of the system, but they were not able to remember all system responses in detail. Therefore, they might have been biased by the responses to the last query or they might have just randomly selected one of the options.

The observed effects with respect to gender and NLP background are interesting, but not crucial to our research at this point, and are therefore not discussed any further.

Instead, we would like to spend some more time with the following question: Why are the results found in the user study so different from the results obtained on the UM data set in the previous chapter?

On the UM data set, it seemed that there is a clear order of systems with respect to performance: The Ranking-UM approach always performed best, the Random baseline performed worst, and the Ranking-noUM approach lay somewhere in between the two other systems (but closer to the Ranking-UM system than to the Random baseline). We therefore expected to observe a similar ordering effect also in the user study. What we however found is that there seems to be no significant difference between the two ranking-based approaches. Moreover, all three approaches seem to behave in a similar way with respect to the $P_{>1}$ metric which is supposed to indicate how well the respective system is capable of selecting Relevant and Very Relevant events.

All these observed results seem to invalidate the results obtained earlier to some extent: It seems like the main difference between the ranking approaches on the one hand and the baseline on the other hand is the usage of thresholding to avoid outputting Irrelevant events. Why are these results so different from our expectations?

Firstly, and most likely, the regressors trained on the two ranking data sets might simply have overfit the data. The training data set for ranking is relatively small (only around 3,200 labeled events), and feature selection, hyperparameter optimization and training were performed on the same data set. It is therefore quite likely that the regressors overfit this particular data set. Although the UM data set was created by 11 different annotators, this sample of annotators might not have been representative for the overall population of potential users. If the set of annotators was for instance considerably more homogeneous than the overall population of potential users, it might have been easier to generalize across annotators within the UM data set than to new users that were not part of this data set. As noted earlier, there seem to be some important differences between the UM data set and the data gathered in this evaluation, e.g., with respect to the average number of keywords per query. These differences might contribute to the difficulty to generalize.

One can of course also explain the results by stating that the ranking approach developed in Section 3.5 is invalid and does not work in general. However, we think that there are some reasonably strong arguments against this explanation: Firstly, although we did not observe all the effects we expected, we still found that both ranking-based approaches had significantly higher values for the AV and $P_{>0}$ metrics than a random baseline. Although these effects can be largely attributed to the thresholding, this thresholding can only work reasonably well if the regressor makes reasonably good estimates of the relevance values. Moreover, it seems quite unlikely that a generally invalid approach would have been able to achieve the good results observed on the UM data set.

A third line of possible explanations targets the survey setup itself: Maybe the instructions for the participants were not clear enough and led them to wrong expectations about the systems' performance. If all of the systems for instance performed far worse than a participant expected, he or she might not be able to sufficiently differentiate between the systems. This would be in line with one participant who criticized that the systems only worked for "world news" and did not respond at all to any queries about his actual interests. These interests might have been too specific to find any match in the news articles of the KnowledgeStore instance. Therefore, by rerunning the survey with more detailed

information about which types of queries are likely to succeed, one might be able to reproduce the effects observed on the UM data set.

With respect to the two main questions from the beginning of this chapter, we can conclude the following:

- **How well does the ranking approach work in general?**
  It seems that the ranking approach in general performs significantly better than the RANDOM baseline. This seems to hold true mainly for the avoidance of IRRELEVANT events, but not for the selection of RELEVANT and VERY RELEVANT events. Moreover, the differences can be mainly attributed to the thresholding approach used as part of the regression-based configurations.

- **How much can information about the user's interests help to improve performance?**
  It seems that adding information about the users' interests does not help to improve performance compared to the regression approach without user model information. Although we were able to observe such effects on the UM data set, we were not able to reproduce them in the user study.

We conclude that the ranking approach, especially the user modeling approach requires some more research: It would be advisable to collect another data set similar to the UM data set. Then, one could investigate further how well the ranking approach generalizes to unseen data. Moreover, due to the larger available amount of training data, one could be more confident that the trained regressors generalize well. It might make also sense to repeat the user study but to give more detailed instructions about the types of queries that are likely to be successful.

5. Conclusion

5.1. Summary

In this thesis, we explored the personalized retrieval of news events based on a user query and a simple user model. Although this project can be seen as standalone research, it is embedded in the context of social dialog systems: The NewsTeller system developed in this thesis can be used as a part of a social dialog system for the task of initiating news-related small talk.

The NewsTeller system itself is implemented as a pipeline with four stages: After searching for events matching the query, these events are filtered according to a well-formedness criterion. The remaining events are then ranked according to their expected relevance and an output sentence is created based on the highest-scoring event.

The filtering problem turned out to be quite difficult to solve. This indicates that errors committed by the NewsReader pipeline while extracting events from news articles cannot be easily detected afterwards. Especially errors that were probably caused by a bad parse tree are hard to detect using only shallow features. Nevertheless, we managed to train a classifier reaching a precision of 63% and a recall of 59%. Although not perfect, this classifier helps to greatly improve the quality of the data passed on to the ranking step.

Based on data collected from 11 annotators, we trained two regressors for predicting the relevance of an event – one using only information about the event and the user query, and another one also taking into account information from the user model. Our experiments showed that both approaches perform considerably better than two simple baselines. The results also indicate that the additional use of information about the user model can help to select events with high relevance values. Experiments with relevance thresholds for determining when the system should rather not output its highest scoring event yielded ambivalent results: It seems that this thresholding mechanism can help to avoid outputting Irrelevant events, but at the same time the optimal threshold choice seems to be highly dependent on the underlying data set.

The final system was then evaluated with a user study in which participants interacted with three different versions of the system. The results of this survey confirmed our hypotheses only partially: It seems that the regression-based approaches perform significantly better than a random baseline. However, this difference can be largely attributed to the thresholding procedure. Moreover, the regression-based approaches do not outperform the random baseline when it comes to the selection of Relevant and Very

RELEVANT events. This contrasts sharply with our earlier results where we observed considerable differences between the systems in this aspect. We hypothesize that the results obtained in the user study did not confirm our expectations because the regressors were overfitting the data sets and failed to generalize. It might therefore be worthwhile to collect a larger data set for the ranking task and to reapply the existing machinery to this larger data set.

When looking at the big picture, we observe that the NewsTeller system (and thus our approach to the ranking problem) seems to work in general. However, it does not work very well, yet: Performance of both the filtering and the ranking components on their respective data sets leaves still room for improvements, the processing time of the system is far too high, and the results of the user study indicate that the system does not generalize well to the application "in the wild". All these issues should be addressed in future research. Nevertheless, we think that our results are promising: We are able to achieve reasonable performance on the data sets with relatively simple approaches and features. Moreover, both ranking-based approaches proved to be significantly better than a random baseline in the user study. We therefore think that this line of research is worth to be continued in the future.

5.2. Future Work

The work presented in this thesis leaves plenty of open ends which can be explored in future research. They can be grouped into three categories: ironing out shortcomings of the current implementation, investigating observed effects in more depth, and extending the functionality of the system.

As already mentioned in some parts of the thesis, the current implementation of the system has several imperfections and weaknesses that should be corrected.

First and foremost, one should mention the processing time which is still way too high for the system to be used as part of a dialog system. In Section 3.7, we already gave some ideas how the response time of the system could be improved: by using a smaller KnowledgeStore instance, by buffering the original news articles locally, and by algorithmically improving the computation of features.

If not being rendered unnecessary by the runtime improvements, the random selection of a fixed number of events at the end of the search step could be replaced by a more targeted heuristic-based approach.

Also the summary generation is implemented in a very naive way right now and could be easily improved. One could improve the detection of sentence boundaries, introduce pronoun replacement based on coreference resolution, and use some simple heuristic to select the sentence to be output for events with multiple mentions.

Furthermore, the simple way in which the filtering approach was generalized to multiple keywords might have to be revisited and potentially be replaced with a more complex approach.

Moreover, as argued above, the NewsTeller system fails to generalize well to unseen users and queries. We assume that this is mainly due to the small size of the training set. Collecting a larger data set and retraining the system might help to improve its performance when evaluated in an online setting.

Some of the effects observed earlier in the thesis might also be worth a more thorough investigation. This includes for example the question of how well the filtering problem can be solved and whether some other approach can help to increase classification performance in this context. Also the use of a threshold at the end of the ranking process yielded ambivalent results which urge for further investigations.

Finally, there are also many ways in which the current system could be extended.

Most importantly, once the runtime issues are largely solved, the NewsTeller system needs to be integrated with the social dialog system. It would make sense to then also evaluate it extrinsically, i.e., by its impact on the perceived conversation quality of the dialog system in general.

Moreover, one could replace the sentence extraction approach used for summary generation by an approach based on natural language generation.

Also extending the scope of the system to multiple dialog turns by dealing with follow-up questions about the presented news events would be an interesting potential line of research.

Furthermore, allowing for weighted keywords both in the user query and the user model might help to improve the system's performance. Automatically adjusting these weights for the user's interests and potentially also inferring new interest keywords would then be a potential next step.

Finally, also a proactive scenario in which the system presents news events without a prior user query would be an interesting extension to the NewsTeller system.

As one can see, there are many directions in which future research could be conducted based on the work presented in this thesis.

Bibliography

[1] Rodrigo Agerri, Eneko Agirre, Itziar Aldabe, Begoña Altuna, Zuhaitz Beloki, Egoitz Laparra, Maddalen López de Lacalle, German Rigau, Aitor Soroa, and Rubén Urizar. Newsreader project. In *30th Conference of the Spanish Society for Natural Language Processing (SEPLN)*, 2014.

[2] T. Asfour, K. Regenstein, P. Azad, J. Schröder, and R. Dillmann. Armar-III: A humanoid platform for perception-action integration. In *2nd International Workshop on Human-Centered Robotic Systems (HCRS)*, pages 0–0, 2006.

[3] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. *Dbpedia: A nucleus for a web of open data*. Springer, 2007.

[4] Sabarish Babu, Stephen Schmugge, Tiffany Barnes, and Larry F Hodges. "what would you like to talk about?" an evaluation of social conversations with a virtual receptionist. In *Intelligent Virtual Agents, 6th International Conference*, 2006.

[5] Timothy Bickmore and Julie Cassell. Small talk and conversational storytelling in embodied conversational interface agents. In *AAAI fall symposium on narrative intelligence*, pages 87–92, 1999.

[6] Timothy Bickmore and Justine Cassell. Social dialogue with embodied conversational agents. In *Advances in natural multimodal dialogue systems*, pages 23–54. Springer, 2005.

[7] Daniel Billsus and Michael J Pazzani. User modeling for adaptive news access. *User modeling and user-adapted interaction*, 10(2-3):147–180, 2000.

[8] Daniel Billsus and Michael J Pazzani. Adaptive news access. In *The adaptive web*, pages 550–570. Springer, 2007.

[9] Richard Boulton. Snowball website. `http://snowball.tartarus.org/`. Accessed: 2015-12-10.

[10] Mark Carman and Muhammad Ibrahim. Investigating performance and scalability for rank learning with regression tree ensembles, 2016.

[11] Francesco Corcoglioniti, Marco Rospocher, Roldano Cattoni, Bernardo Magnini, and Luciano Serafini. Interlinking unstructured and structured knowledge in an integrated framework. In *7th IEEE International Conference on Semantic Computing (ICSC), Irvine, CA, USA*, 2013.

[12] Alberto Díaz and Pablo Gervás. User-model based personalized summarization. *Information Processing & Management*, 43(6):1715–1734, 2007.

[13] Alberto Díaz, Pablo Gervás, and Antonio García. Evaluation of a system for personalized summarization of web contents. In *User Modeling 2005*, pages 453–462. Springer, 2005.

[14] Alberto Díaz, Pablo Gervás, Antonio García, and Laura Plaza. Development and use of an evaluation collection for personalisation of digital newspapers. In *LREC*, 2010.

[15] Fernando Diaz. Information retrieval: Ranking, 2011.

[16] Yajuan Duan, Long Jiang, Tao Qin, Ming Zhou, and Heung-Yeung Shum. An empirical study on learning to rank of tweets. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 295–303. Association for Computational Linguistics, 2010.

[17] Birgit Endrass, Matthias Rehm, and Elisabeth André. Planning small talk behavior with cultural influences for multiagent systems. *Computer Speech & Language*, 25(2):158–174, 2011.

[18] Susan Gauch, Mirco Speretta, Aravind Chandramouli, and Alessandro Micarelli. User profiles for personalized information access. In *The adaptive web*, pages 54–89. Springer, 2007.

[19] Michael Grobe. Rdf, jena, sparql and the'semantic web'. In *Proceedings of the 37th annual ACM SIGUCCS fall conference: communication and collaboration*, pages 131–138. ACM, 2009.

[20] Udo Hahn and Inderjeet Mani. The challenges of automatic summarization. *Computer*, 33(11):29–36, 2000.

[21] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[22] LI Hang. A short introduction to learning to rank. *IEICE TRANSACTIONS on Information and Systems*, 94(10):1854–1862, 2011.

[23] Haibo He, Edwardo Garcia, et al. Learning from imbalanced data. *Knowledge and Data Engineering, IEEE Transactions on*, 21(9):1263–1284, 2009.

