UNIVERSITÄT KARLSRUHE (TH)
FAKULTÄT FÜR INFORMATIK
INSTITUT FÜR ANTHROPOMATIK
Prof. Dr. A. Waibel

THESIS

Language Model Adaptation using Interlinked Semantic Data

SUBMITTED BY

Kevin Kilgour

MAI 2009

ADVISORS

Dipl.-Inform. Florian Kraft
Prof. Dr.rer.nat Alex Waibel

interACT
Institut für Anthropomatik
Universität Karlsruhe (TH)
Title: Language Model Adaptation using Interlinked Semantic Data
Author: Kevin Kilgour

Kevin Kilgour
D-Bonhöffer-Str 31
76461 Muggensturm
email: kevin.kilgour@gmail.com

Statement of authorship

I hereby declare that this thesis is my own original work which I created without illegitimate help from others, that I have not used any sources or resources other than the ones indicated and that due acknowledgement is given where reference is made to the work of others.

Karlsruhe, 31. Mai 2009

. . . . . .
(Kevin Kilgour)

Acknowledgments

I would like to thank everyone who has supported and helped me during my work on this diploma thesis. In particular I would like to thank Florian Kraft for being a great advisor, allowing me a lot of freedom and giving me helpful writing guidelines. Had it not been for Florian throwing me in at the deep end with the Janus language model source code last year I might not have discovered how interesting language modeling is. Thanks also to Prof. Alex Waibel for letting me write this diploma thesis at his institute and Christian Fügen whose language model's source code I used as a template for my language model.

I also have to thank my father for proofreading this thesis multiple times and my mother who has in the past months often provided me with free food. Thanks go also to my sisters, friends and fellow students for their support and encouragement. And last but not least I would like to thank Kira for her patience and support when I couldn't spend a lot of time with her.

Abstract

This thesis presents an adaptive multi domain language model built from large sources of human created, interlinked and structured data. The sources' interlinked structure is used to create multiple n-gram language models which are dynamically interpolated to produce a context dependant language model. The language model is evaluated on its performance with a speech recognition system used to decode European parliament recordings and compared to both a general purpose language model trained on the same data and a manually adapted and built domain language model.

Contents

List of Figures

List of Tables

1 Introduction

There are three main driving forces behind the development of natural language processing (NLP) technologies. The first is the desire to improve the interaction of man[1] and machine by giving machines the ability to understand or at least extract some meaning from their operators' native language. This is in stark contrast to earlier machine development. Since the industrial revolution people have been having to adapt to machines by learning special skills to operate them.

It is not just interaction of man and machine that can benefit from improved computer understanding of natural languages. Such computer systems can also aide the interaction between people. Having a machine translate for two people who do not speak a common language is an obvious example but systems that can work with natural languages could do even more; like transcribing and summarizing business meetings.

The third factor is the ever increasing amount of natural language data, text, audio and video[2] that is stored in computer systems around the world. Systems designed to sort, categorize, or extract information from these data repositories also need to be able to deal with natural languages.

The language model, a mathematical model with similar characteristics and properties to the real language, is the most important part of a natural language processing systems. This thesis presents a novel way of building a language model and adapting it at run-time. The first sections of this chapter will explain why it necessary to adapt a language model and what the objectives of this thesis are. In order to evaluate the adaptive language model it was incorporated into a speech recognition system. The last parts of this chapter briefly explain the history and concepts of speech recognition.

Chapter 2 explains in more detail the concept of language modeling, in particular the details