[24] Lynette Hirschman and Robert Gaizauskas. Natural language question answering: the view from here. *natural language engineering*, 7(04):275–300, 2001.

[25] Katherine Isbister, Hideyuki Nakanishi, Toru Ishida, and Cliff Nass. Helper agent: Designing an assistant for human-human interaction in a virtual meeting space. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 57–64. ACM, 2000.

[26] K Sparck Jones et al. Automatic summarizing: factors and directions. *Advances in automatic text summarization*, pages 1–12, 1999.

[27] Florian Kaiser. Development of a domain-independent interactive question answering system. Bachelor's thesis, Karlsruhe Institute of Technology, 2015.

[28] Igor Kononenko, Edvard Šimec, and Marko Robnik-Šikonja. Overcoming the myopia of inductive learning algorithms with relieff. *Applied Intelligence*, 7(1):39–55, 1997.

[29] Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

[30] Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender systems handbook*, pages 73–105. Springer, 2011.

[31] Yuanhua Lv, Taesup Moon, Pranam Kolari, Zhaohui Zheng, Xuanhui Wang, and Yi Chang. Learning to model relatedness for news recommendation. In *Proceedings of the 20th international conference on World wide web*, pages 57–66. ACM, 2011.

[32] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014.

[33] Michael F McTear. Spoken dialogue technology: enabling the conversational user interface. *ACM Computing Surveys (CSUR)*, 34(1):90–169, 2002.

[34] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[35] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[36] Ananth Mohan, Zheng Chen, and Kilian Q Weinberger. Web-search ranking with initialized gradient boosted regression trees. In *Yahoo! Learning to Rank Challenge*, pages 77–89. Citeseer, 2011.

[37] Raymond J Mooney and Loriene Roy. Content-based book recommending using learning for text categorization. In *Proceedings of the fifth ACM conference on Digital libraries*, pages 195–204. ACM, 2000.

[38] Martha Palmer, Daniel Gildea, and Paul Kingsbury. The proposition bank: An annotated corpus of semantic roles. *Computational linguistics*, 31(1):71–106, 2005.

[39] Jeff Z Pan. Resource description framework. In *Handbook on Ontologies*, pages 71–90. Springer, 2009.

[40] Michael J Pazzani and Daniel Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007.

[41] Cassio Pennachin and Ben Goertzel. Contemporary approaches to artificial general intelligence. In *Artificial general intelligence*, pages 1–30. Springer, 2007.

[42] Byron Reeves and Clifford Nass. *How people treat computers, television, and new media like real people and places*. CSLI Publications and Cambridge university press, 1996.

[43] Nico Schlaefer, Petra Gieselmann, and Guido Sautter. The ephyra qa system at trec 2006. In *Proceedings of the Fifteenth Text REtrieval Conference*, 2006.

[44] Nico Schlaefer, Petra Gieselmann, Thomas Schaaf, and Alex Waibel. A pattern learning approach to question answering within the ephyra framework. In *Text, speech and dialogue*, pages 687–694. Springer, 2006.

[45] Nico Schlaefer, Jeongwoo Ko, Justin Betteridge, Manas A Pathak, Eric Nyberg, and Guido Sautter. Semantic extensions of the ephyra qa system for trec 2007. In *TREC*, 2007.

[46] Maria Schmidt and Jan Niehues. Natural language processing and dialog modeling – social spoken dialog systems (lecture slides), 2015.

[47] Maria Schmidt, Jan Niehues, and Alex Waibel. Towards an open-domain social dialog system. In *Proceedings of the Seventh International Workshop on Spoken Dialogue System*, 2016.

[48] Amit Singhal. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24(4):35–43, 2001.

[49] Gabriel Skantze. Error handling in spoken dialogue systems. *Computer Science and Communication Department of Speech, Music and Hearing*, 2007.

[50] Anastasios Tombros and Mark Sanderson. Advantages of query biased summaries in information retrieval. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 2–10. ACM, 1998.

[51] Koji Tsuda and Hiroto Saigo. Graph classification. In *Managing and mining graph data*, pages 337–363. Springer, 2010.

[52] Alan M Turing. Computing machinery and intelligence. *Mind*, pages 433–460, 1950.

[53] Marieke van Erp, Gleb Satyukov, Piek Vossen, and Marit Nijssen. Discovering and visualising stories in news. In *Proceedings of the 9th Language Resources and Evaluation Conference (LREC2014)*, Reykjavik, Iceland, May 26-31 2014.

[54] Willem Robert Van Hage, Véronique Malaisé, Roxane Segers, Laura Hollink, and Guus Schreiber. Design and use of the simple event model (sem). *Web Semantics: Science, Services and Agents on the World Wide Web*, 9(2):128–136, 2011.

[55] Piek Vossen, German Rigau, Luciano Serafini, Pim Stouten, Francis Irving, and Willem Robert Van Hage. Newsreader: recording history from daily news streams. In *Proceedings of the 9th Language Resources and Evaluation Conference (LREC2014)*, Reykjavik, Iceland, May 26-31 2014.

[56] World Wide Web Consortium (W3C). Sparql 1.1 query language. `http://www.w3.org/TR/2013/REC-sparql11-query-20130321/`. Accessed: 2015-10-13.

[57] The Wikimedia Foundation. Wikinews, the free news source. `https://en.wikinews.org/wiki/Main_Page`. Accessed: 2015-10-22.

[58] Koichiro Yoshino and Tatsuya Kawahara. Information navigation system based on pomdp that tracks user focus. In *15th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, page 32, 2014.

[59] Koichiro Yoshino and Tatsuya Kawahara. News navigation system based on proactive dialogue strategy. In *International Workshop series on Spoken Dialogue Systems technology*, 2015.

[60] Koichiro Yoshino, Shinsuke Mori, and Tatsuya Kawahara. Spoken dialogue system based on information extraction using similarity of predicate argument structures. In *Proceedings of the SIGDIAL 2011 Conference*, pages 59–66. Association for Computational Linguistics, 2011.

A. Appendix: Implementation Details

This appendix gives more details about the implementation of the NewsTeller system that were not mentioned in Chapter 3. The source code of the NewsTeller system is available online under `https://github.com/ukcvo/NewsTeller`.

A.1. External Resources

This section gives an overview over the external resources that were used for the implementation of the NewsTeller system.

We used Apache Maven (see `https://maven.apache.org`) as dependency management tool to include libraries into our project. Table A.1 shows all top-level external libraries used in the implementation of the NewsTeller system. We only list the "top level" of libraries, i.e., libraries whose functionality is directly used by the NewsTeller system. Libraries that are only indirectly used because they are a dependency of any of the "top level" libraries are not listed. All libraries are given with their Maven definition (group ID, artifact ID, and version) as well as a short description of their purpose (i.e., why they are needed in the NewsTeller system).

In addition to these Java libraries, the following resources were used:

- **Propbank frames**: We used propbank frames from `https://propbank.github.io` (accessed on Nov 23 2015). The frames contain the different word senses as well as a list of arguments for each word sense. This information was compared to the arguments present in the RDF-based event representation to define features for the event filtering step.
- **WordNet**: As already shown in Table A.1, we used the WordNet [35] 3.1 data set. We included it as Maven dependency but the original resources are also available at `https://wordnet.princeton.edu/wordnet/`. The WordNet synsets along with their part-of-speech annotation were used to define a feature for the filtering classifier.
- **Word embeddings**: We used pre-trained word embeddings trained on the GoogleNews corpus which are available online at `https://code.google.com/archive/p/word2vec/` to define various features for the event ranking step.
- **Stop words**: We use a list of stop words obtained from `http://anoncvs.postgresql.org/cvsweb.cgi/pgsql/src/backend/snowball/stopwords/` for all features based on word embeddings to filter out frequently occurring function words like "a", "the", or "because".

Group	Artifact	Version	Purpose
org.springframework	spring-beans	4.2.2.RELEASE	Dependency injection
org.springframework	spring-context	4.2.2.RELEASE	Dependency injection
org.springframework	spring-core	4.2.2.RELEASE	Dependency injection
org.springframework	spring-expression	4.2.2.RELEASE	Dependency injection
junit	junit	4.12	Unit tests
com.github.rholder	snowball-stemmer	1.3.0.581.1	Keyword stemming
org.jumpmind.symmetric	symmetric-csv	3.5.19	CSV I/O for data sets
net.sf.extjwnl	extjwnl	1.9.1	WordNet access
net.sf.extjwnl	extjwnl-data-wn31	1.2	WordNet 3.1 data set
nz.ac.waikato.cms.weka	weka-dev	3.7.13	Machine learning
eu.fbk.knowledgestore	ks-client	1.5.1	KnowledgeStore access
org.slf4j	jcl-over-slf4j	1.7.12	Logging
org.slf4j	slf4j-jdk14	1.7.12	Logging
edu.stanford.nlp	stanford-corenlp	3.6.0	Sentence Extraction & Tokenization
org.deeplearning4j	deeplearning4j-core	0.4-rc3.8	Word Embeddings

Table A.1.: Table showing all libraries used for implementing the NewsTeller system.

A.2. Systematic Input/Output Breakdown

NewsTeller	
Input	List of user query keywords, user model (list of user interest keywords and list of previously selected events)
Output	Sentence about a relevant news event
Task	Find an event relevant to the user query keywords and the user interest keywords while avoiding events that have been selected in previous turns. Output a sentence about the selected event.
Processing Steps	1.) Retrieve the most relevant event (*Event Retrieval*) 2.) Create a summary sentence about the selected event (*Summary Creation*)

1.) Event Retrieval	
Input	List of user query keywords, user model (list of user interest keywords and list of previously selected events)
Output	Most relevant news event
Task	Personalized Information Retrieval of a news event based on the input
Processing Steps	1.1.) Search for potentially relevant events (*Event Search*) 1.2.) Filter events according to their well-formedness (*Event Filtering*) 1.3.) Rank events based on their expected relevance (*Event Ranking*) 1.4.) Select the most relevant event (*Event Selection*)

1.1.) Event Search	
Input	List of user query keywords
Output	Set of potentially relevant news events
Task	Find events potentially relevant to the user query
Processing Steps	1.1.1.) Create a SPARQL query based on the user query. 1.1.2.) Send the SPARQL query to the KnowledgeStore and collect the results.
	1.1.3.) If the SPARQL query returned more than 1,000 results: randomly select 1,000 events

1.2.) Event Filtering	
Input	Set of potentially relevant news events, list of user query keywords
Output	Set of filtered news events
Task	Filter the news events according to their well-formedness.

Processing Steps	1.2.1.) Extract features for each event, based on the KnowledgeStore content and the user query 1.2.2.) Classify each event as USABLE or NOT USABLE based on its feature values 1.2.3.) Remove all events that were classified as NOT USABLE
1.3.) Event Ranking	
Input	Set of filtered news events, list of user query keywords, user model (list of user interest keywords and list of previously selected events)
Output	List of news events sorted in descending order according to their estimated relevance
Task	Rank the given news events according to their expected relevance to both the user query and the user's general interests.
Processing Steps	1.3.1.) Extract features for each event, based on the KnowledgeStore content, the user query, and the user's interests 1.3.2.) Use a regressor to estimate each event's relevance value based on its feature values 1.3.3.) Sort the events in descending order according to their estimated relevance values.
1.4.) Event Selection	
Input	List of news events sorted in descending order based on the events' estimated relevance values, user model (list of user interest keywords and list of previously selected events), threshold
Output	Most relevant news event
Task	Pick the most relevant news event that has not been talked about, yet
Processing Steps	1.4.1.) Look at the top element of the list as candidate event 1.4.2.) If the candidate event has already been selected in previous turns: look at the next event in the list and repeat this step 1.4.3.) If the expected relevance value of the candidate event is lower than the threshold, return `null`, otherwise return the candidate event
2.) Summary Creation	
Input	Most relevant news event, user model (list of user interest keywords and list of previously selected events)
Output	Sentence about most relevant news event
Task	Create a sentence about the given event and add the given event to the list of previously selected events in the user model

Processing Steps	2.1.) Extract one of the sentences in which the event was mentioned 2.2.) Add the event along with its summary to the list of previously selected events in the user model

Table A.2.: Table showing the different components of the NewsTeller system along with their respective input, output, task, and processing steps.

A.3. Filtering

This section contains more information about the event filtering step.

A.3.1. Data Set

Table A.3 shows all queries used for creating the filtering data set along with the number of events retrieved for them and the number of Usable events among these.

A.3.2. Features

The features that were used for the filtering problem are the following:

- **a1**: Counts the number of `propbank:A1` links.
- **actorPositionLeft**: Retrieves the labels of all actors, calculates the fraction of actors appearing before the event label in the original sentence.
- **actorPositionRight**: Retrieves the labels of all actors, calculates the fraction of actors appearing before the event label in the original sentence.
- **appearKeywordLabelPartsInText**: Retrieves the DBpedia labels of all entities, i.e., actors and places, that match the keyword. Uses inheritance (along `rdf:type` links) if no DBpedia label is available at the instance level. Splits the labels into their parts and calculates the for each label the fraction of tokens that appear in the overall text. Aggregates these fractions per entity by using the maximum and across entities by using the average.
- **appearKeywordLabelsInSentence**: Same as appearKeywordLabelPartsInTextFeature, but matching the whole label against the sentence instead of matching label-parts against the whole text.
- **appearKeywordDescriptionPartsInText**: Same as appearKeywordLabelPartsInTextFeature, but looking at the description-parts appearing in the complete text instead of the label parts apperaing in the sentence.
- **appearLabelsInText**: Same as appearKeywordLabelsInSentenceFeature, but looking at all entities (not just the ones matching the keyword) and matching their labels against the complete text (not just the sentence).

Query	#events	#Usable	%Usable
artificial intelligence	93	6	6.45%
bankruptcy	106	16	15.09%
Berlin	198	35	17.68%
champions league	228	16	7.02%
chancellor	203	34	16.75%
charity	25	5	20.00%
Chernobyl	346	6	1.73%
cinema	101	14	13.86%
comedy	402	45	11.19%
concert	234	44	18.80%
contradict	65	22	33.85%
Edinburgh	120	22	18.33%
Edmund Hillary	10	3	30.00%
erupt	211	120	56.87%
European Space Agency	59	11	18.64%
Facebook	315	31	9.84%
Fukushima	56	8	14.29%
GermanWings	5	2	40.00%
Hawking	165	3	1.82%
Himalaya	16	3	18.75%
hurricane Katrina	153	32	20.92%
IBM	148	22	14.86%
Iceland	218	42	19.27%
kiss	75	5	6.67%
Lehman Brothers	46	7	15.22%
manipulate	127	24	18.90%
marathon	70	10	14.29%
Medvedev	80	18	22.50%
Merkel	74	18	24.32%
Michael Jackson	250	15	6.00%
Mount Everest	22	2	9.09%
museum	315	61	19.37%
pope Francis	20	2	10.00%
power station	105	24	22.86%
Rafael Nadal	52	13	25.00%
Real Madrid	59	9	15.25%
Rhine	15	1	6.67%
riot	374	88	23.53%
Roger Federer	50	14	28.00%
Rome	199	9	4.52%
Sahara	54	5	9.26%
Star Wars	81	9	11.11%
Tom Cruise	112	19	16.96%
upload	75	16	21.33%
volcano	291	66	22.68%
Watson	71	3	4.23%

Table A.3.: Table showing all queries used for creating the filtering data set.

- **appearSplitDescriptionsInText**: Same as appearKeywordDescriptionPartsInTextFeature, but looking at all entities and not only the ones matching the keyword stem.
- **appearSplitLabelsInText**: Same as appearKeywordLabelPartsInTextFeature, but looking at all entities and not only the ones matching the keyword stem.
- **hasDBpediaEntities**: Counts the number of entities that have a DBpedia label matching the given keyword.
- **keywordEntityMatchingKeyword**: Retrieves all entities matching the given keyword and counts how many of their labels contain the original keyword (i.e., not its stemmed form, but the complete word).
- **keywordInTextContains**: Checks if the given keyword in its original form (i.e., not stemmed) in contained in the original news article.
- **locationPrepBeforeActor**: Counts the fraction of actors that are preceded by a location preposition in the original news article.
- **maxConstituentSeparatedByEvent**: Counts the maximum number of other events that lie between any two constituents of the given event.
- **maxEntitiesPerMention**: Counts for each mention the number of event entities appearing in the given mention's sentence and takes the maximum across all mentions.
- **minWordDistance**: Counts the minimum amount of words between any two constituents of the given event.
- **needsA2**: Retrieves the PropBank roleset associated with the event mention and checks for this roleset if an A2 argument is needed.
- **nonzeroEntitiesPerMention**: Determines how many mentions of the given event have more than zero event entities in their sentence.
- **numberOfMentions**: Counts the number of mentions for the given event.
- **overlap**: Checks the constituents of the given events for overlaps.
- **pos**: Checks if the event's mentions have the part-of-speech tag "verb".
- **prepPhrase**: Counts the fraction of actors whose label starts with a preposition.
- **prepPhraseLocation**: Counts the fraction of actors whose label starts with a location preposition.
- **propbankArgument**: Compares the arguments existing in the RDF structure of the event to the arguments expected based on the propbank roleset. This roleset is retrieved from the respective event mention. Takes the maximum fraction of satisfied arguments across all matching rolesets and all mentions.

- **propbankAvgArgument**: Same as propbankArgumentFeature, but using the average instead of the maximum.

- **smartPropbankFallbackAvgArgument**: Same as propbankAvgArgumentFeature but automatically choosing whether to use the PropBank or the NomBank rolesets based on the part-of-speech information of the respective mention, and using the other roleset type as a fallback solution.

- **wordnet**: Retrieves all WordNet synsets which contain the event label and computes the fraction of synsets describing verbs.

- **wordnetRef**: Counts the number of WordNet references stored in the event's mentions.

Table A.4 indicates for each feature the layers of the KnowledgeStore that were used as well as whether the keyword was taken into account.

The global random forest classifier uses all of the above features. The specialized classifiers for the subclasses of Not Usable that were used in the ensemble-based approaches each use only a subset:

- No Event: pos, wordnet, wordnetRef

- Keyword Entity Categorization: appearKeywordLabelPartsInText, appearKeywordLabelsInSentence, appearKeywordDescriptionPartsInText

- Missing Object: a1, needsA2, POS, prepPhrase, smartPropbankFallbackAvgArgument

- Overlapping Constituents: overlap

- Event Merge: nonzeroEntitiesPerMention, numberOfMentions

- Missing Subject: actorPositionLeft, actorPositionRight, POS, propbankArgument

- Broken Entity: locationPrepBeforeActor, prepPhraseLocation

- Wrong Parse: locationPrepBeforeActor, maxConstituentSeparatedByEvent, maxEntitiesPerMention, minWordDistance, prepPhrase, propbankAvgArgument

- Other Entity Categorization: appearLabelsInText, appearSplitDescriptionsInText, appearSplitLabelsInText

- Keyword Regex Mismatch: hasDBpediaEntities, keywordEntityMatchingKeyword, keywordInTextContains

Feature	R	M	E	K	X
a1			X		
actorPositionLeft	X		X		
actorPositionRight	X		X		
appearKeywordLabelPartsInText	X		X	X	
appearKeywordLabelsInSentence	X		X	X	
appearKeywordDescriptionPartsInText	X		X	X	
appearLabelsInText	X		X		
appearSplitDescriptionsInText	X		X		
appearSplitLabelsInText	X		X		
hasDBpediaEntities			X	X	
keywordEntityMatchingKeyword			X	X	
keywordInTextContains	X				
locationPrepBeforeActor	X		X		
maxConstituentSeparatedByEvent	X	X	X		
maxEntitiesPerMention	X	X	X		
minWordDistance	X	X	X		
needsA2		X			X
nonzeroEntitiesPerMention	X		X		
numberOfMentions			X		
overlap		X	X		
pos		X			
prepPhrase			X		
prepPhraseLocation			X		
propbankArgument		X	X		X
propbankAvgArgument		X	X		X
smartPropbankFallbackAvgArgument		X	X		X
wordnet			X		X
wordnetRef		X			

Table A.4.: Table showing for each features which information it uses: the **R**esource layer, **M**ention layer, and **E**ntity layer of the KnowledgeStore, as well as the **K**eywords and e**X**ternal sources.

A.3.3. Feature Selection

During feature selection, we employed different feature selection algorithms. Each one of them was asked for the top k features and the features being mentioned most frequently across all feature selection algorithms were used as candidates for a further wrapper-based optimization. The feature selection algorithms being used were the following (all of them being part of the WEKA ML framework):

- **RELIEF-F**: Estimates the usefulness of a feature based on repeatedly sampling an instance and comparing the distance with respect to the given feature for the sampled instance to the closest instance of the same class and the closest instance of another class. Sorts the features according to their estimated usefulness. The algorithm itself is described in [28].

- **oneR**: Estimates the usefulness of a feature based on its classification performance when used as the only feature. Sorts the features according to their estimated usefulness.

- **gainRatio**: Estimates the usefulness of a feature based on its gain ratio with respect to the class label. This gain ratio is defined based on information entropy: $GainRatio(class, feature) = (H(class) - H(class|feature))\ /H(feature)$ Sorts the features according to their estimated usefulness.

- **CFS**: Uses a greedy incremental search among feature sets. For each feature set under consideration, the predictive ability of each feature and the redundancy between features are taken into account, looking for a predictive feature set without redundancy.

- **Wrapper**: Uses a greedy incremental search among feature sets. Each feature set is used to train a classifier and the classifier performance obtained in cross-validation is used as indicator of feature set quality.

Feature selection was performed for each subclass of Not Usable individually. For the wrapper feature selection algorithm, we always used a decision tree and a naive Bayes classifier for all subclasses, plus for each subclass the classifiers that had performed best in preliminary experiments on the overall set of all features under consideration.

The classifier used for a further wrapper-based reduction of the feature set size was the classifier showing best performance on the candidate feature set obtained by taking into account the results of all feature selection algorithms.

A.4. Ranking

This section gives more information about the event ranking step.

A.4.1. Data Sets

The first data set being used in the ranking problem was the so called "bootstrap data set". It was obtained by labeling the events from the filtering data set that were left after applying the filtering classifier in a ten-fold cross-validation. In addition to the single-keyword queries shown in Table A.5, some of these single-keyword queries were combined into queries containing two or three keywords (see Table A.6).

In addition to the objectively labeled bootstrap data set, another data set was created which takes into account also personal general interests. This user model data set ("UM data set") was obtained by asking eleven annotators for their general interests and some queries as well as to label some events for each of these queries.

Figures A.1 and A.2 show the questionnaire used to elicit interests and queries, and the instructions for labeling the events, respectively. Note that the descriptions were very short and abstract in order to not bias the annotators too much.

Table A.7 gives an overview over the interests and queries of the different annotators. Table A.8 shows the overall ratings for each user.

A.4.2. Features

The following list includes all features that were used by one ore more of the regressors discussed in Section 3.5:

- **BM25Sentence1.2**: Computes the BM25 score for the user query and the given event with respect to the underlying sentence. Uses a parameter of $k = 1.2$ for the BM25 formula.
- **BM25Sentence1.4true**: Computes the BM25 score for the user interests and the given event with respect to the underlying sentence. Uses a parameter of $k = 1.4$ for the BM25 formula.
- **BM25Sentence2.0**: Computes the BM25 score for the user query and the given event with respect to the underlying sentence. Uses a parameter of $k = 2.0$ for the BM25 formula.
- **BM25Title1.6true**: Computes the BM25 score for the user interests and the given event with respect to the title of the underlying news article. Uses a parameter of $k = 1.6$ for the BM25 formula.
- **entityContains_it_ep**: Counts how many event entities contain a form of "it" (e.g., "it", "itself"). Looks at all entities and their respective `skos:prefLabel`.
- **entityEmbeddings02ed**: Compares the embeddings of the user query to the embeddings of the event entities by using cosine similarity. Uses the DBpedia description of all entities. Takes the maximum similarity across entities and the average across all query keywords.

Query	#events	Irrelevant	Partially Relevant	Relevant	Very Relevant
artificial intelligence	4	1	2	1	0
bankruptcy	20	7	7	4	2
Berlin	26	13	8	4	1
champions league	19	6	7	4	2
chancellor	38	12	15	8	3
charity	4	2	1	1	0
Chernobyl	3	1	1	0	1
cinema	9	2	5	2	0
comedy	31	15	10	6	0
concert	26	16	8	2	0
contradict	29	16	5	7	1
Edinburgh	16	5	3	8	0
Edmund Hillary	3	0	0	3	0
erupt	137	81	21	26	9
European Space Agency	4	1	2	0	1
Facebook	29	13	13	2	1
Fukushima	3	1	0	1	1
GermanWings	2	1	0	0	1
Hawking	9	6	1	2	0
hurricane Katrina	26	9	6	11	0
IBM	18	10	4	3	1
Iceland	33	12	8	8	5
kiss	8	5	3	0	0
Lehman Brothers	8	1	2	4	1
manipulate	26	10	9	5	2
marathon	4	2	1	0	1
Medvedev	18	7	3	4	4
Merkel	20	9	5	4	2
Michael Jackson	8	3	1	3	1
Mount Everest	2	1	0	1	0
museum	44	24	13	7	0
pope Francis	5	3	1	0	1
power station	10	2	2	5	1
Rafael Nadal	13	5	4	3	1
Real Madrid	11	5	4	1	1
riot	92	57	20	14	1
Roger Federer	15	2	4	8	1
Rome	7	2	3	1	1
Sahara	7	3	2	2	0
Star Wars	5	1	1	1	2
Tom Cruise	12	3	7	2	0
upload	29	16	7	5	1
volcano	82	37	21	19	5
Watson	11	8	2	1	0

Table A.5.: Table showing all single-keyword queries from the bootstrap data set.

Query	#events	Irrelevant	Partially Relevant	Relevant	Very Relevant
Two keywords					
artificial intelligence, bankruptcy	24	16	8	0	0
bankruptcy, hurricane Katrina	46	18	25	2	1
bankruptcy, Iceland	53	23	25	3	2
Berlin, champions league	45	26	18	1	0
Berlin, marathon	29	23	5	0	1
chancellor, bankruptcy	58	43	9	6	0
chancellor, Merkel	39	21	7	8	3
concert, Edinburgh	42	36	2	4	0
erupt, volcano	191	144	29	14	4
Fukushima, Chernobyl	6	3	1	2	0
GermanWings, Edinburgh	18	16	2	0	0
Lehman Brothers, bankruptcy	26	18	4	3	1
Merkel, Berlin	46	28	18	0	0
Merkel, Lehman Brothers	28	25	3	0	0
pope Francis, Tom Cruise	17	14	3	0	0
Rafael Nadal, Roger Federer	26	12	9	4	1
Real Madrid, champions league	30	23	6	0	1
riot, erupt	227	177	33	16	1
Three keywords					
chancellor, Merkel, Berlin	65	45	16	4	0
Iceland, erupt, volcano	222	205	11	1	5

Table A.6.: Table showing all multiple-keyword queries from the bootstrap data set.

ID	Interests	Queries
usr_01	tennis RHCP exchange rates Putin Germany information technology	exchange rates, Russian ruble to euro Moscow, latest events northern Germany, refugees RHCP, next concert, Germany (*) tennis, courts, winter tennis, trainer, prices information technology, latest news Putin, Merkel Putin, Syria Apple, computer novelties Frankfurt am Main, Moscow, cheap flights northern Germany, places to visit (*)
usr_02	football video games cars movies Internet Breaking Bad	Bundesliga, results USA, election Donald Trump, daughter jogging, shoes, test Dark Souls 3, review, PC Mario Kart, online Formula 1, car presentations Formula 1, Hockenheim, date Christopher Nolan, new movie (*) Oscars, nominations Better Call Saul, first episode (*) Lost, ending, explanation
usr_03	graphic design user experience design football new technologies politics	graphic design, typography (*) graphic design, posters apps, user experience design new technologies, Silicon Valley (*) football, Champions League, Bundesliga football, transfer Apple, Google, Microsoft startup, Kickstarter Germany, politics, Internet politics, satire design, books Internet, blogs, design graphic design, corporate design, logos Internet, lifestyle

ID	Interests	Queries
usr_04	cricket adventure Italy Big Bang Theory Friends TV show Albert Einstein	world cup terrorist, attack Barack Obama football, match deep learning (*) Christopher Nolan (*) best, cricketer Jobs Nobel, prize political parties wiki, news, leak 4g, network
usr_05	politics science culture women travel animals	gravitational waves (*) philharmony, concert, Trento Venice, carnival Turkey, bombs, Syria elephants, Asia MIT, Media Lab refugees, Europe Italy, civil union Elon Musk (*) elections, US gender, salary, inequality aboriginal, Australians
usr_06	machine learning computer science Eurozone Schengen visa Italian recipes polititcs	transfer learning (*) deep learning (*) Brazilian, economy Italian, elections American, elections Premier League word embeddings (*) supervised learning (*) quantum computing (*) tourism, Brazil tourism, Europe middle east, crisis

ID	Interests	Queries
usr_07	travel politics movies business food psychology	elections, eastern, Europe international, mergers largest, airport, worldwide new, movie, hits DAX, development, 2015 airplane, crash, Europe UFO, sightings best, restaurant, worldwide food, trends, 2015 Donald Trump Nobel Peace Prize, winner, 2014 gold, price, development
usr_08	politics social media cats law VIPs Game of Thrones	Jon Snow, Khaleesi, season 6 (*) books, George R. R. Martin, a song of ice and fire jurisdiction, current, verdicts Instagram, Facebook, Twitter fashion, trends interior design, decoration literature German politics, international politics, European politics (*) funny cat videos, cat pictures (*) traveling, Barcelona, Mallorca food, cooking, restaurant sports, swimming, skiing
usr_09	US politics Super Bowl planes soccer music	Clinton, Sanders lottery, money car, race track penalty, win success, statistics
usr_10	cars sports politics stock market research advancements travel	F1, 2010, drivers cricket, world cup, 2011 electric cars (*) travel, Cambodia Arab, revolution Black Friday (*) Lehman Brothers world economy MIT, Media Lab Olympics, London Sochi (*) Modi

ID	Interests	Queries
usr_11	Iran	Islam, Christianity
	religion	Iran, nuclear deal
	politics	Iran, foreign policies
	Italy	Italy, touristic attractions
	touristic attractions	Venice, history
		Iran, congress, election

Table A.7.: Table showing the interests and queries from all annotators. Queries marked with an asterisk did not return any events and did therefore not appear in the data set.

ID	#queries	Irrelevant	Partially Relevant	Relevant	Very Relevant
usr_01	10	51.02%	34.69%	7.14%	7.14%
usr_02	10	67.00%	26.00%	3.00%	4.00%
usr_03	12	42.00%	22.00%	25.00%	11.00%
usr_04	10	23.00%	44.00%	24.00%	9.00%
usr_05	10	25.00%	60.00%	8.00%	7.00%
usr_06	7	7.07%	55.56%	17.17%	20.20%
usr_07	12	53.00%	31.00%	5.00%	11.00%
usr_08	9	45.00%	20.00%	21.00%	14.00%
usr_09	5	29.00%	45.00%	12.00%	14.00%
usr_10	10	52.00%	24.00%	13.00%	11.00%
usr_11	6	64.65%	13.13%	7.07%	15.15%

Table A.8.: Table showing information about the data set aggregated for each annotator.

- **entityEmbeddings02edtrue**: Compares the embeddings of the user interests to the embeddings of the event entities by using cosine similarity. Uses the DBpedia description of all entities. Takes the maximum similarity across entities and the average across all interest keywords.
- **entityEmbeddings11ed**: Compares the embeddings of the user query to the embeddings of the event entities by using cosine similarity. Uses the DBpedia description of all entities. Takes the minimum similarity across entities and the minimum across all query keywords.
- **entityEmbeddings11kp**: Compares the embeddings of the user query to the embeddings of the event entities by using cosine similarity. Uses the `skos:prefLabel` of the entities matching any query keyword. Takes the minimum similarity across entities and the minimum across all query keywords.
- **entityEmbeddings12em**: Compares the embeddings of the user query to the embeddings of the event entities by using cosine similarity. Uses all entities and looks

only at labels matching any query keyword. Takes the maximum similarity across entities and the minimum across all query keywords.

- **entityEmbeddings12eptrue**: Compares the embeddings of the user interests to the embeddings of the event entities by using cosine similarity. Uses the `skos:prefLabel` of all entities. Takes the maximum similarity across entities and the minimum across all interest keywords.
- **entityEmbeddings32kl**: Compares the embeddings of the user query to the embeddings of the event entities by using cosine similarity. Uses the dbpedia label of the entities matching any query keyword. Takes the maximum similarity across entities and the geometric mean across all query keywords.
- **interestQueryEmbeddings01**: Compares the embeddings of the user query to the embeddings of the user interests by using cosine similarity. Takes the minimum across user query keywords and the average across user interest keywords.
- **keywordComparisonGeom**: Compares the embeddings of the user query keywords amongst each other using cosine similarity. Aggregates the similarities using the geometric mean.
- **keywordComparisonMaxtrue**: Compares the embeddings of the user interest keywords amongst each other using cosine similarity. Aggregates the similarities using the maximum.
- **keywordComparisonMin**: Compares the embeddings of the user query keywords amongst each other using cosine similarity. Aggregates the similarities using the minimum.
- **keywordComparisonMintrue**: Compares the embeddings of the user interest keywords amongst each other using cosine similarity. Aggregates the similarities using the minimum.
- **keywordEntities1ffd**: Determines the fraction of event entities whose description contains the query keyword in its original form. Aggregates across all the user query keywords using the minimum.
- **keywordInSentence11ftft**: Splits each query keyword into its tokens, then checks for each keyword-sentence combination how many keyword tokens appear in the sentence (based on the sentences in which the event is mentioned). Aggregates across keywords and across sentences using the minimum.
- **keywordInSentence11ttft**: Splits each query keyword into its tokens, then checks for each keyword-title combination how many keyword tokens appear in the title of the article in which the event has been mentioned. Aggregates across keywords and across sentences using the minimum.
- **numberOfDummyEntities**: Counts the number of entities that are not part of the DBpedia subgraph and that have no outgoing `rdf:type` links.

- **numberOfKeywordEntitiesAvgtrue**: Counts for each interest keyword the number of entities matching this keyword. Aggregates across keywords by computing the average.
- **numberOfNonEntities**: Counts the number of entities that are not part of the DBpedia subgraph.
- **recencyMonths**: Determines the age (in months) of the news article, in which the given event was mentioned.
- **sameDocument1tt**: Looks at all the sentences in which the given event has been mentioned and counts how many of the other events to be ranked appear in the same sentence. Aggregates over the sentences using the minimum function and normalizes the result with respect to the total number of sentences for all events under consideration.
- **sameDocument2tt**: Looks at all the sentences in which the given event has been mentioned and counts how many of the other events to be ranked appear in the same sentence. Aggregates over the sentences using the maximum function and normalizes the result with respect to the total number of sentences for all events under consideration.
- **sentenceLengthCharMin**: Measures the sentence length in characters for each of the sentences in which the event is mentioned and aggregates them by taking the minimum.
- **textEmbeddings11f**: Compares the word embedding of each query keyword to the embedding of each sentence in which the event was mentioned. This sentence vector is obtained by summing the word vectors of the words appearing in the sentence. The comparison is done by using cosine similarity. Aggregates across sentences and across keywords by taking the minimum.
- **textEmbeddings13f**: Compares the word embedding of each query keyword to the embedding of each sentence in which the event was mentioned. This sentence vector is obtained by summing the word vectors of the words appearing in the sentence. The comparison is done by using cosine similarity. Aggregates across sentences using the geometric mean, and across keywords by taking the minimum.
- **textEmbeddings31t**: Compares the word embedding of each query keyword to the embedding of each title of the articles in which the event was mentioned. The vector for the title is obtained by summing the word vectors of the words appearing in the title. The comparison is done by using cosine similarity. Aggregates across titles by taking the minimum and across keywords by computing the geometric mean.

Table A.9 classifies these features with respect to different aspects.

Feature	RL	EL	E	Q	I
BM25Sentence1.2	X		X	X	
BM25Sentence1.4true	X		X		X
BM25Sentence2.0	X		X	X	
BM25Title1.6true	X		X		X
entityContains_it_ep		X	X		
entityEmbeddings02ed		X	X	X	
entityEmbeddings02edtrue		X	X		X
entityEmbeddings11ed		X	X	X	
entityEmbeddings11kp		X	X	X	
entityEmbeddings12em		X	X	X	
entityEmbeddings12eptrue		X	X		X
entityEmbeddings32kl		X	X	X	
interestQueryEmbeddings01				X	X
keywordComparisonGeom				X	
keywordComparisonMaxtrue					X
keywordComparisonMin				X	
keywordComparisonMintrue					X
keywordEntities1ffd		X	X	X	
keywordInSentence11ftft	X		X	X	
keywordInSentence11ttft	X		X	X	
numberOfDummyEntities		X	X		
numberOfKeywordEntitiesAvgtrue		X	X		X
numberOfNonEntities		X	X		
recencyMonths	X		X		
sameDocument1tt	X		X		
sameDocument2tt	X		X		
sentenceLengthCharMin	X		X		
textEmbeddings11f	X		X	X	
textEmbeddings13f	X		X	X	
textEmbeddings31t	X		X	X	

Table A.9.: Table categorizing each features with respect to the KnowledgeStore layer (**R**esource **L**ayer and **E**ntity Layer) and with respect to the information sources used (**E**vent, **Q**uery, **I**nterests).

The “no-UM baseline” regressor uses the following feature set:

- textEmbeddingsFeature11f
- textEmbeddingsFeature13f
- entityEmbeddingsFeature02ed
- entityEmbeddingsFeature11ed
- entityEmbeddingsFeature12em
- sentenceLengthFeatureCharMin
- numberOfNonEntitiesFeature
- numberOfDummyEntitiesFeature
- recencyFeatureMonths
- keywordEntitiesFeature1ffd
- keywordInSentenceFeature11ftft
- keywordInSentenceFeature11ttft
- BM25FeatureSentence1.2

The “no-UM addRemove” regressor uses the following feature set:

- textEmbeddingsFeature13f
- entityEmbeddingsFeature11ed
- entityEmbeddingsFeature12em
- sentenceLengthFeatureCharMin
- numberOfNonEntitiesFeature
- numberOfDummyEntitiesFeature
- keywordEntitiesFeature1ffd
- keywordInSentenceFeature11ftft
- keywordInSentenceFeature11ttft
- BM25FeatureSentence1.2
- textEmbeddingsFeature31t
- entityContainsFeature_it_ep

- sameDocumentFeature1tt

The “UM addRemove” regressor uses the following feature set:

- textEmbeddingsFeature13f
- entityEmbeddingsFeature02ed
- entityEmbeddingsFeature12em
- sentenceLengthFeatureCharMin
- numberOfNonEntitiesFeature
- recencyFeatureMonths
- keywordEntitiesFeature1ffd
- keywordInSentenceFeature11ftft
- BM25FeatureSentence1.2
- BM25FeatureSentence1.4true
- entityEmbeddingsFeature02edtrue
- numberOfKeywordEntitiesFeatureAvgtrue
- keywordComparisonFeatureMintrue

The “UM fromScratch” regressor uses the following feature set:

- textEmbeddingsFeature11f
- textEmbeddingsFeature13f
- textEmbeddingsFeature31t
- entityEmbeddingsFeature11kp
- entityEmbeddingsFeature12em
- entityEmbeddingsFeature32kl
- keywordInSentenceFeature11ftft
- BM25FeatureSentence2.0
- keywordComparisonFeatureMin
- keywordComparisonFeatureGeom
- sameDocumentFeature2tt

- BM25FeatureTitle1.6true
- entityEmbeddingsFeature12eptrue
- interestQueryEmbeddingsFeature01
- keywordComparisonFeatureMintrue

The "UM addRemove v2" regressor uses the following feature set:

- textEmbeddingsFeature13f
- entityEmbeddingsFeature11ed
- numberOfNonEntitiesFeature
- numberOfDummyEntitiesFeature
- keywordInSentenceFeature11ftft
- keywordInSentenceFeature11ttft
- BM25FeatureSentence1.2
- textEmbeddingsFeature31t
- constituentContainsFeature_it_ep
- sameDocumentFeature1tt
- entityEmbeddingsFeature02edtrue
- numberOfKeywordEntitiesFeatureAvgtrue
- keywordComparisonFeatureMaxtrue

A.4.3. Feature Selection

We used the following feature selection algorithms to determine a candidate feature set for the random forest regressor used to predict an event's relevance value:

- **RELIEF-F**: Estimates the usefulness of a feature based on repeatedly sampling an instance and comparing the distance with respect to the given feature for the sampled instance to the closest instance of the same class and the closest instance of another class. Sorts the features according to their estimated usefulness. The algorithm itself is described in [28].
- **CFS**: Uses a greedy incremental search among feature sets. For each feature set under consideration, the predictive ability of each feature and the redundancy between features are taken into account, looking for a predictive feature set without redundancy.

Regressor	AV	$P_{>0}$	$P_{>1}$
Average baseline	0.8812	0.5842	0.2079
Single feature baseline	1.0891	0.6436	0.2970
no-UM baseline	1.2851	0.7030	0.3634
no-UM addRemove	1.4386	0.7663	0.4237
UM addRemove	1.4455	0.7584	0.4624
UM fromScratch	1.4475	0.7634	0.4406
UM addRemove v2	1.5525	0.7723	0.4842

Table A.10.: Table showing the average performance of different regressors (with and without user model based features) on the UM data set in a user-based leave-one-out evaluation for the absolute versions of the metrics.

- **Correlation**: Uses the correlation between the given feature and the class label to estimate the given feature's usefulness. Sorts the features according to their estimated usefulness.

- **oneR**: Estimates the usefulness of a feature based on its classification performance when used as the only feature. Sorts the features according to their estimated usefulness.

- **gainRatio**: Estimates the usefulness of a feature based on its gain ratio with respect to the class label. This gain ratio is defined based on information entropy: $GainRatio(class, feature) = (H(class) - H(class|feature))\,/H(feature)$ Sorts the features according to their estimated usefulness.

- **Wrapper**: Uses a greedy incremental search among feature sets. Each feature set is used to train a regressor/classifier and its performance in cross-validation is used as indicator of feature set quality. For the regression feature selection, we used two classifiers on the discretized class labels (naïve Bayes and decision tree) as well as a random forest regressor on the numeric class label.

Note that some of the feature selection algorithms listed above only work with discrete classes (at least in their Weka implementation). This is the case for the Correlation, oneR, gainRatio and the classifier-based Wrapper approaches. In those cases, the numeric relevance label was discretized before applying the respective feature selection algorithm.

After having selected a candidate feature set, this feature set was further reduced by using a wrapper-based approach using a random forest regressor.

The feature selection for the final UM-based regressor was performed using the "add, then remove" procedure described in Section 3.5.2.3. In addition to this, the feature selection methodology described above was used to find a competitor feature set.

Regressor	**AV**	$P_{>0}$	$P_{>1}$
Average baseline	0.8812	0.5842	0.2079
Single feature baseline	1.0891	0.6436	0.2970
no-UM baseline	1.2238	0.6723	0.3505
no-UM addRemove	1.3792	0.7545	0.3931
UM addRemove	1.3683	0.7406	0.4158
UM fromScratch	1.3733	0.7386	0.4168
UM addRemove v2	1.4683	0.7614	0.4703

Table A.11.: Table showing the average performance of different regressors (with and without user model based features) on the UM data set in a query-based leave-one-out evaluation for the absolute versions of the metrics.

Regressor	**AV**	$P_{>0}$	$P_{>1}$
Bootstrap data set			
no-UM baseline	1.6094	0.8844	0.5172
no-UM addRemove	1.4672	0.8250	0.4641
Transfer condition			
no-UM baseline	1.2812	0.6931	0.3881
no-UM addRemove	1.3217	0.7178	0.4050

Table A.12.: Table comparing the two no-UM regressors both on the bootstrap data set using query-based leave-one-out and in the transfer condition for the absolute versions of the metrics.

A.4.4. Results

Table A.10 shows the absolute versions of the application-specific metrics as obtained in a user-based leave-one-out evaluation, Table A.11 shows them for the query-based evaluation and Table A.12 shows them for the application of the improved no-UM regressor on the bootstrap data set.

Preliminary Questionnaire

Before you can participate in the data collection user study, we need you to provide us with some information. We therefore would like to ask you to read and fill out this questionnaire.

The system under development is an interactive, personalized news retrieval system. Users can formulate a query to the system (similar to the queries used for search engines) and the system responds with information about a news event that is relevant to this query. The system covers news articles from November 2004 to October 2015. In order for the system to know which news are relevant, we need to gather some data from potential users (i.e. you).

Information about a user's general interests is represented as unordered set of keywords (i.e. all keywords are assumed to be equally important). Entities like “John Doe” are represented as a single keyword.

Please write down **two to six keywords that describe your general long-term interests** (please use correct capitalization):

- _____ • _____
- _____ • _____
- _____ • _____

Also a query to the system is represented as unordered set of keywords. Again, entities like “John Doe” are represented as a single keyword. A query for news events involving John Doe and Jane Roe would be represented as “John Doe, Jane Roe”.

Please formulate **twelve queries consisting of one to three keywords** (please use correct capitalization and separate the different keywords with commas):

- _____ • _____
- _____ • _____
- _____ • _____
- _____ • _____
- _____ • _____
- _____ • _____

Please return the compiled form at your earliest convenience. We will use this information to retrieve a set of news events for each of your queries. For each query, you will then be asked to judge the relevance of each retrieved news event. More detailed instructions will be provided together with the news events to be rated.

Thank you for participating!

Figure A.1.: The questionnaire used in the process of creating the UM data set.

Task Description

In the preliminary questionnaire, you provided us with information about your general interests and some example queries. You described your general interests with the following keywords:

- *<<Keyword 1>>*
- *<<Keyword 2>>*
- *<<Keyword 3>>*
- *<<Keyword 4>>*
- *<<Keyword 5>>*
- *<<Keyword 6>>*

In the attached Excel sheet you will find for each of your queries a number of potential system responses which were generated based on news articles. We would like you to perform the following task:

For each of the queries, read the different system responses and judge their relevance to both the query and your general interests. Please enter the relevance of each system response in the column “relevanceRank” using the following coding scheme:

Relevance of system response	Code
irrelevant	0
partially relevant	1
relevant	2
very relevant	3

Please return the completed Excel sheet as soon as you are done with labeling all examples. Please perform this task in one session, i.e. without taking breaks. We estimate that it should take you about 20 to 40 minutes to finish. If you have any questions about the task itself, please do not hesitate to contact us.

Thank you for participating in this data collection study!

Figure A.2.: The description of the labeling task used in the process of creating the UM data set.

B. Appendix: User Study Details

B.1. Survey Screenshots

Figures B.1 to B.8 contain screenshots of the different screens shown to the participants during the online survey. Note that the query formulation and the labeling of the corresponding system responses is shown only for the first question. The respective screens for the remaining four queries look identical are are therefore omitted.

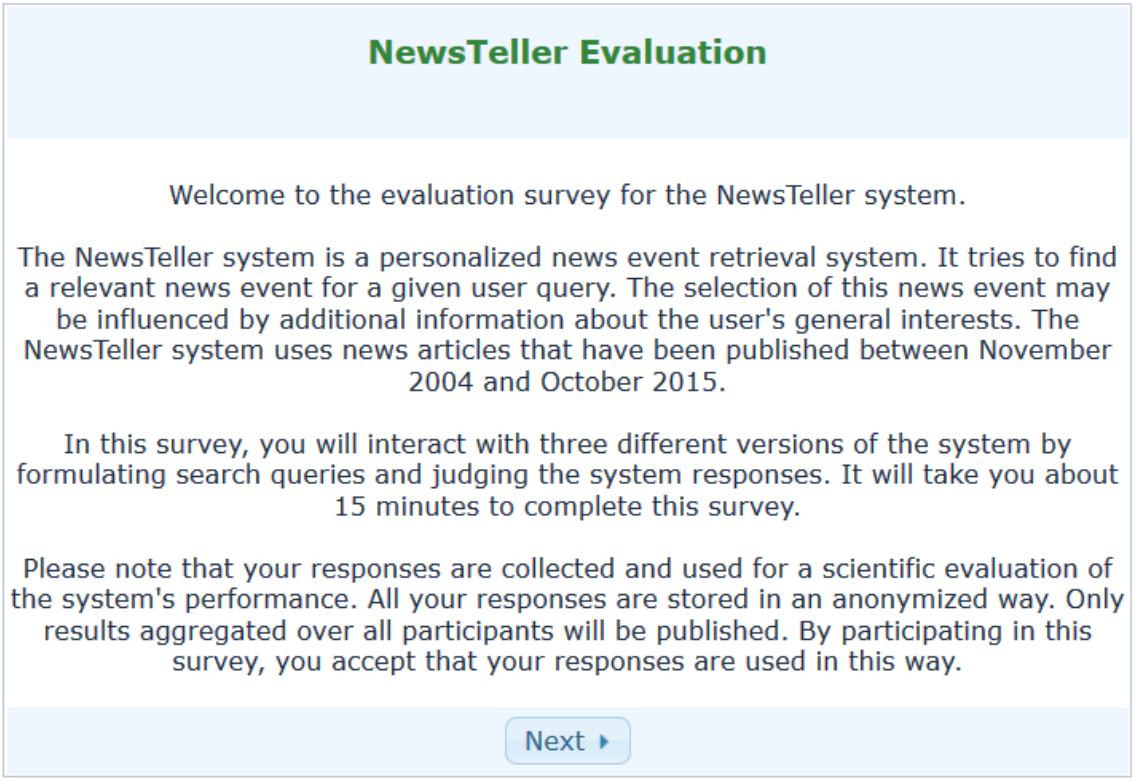

Figure B.1.: Welcome screen of the survey with some initial explanations.

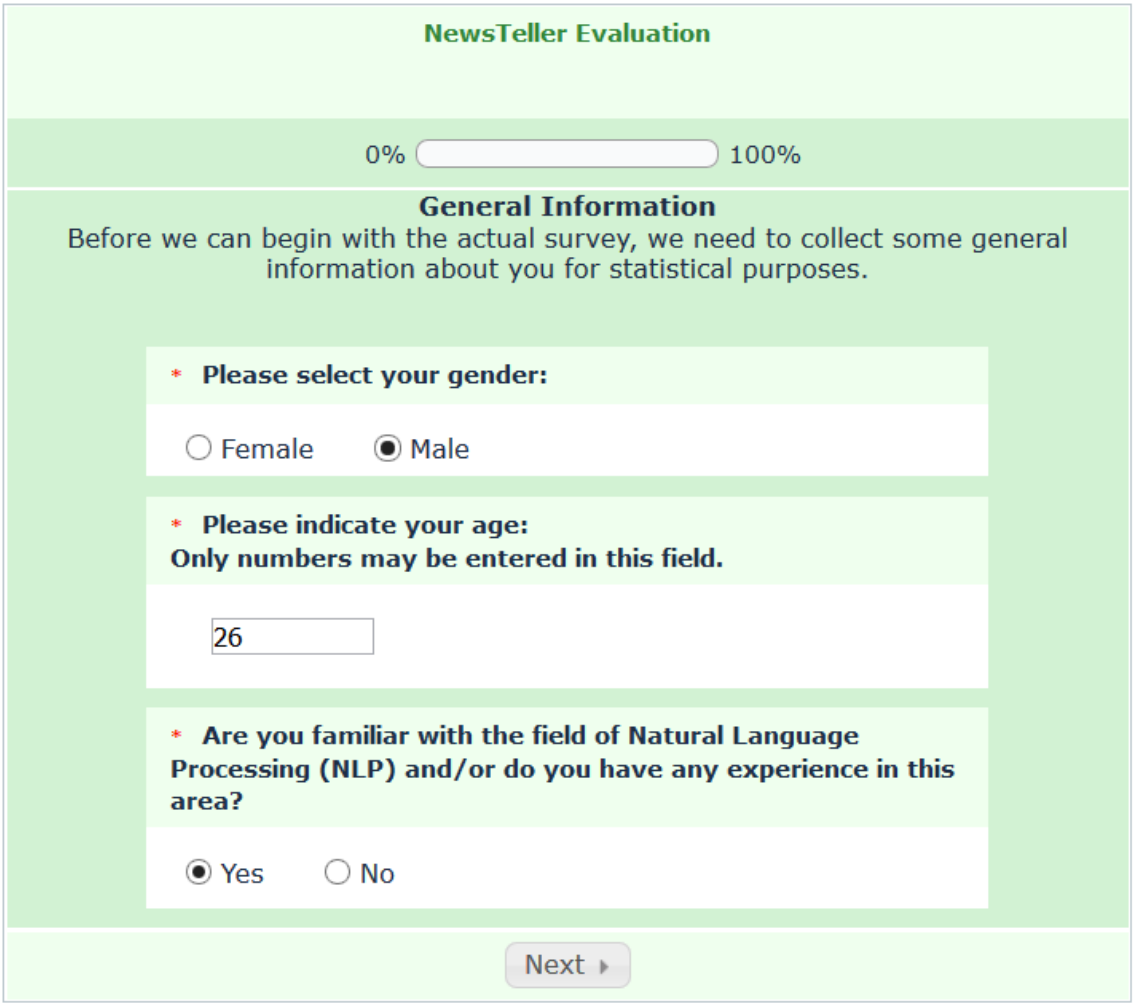

Figure B.2.: Second screen of the survey eliciting general information about the participants.

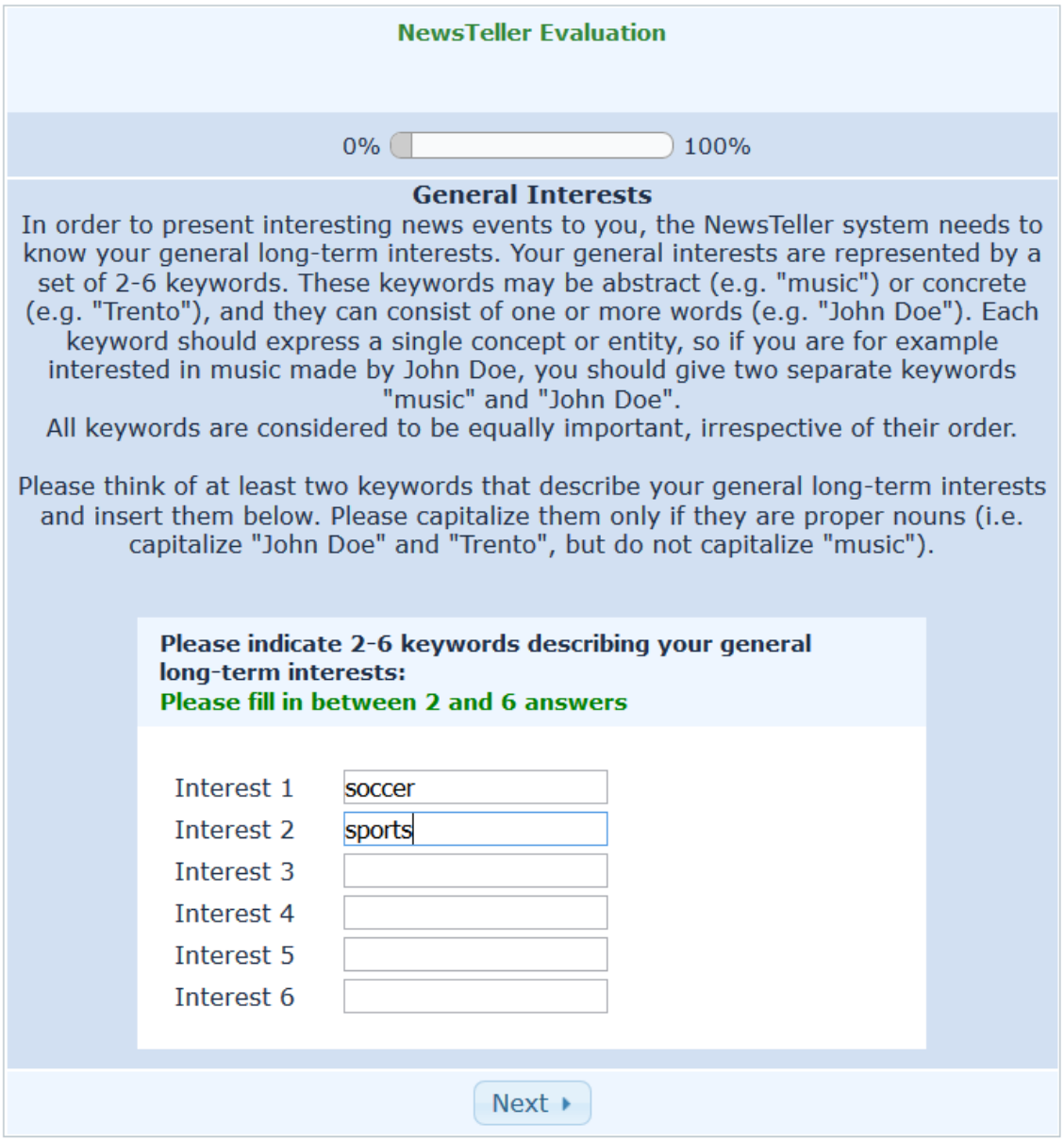

Figure B.3.: Third screen of the survey eliciting the participants' interests.

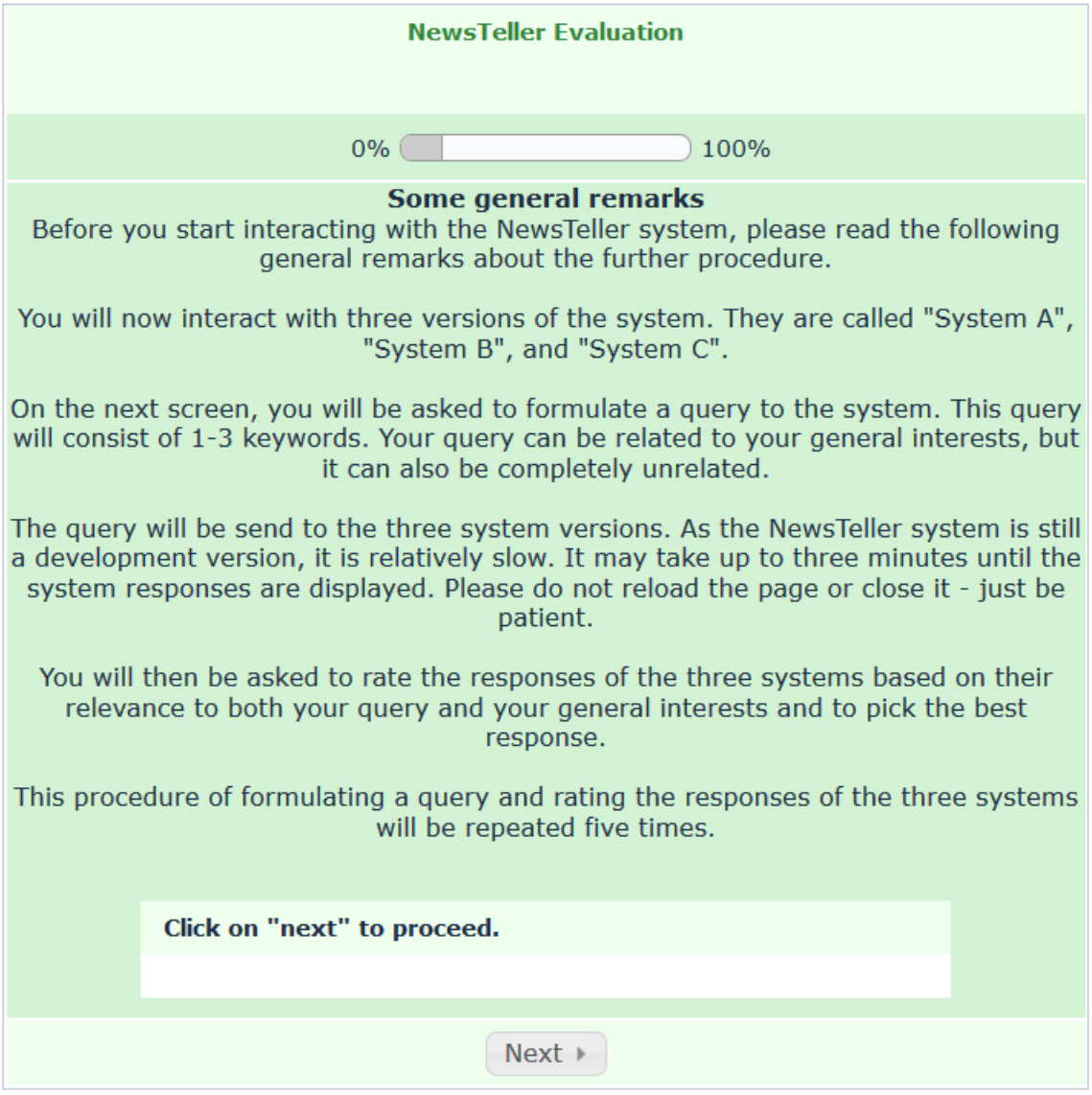

Figure B.4.: Fourth screen of the survey containing some more detailed information about the survey procedure.

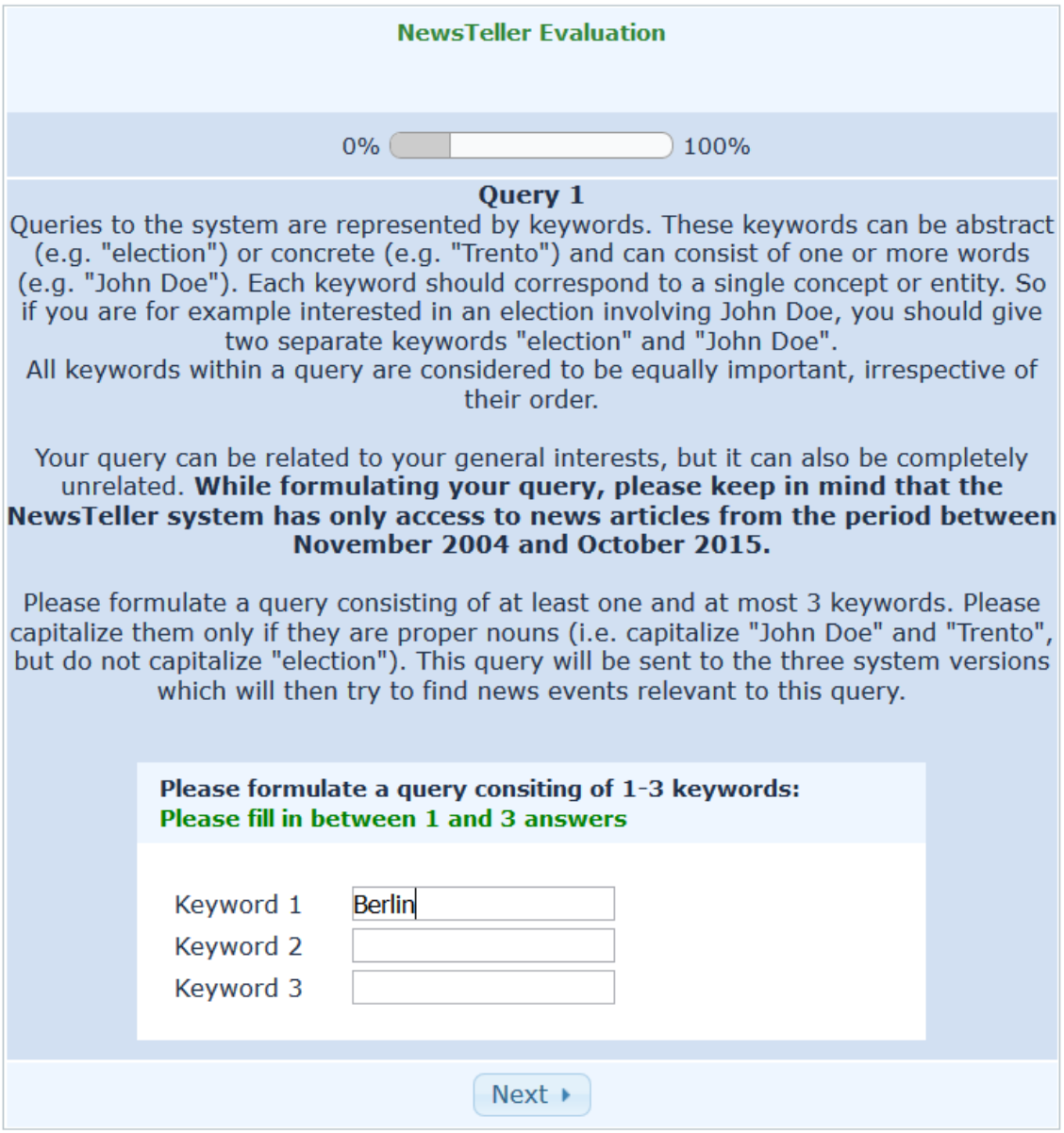

Figure B.5.: Fifth screen of the survey eliciting the first query to the NewsTeller system.

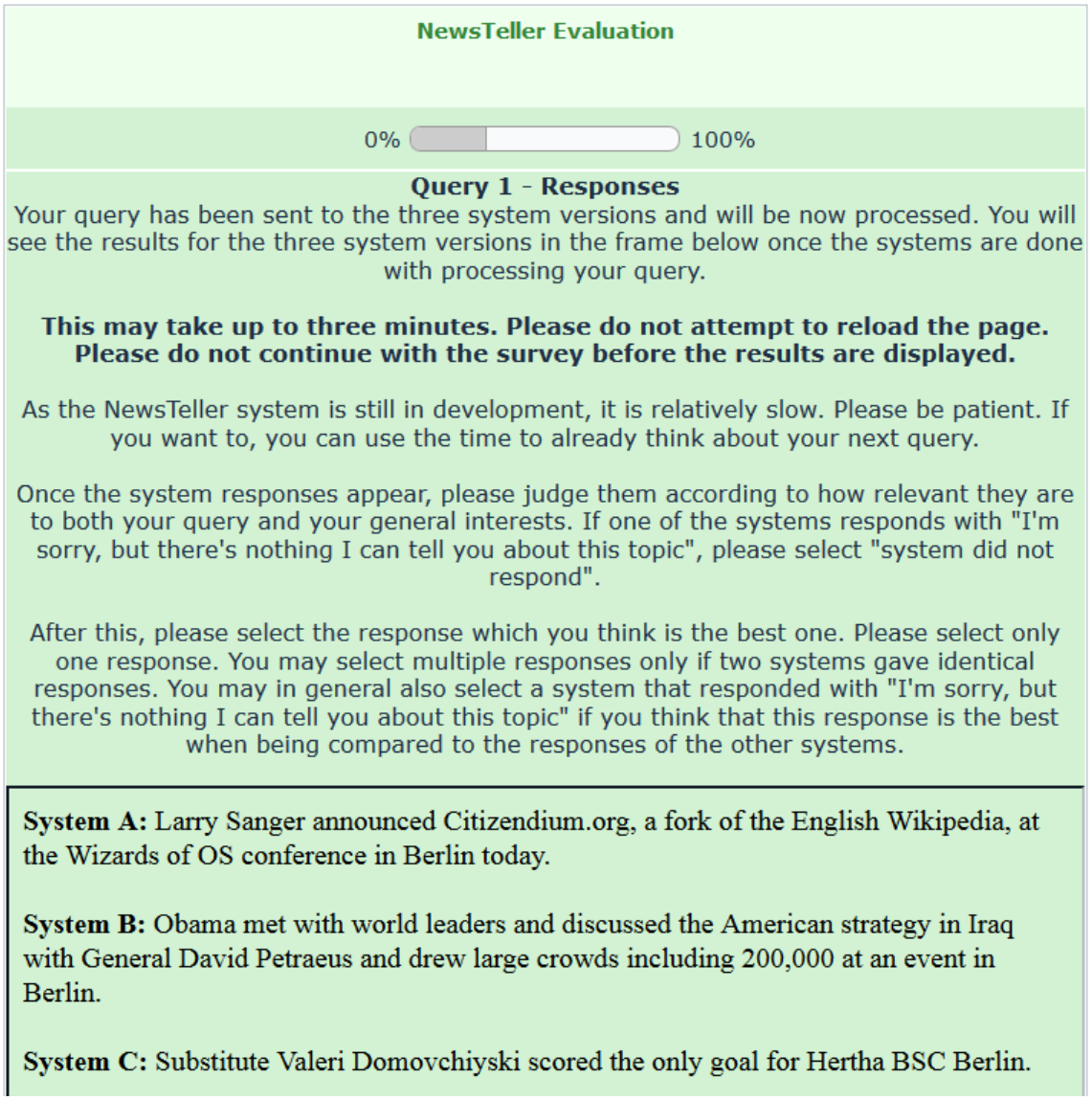

Figure B.6.: Upper half of the sixth screen of the survey presenting the system responses and asking the user to rate them.

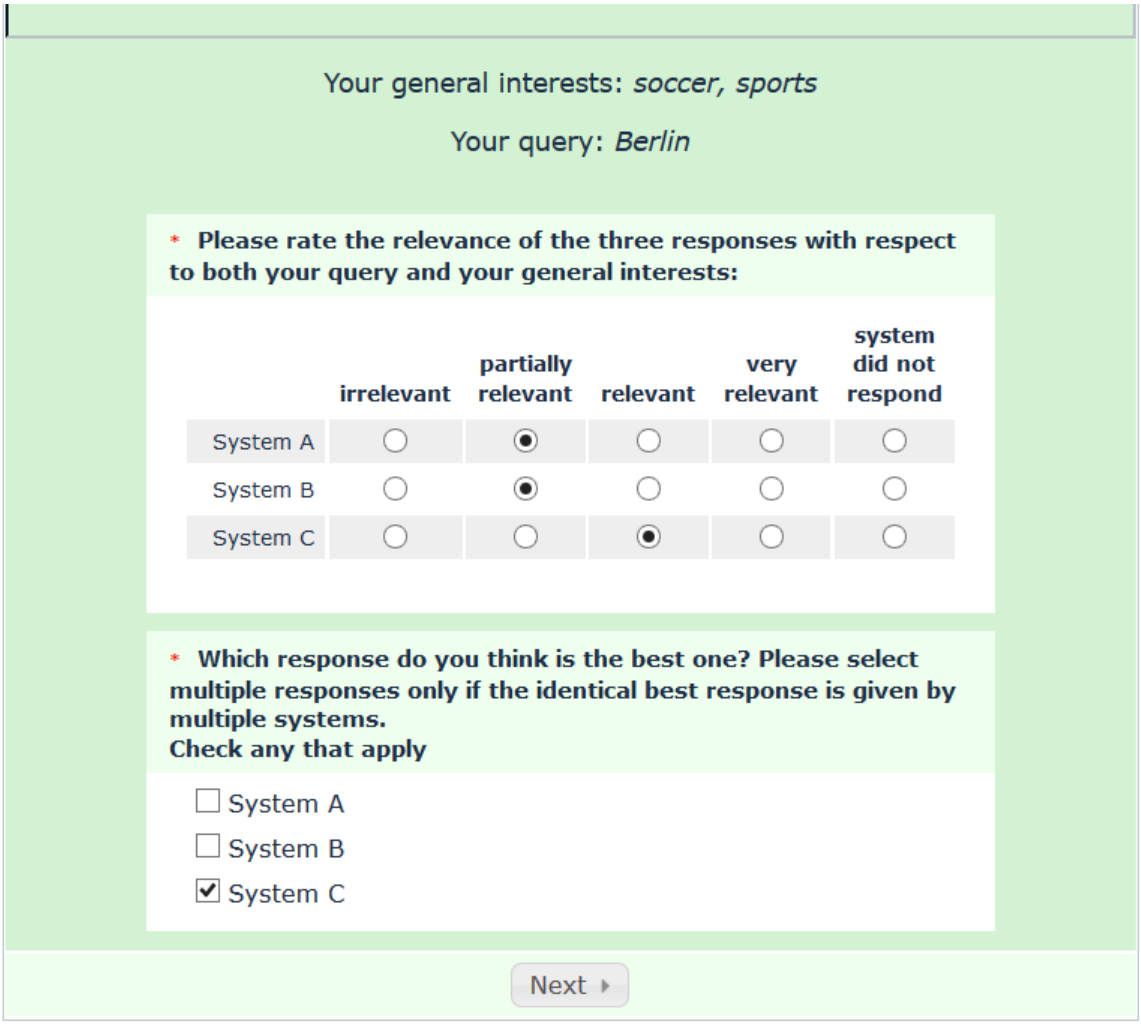

Figure B.7.: Lower half of the sixth screen of the survey presenting the system responses and asking the user to rate them.

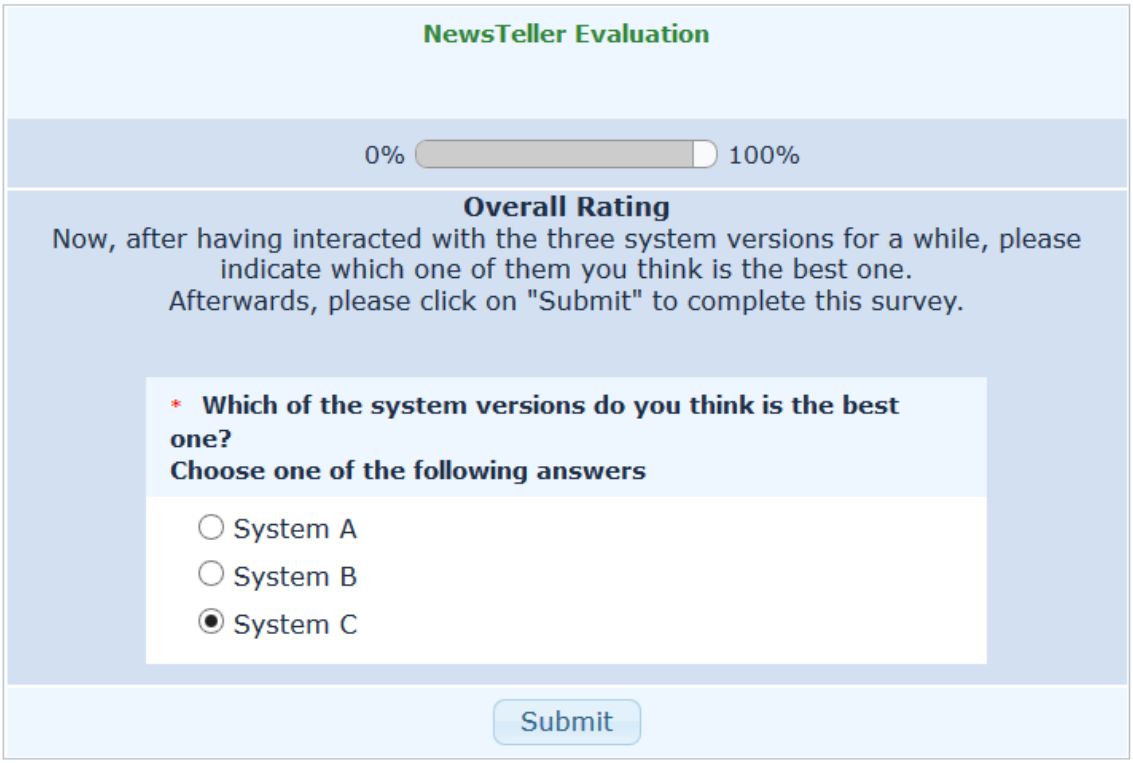

Figure B.8.: Last screen of the survey asking the user to pick the system with the best performance overall.

Configuration	AV	$P_{>0}$	$P_{>1}$
male	*1.1601*	*0.7172*	*0.2782*
Random	0.9823	0.6276	0.2552
Ranking-NoUM	1.2903	0.7724	0.2897
Ranking-UM	1.2400	0.7517	0.2897
female	*1.0000*	*0.6772*	*0.2175*
Random	0.8118	0.5684	0.1789
Ranking-NoUM	1.1538	0.7474	0.2316
Ranking-UM	1.0857	0.7158	0.2421

Table B.1.: Table showing the metrics AV, $P_{>0}$, and $P_{>1}$ for the three system configurations with respect to gender.

B.2. Additional Results

This section lists some additional results for two analyses: based on the participants' gender (Section B.2.1) and based on the participants' experience in the NLP field (Section B.2.2).

B.2.1. Gender

Table B.1 shows the metrics of the three systems, computed with respect to the participants' gender. We performed several χ^2 tests:

- Average Value (AV):
  - **Male: Random vs. Ranking-NoUM**: $\chi^2 = 7.46, p = 0.0587$
  - **Male: Random vs. Ranking-UM**: $\chi^2 = 5.97, p = 0.1129$
  - **Male: Ranking-NoUM vs. Ranking-UM**: $\chi^2 = 0.59, p = 0.8986$
  - **Female: Random vs. Ranking-NoUM**: $\chi^2 = 8.92, p = 0.0303$
  - **Female: Random vs. Ranking-UM**: $\chi^2 = 7.56, p = 0.0560$
  - **Female: Ranking-NoUM vs. Ranking-UM**: $\chi^2 = 1.19, p = 0.7566$
  - **Male-Random vs. female-Random**: $\chi^2 = 11.00, p = 0.0117$
  - **Male-Ranking-NoUM vs. female-Ranking-NoUM**: $\chi^2 = 6.08, p = 0.1080$
  - **Male-Ranking-UM vs. female-Ranking-UM**: $\chi^2 = 2.90, p = 0.4075$
  - **Male vs. female**: $\chi^2 = 18.33, p = 0.0004$
- Precision > 0 ($P_{>0}$):
  - **Male: Random vs. Ranking-NoUM**: $\chi^2 = 13.01, p = 0.0003$
  - **Male: Random vs. Ranking-UM**: $\chi^2 = 9.56, p = 0.0020$
  - **Male: Ranking-NoUM vs. Ranking-UM**: $\chi^2 = 0.35, p = 0.5524$

  - **Female: Random vs. Ranking-NoUM**: $\chi^2 = 12.40, p = 0.0004$
  - **Female: Random vs. Ranking-UM**: $\chi^2 = 8.41, p = 0.0037$
  - **Female: Ranking-NoUM vs. Ranking-UM**: $\chi^2 = 0.50, p = 0.4787$
  - **Male-Random vs. female-Random**: $\chi^2 = 1.42, p = 0.2329$
  - **Male-Ranking-NoUM vs. female-Ranking-NoUM**: $\chi^2 = 0.34 p = 0.5604$
  - **Male-Ranking-UM vs. female-Ranking-UM**: $\chi^2 = 0.66, p = 0.4175$
  - **Male vs. female**: $\chi^2 = 2.25, p = 0.1333$

- Precision > 1 ($P_{>1}$):
  - **Male: Random vs. Ranking-NoUM**: $\chi^2 = 0.91, p = 0.3409$
  - **Male: Random vs. Ranking-UM**: $\chi^2 = 0.91, p = 0.3409$
  - **Male: Ranking-NoUM vs. Ranking-UM**: $\chi^2 = 0.00, p = 1.0000$
  - **Female: Random vs. Ranking-NoUM**: $\chi^2 = 1.79, p = 0.1808$
  - **Female: Random vs. Ranking-UM**: $\chi^2 = 2.58, p = 0.1083$
  - **Female: Ranking-NoUM vs. Ranking-UM**: $\chi^2 = 0.06, p = 0.8078$
  - **Male-Random vs. female-Random**: $\chi^2 = 2.90, p = 0.0884$
  - **Male-Ranking-NoUM vs. female-Ranking-NoUM**: $\chi^2 = 1.56 p = 0.2121$
  - **Male-Ranking-UM vs. female-Ranking-UM**: $\chi^2 = 1.04, p = 0.3069$
  - **Male vs. female**: $\chi^2 = 5.22, p = 0.0224$

With respect to the "best response" question, male participants selected the Random baseline in 55.86% of the cases, the Ranking-NoUM system in 59.31% of the cases, and the Ranking-UM approach in 62.76% of the cases. Female participants, however, selected the Random baseline in 47.36% of the cases, the Ranking-NoUM system in 54.73% of the cases, and the Ranking-UM approach in 57.89% of the cases.

We tested the following differences:

- **Male: Random vs. Ranking-NoUM**: $\chi^2 = 0.70, p = 0.4030$
- **Male: Random vs. Ranking-UM**: $\chi^2 = 2.80, p = 0.0944$
- **Male: Ranking-NoUM vs. Ranking-UM**: $\chi^2 = 0.71, p = 0.3980$
- **Female: Random vs. Ranking-NoUM**: $\chi^2 = 2.07, p = 0.1503$
- **Female: Random vs. Ranking-UM**: $\chi^2 = 4.22, p = 0.0399$
- **Female: Ranking-NoUM vs. Ranking-UM**: $\chi^2 = 0.38, p = 0.5363$
- **Male-Random vs. female-Random**: $\chi^2 = 2.78, p = 0.0955$
- **Male-Ranking-NoUM vs. female-Ranking-NoUM**: $\chi^2 = 0.82 p = 0.3642$

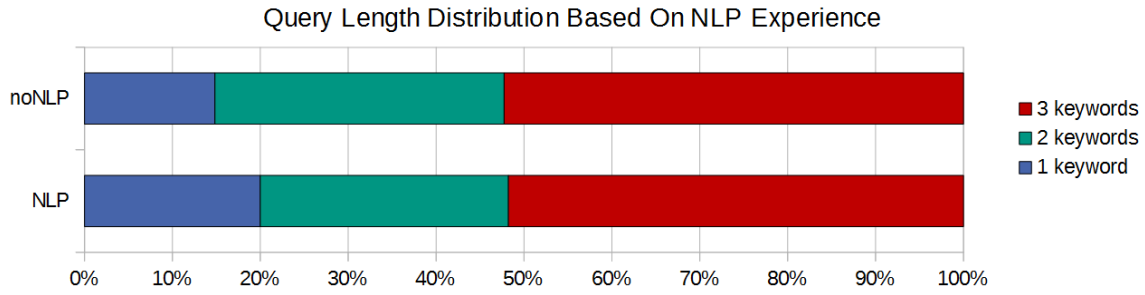

Figure B.9.: Distribution of query length in number of keywords differentiated by the participants' NLP experience. "noNLP" stands for "no experience with NLP", and "NLP" stands for "experienced in NLP".

Configuration	**AV**	$P_{>0}$	$P_{>1}$
NLP	*1.0354*	*0.6549*	*0.2392*
RANDOM	0.8667	0.5765	0.2118
RANKING-NOUM	1.2105	0.7412	0.2588
RANKING-UM	1.0758	0.6471	0.2471
noNLP	*1.1280*	*0.7269*	*0.2624*
RANDOM	0.9350	0.6194	0.2323
RANKING-NOUM	1.2475	0.7742	0.2710
RANKING-UM	1.2404	0.7871	0.2839

Table B.2.: Table showing the metrics AV, $P_{>0}$, and $P_{>1}$ for the three system configurations with respect to NLP experience.

- **Male-RANKING-UM vs. female-RANKING-UM**: $\chi^2 = 0.96, p = 0.3268$

With respect to the final question about the best system, 34.48% of the male participants selected the RANDOM baseline, 17.24% the RANKING-NOUM system and 48.28% the RANKING-UM approach. When looking only at female participants, 26.32% selected the RANDOM baseline, 31.58% the RANKING-NOUM system and 42.11% the RANKING-UM approach. The distribution induced by male participants did not differ significantly from a uniform distribution ($\chi^2 = 4.21, p = 0.1220$) although a slight trend is visible. Also the distribution induced by female participants did not differ significantly from a uniform distribution ($\chi^2 = 0.74, p = 0.6918$). Both distributions did not differ from each other significantly ($\chi^2 = 2.78, p = 0.2487$).

B.2.2. NLP Experience

Figure B.9 illustrates the length of the user queries for participants without prior experience in the NLP field ("noNLP") and for participants with experience in the NLP area ("NLP"). Participants without any NLP experience used on average 2.37 keywords per query, whereas participants with an NLP background used on average 2.32 keywords per query. This difference is not statistically significant ($\chi^2 = 3.27, p = 0.1952$).

Table B.2 shows the metrics of the three systems, computed with respect to the participants' NLP experince. "NLP" stands for participants with a background in NLP whereas "noNLP" denotes participants without any experience in the field of NLP. We performed several χ^2 tests:

- Average Value (AV):
  - **NLP: Random vs. Ranking-NoUM**: $\chi^2 = 8.12, p = 0.0436$
  - **NLP: Random vs. Ranking-UM**: $\chi^2 = 7.99, p = 0.0462$
  - **NLP: Ranking-NoUM vs. Ranking-UM**: $\chi^2 = 2.62, p = 0.4547$
  - **noNLP: Random vs. Ranking-NoUM**: $\chi^2 = 9.15, p = 0.0287$
  - **noNLP: Random vs. Ranking-UM**: $\chi^2 = 14.76, p = 0.0020$
  - **noNLP: Ranking-NoUM vs. Ranking-UM**: $\chi^2 = 2.71, p = 0.4388$
  - **NLP-Random vs. noNLP-Random**: $\chi^2 = 2.76, p = 0.4295$
  - **NLP-Ranking-NoUM vs. noNLP-Ranking-NoUM**: $\chi^2 = 3.17, p = 0.3655$
  - **NLP-Ranking-UM vs. noNLP-Ranking-UM**: $\chi^2 = 40.36, p = 8.95 \cdot 10^{-9}$
  - **NLP vs. noNLP**: $\chi^2 = 24.51, p = 1.95 \cdot 10^{-5}$
- Precision > 0 ($P_{>0}$):
  - **NLP: Random vs. Ranking-NoUM**: $\chi^2 = 9.44, p = 0.0021$
  - **NLP: Random vs. Ranking-UM**: $\chi^2 = 1.73, p = 0.1878$
  - **NLP: Ranking-NoUM vs. Ranking-UM**: $\chi^2 = 3.93, p = 0.0476$
  - **noNLP: Random vs. Ranking-NoUM**: $\chi^2 = 15.76, p = 7.18 \cdot 10^{-5}$
  - **noNLP: Random vs. Ranking-UM**: $\chi^2 = 18.50, p = 1.70 \cdot 10^{-5}$
  - **noNLP: Ranking-NoUM vs. Ranking-UM**: $\chi^2 = 0.15, p = 0.7008$
  - **NLP-Random vs. noNLP-Random**: $\chi^2 = 1.17, p = 0.2799$
  - **NLP-Ranking-NoUM vs. noNLP-Ranking-NoUM**: $\chi^2 = 0.88, p = 0.3480$
  - **NLP-Ranking-UM vs. noNLP-Ranking-UM**: $\chi^2 = 13.31, p = 0.0003$
  - **NLP vs. noNLP**: $\chi^2 = 10.66, p = 0.0011$
- Precision > 1 ($P_{>1}$):
  - **NLP: Random vs. Ranking-NoUM**: $\chi^2 = 1.13, p = 0.2883$
  - **NLP: Random vs. Ranking-UM**: $\chi^2 = 0.63, p = 0.4258$
  - **NLP: Ranking-NoUM vs. Ranking-UM**: $\chi^2 = 0.06, p = 0.8044$
  - **noNLP: Random vs. Ranking-NoUM**: $\chi^2 = 1.30, p = 0.2538$
  - **noNLP: Random vs. Ranking-UM**: $\chi^2 = 2.32, p = 0.1281$
  - **noNLP: Ranking-NoUM vs. Ranking-UM**: $\chi^2 = 0.13, p = 0.7178$
  - **NLP-Random vs. noNLP-Random**: $\chi^2 = 0.39, p = 0.5323$

- **NLP-Ranking-NoUM vs. noNLP-Ranking-NoUM**: $\chi^2 = 0.12, p = 0.7299$
- **NLP-Ranking-UM vs. noNLP-Ranking-UM**: $\chi^2 = 1.13, p = 0.2880$
- **NLP vs. noNLP**: $\chi^2 = 1.37, p = 0.2419$

With respect to the "best response" question, participants from the "NLP" group selected the Random baseline in 49.42% of the cases, the Ranking-NoUM system in 57.65% of the cases, and the Ranking-UM approach in 57.65% of the cases. Participants from the "noNLP" group, however, selected the Random baseline in 54.19% of the cases, the Ranking-NoUM system in 57.42% of the cases, and the Ranking-UM approach in 62.58% of the cases.

We tested the following differences:

- **NLP: Random vs. Ranking-NoUM**: $\chi^2 = 2.31, p = 0.1289$
- **NLP: Random vs. Ranking-UM**: $\chi^2 = 2.31, p = 0.1289$
- **NLP: Ranking-NoUM vs. Ranking-UM**: $\chi^2 = 0.00, p = 1.0000$
- **noNLP: Random vs. Ranking-NoUM**: $\chi^2 = 0.65, p = 0.4202$
- **noNLP: Random vs. Ranking-UM**: $\chi^2 = 4.39, p = 0.0361$
- **noNLP: Ranking-NoUM vs. Ranking-UM**: $\chi^2 = 1.69, p = 0.1938$
- **NLP-Random vs. noNLP-Random**: $\chi^2 = 1.42, p = 0.2338$
- **NLP-Ranking-NoUM vs. noNLP-Ranking-NoUM**: $\chi^2 = 0.003, p = 0.9542$
- **NLP-Ranking-UM vs. noNLP-Ranking-UM**: $\chi^2 = 1.55, p = 0.2138$

With respect to the final question about the best system, 29.41% of the participants with an NLP background selected the Random baseline, 23.53% the Ranking-NoUM system and 47.06% the Ranking-UM approach. When looking only at participants without experience in the NLP field, 32.26% selected the Random baseline, 22.58% the Ranking-NoUM system and 45.16% the Ranking-UM approach. The distribution induced by participants from the "NLP" group did not differ significantly from a uniform distribution ($\chi^2 = 1.53, p = 0.4655$). Also the distribution induced by participants from the "noNLP" group did not differ significantly from a uniform distribution ($\chi^2 = 2.39, p = 0.3031$). Both distributions did not differ from each other significantly ($\chi^2 = 0.12, p = 0.9413$).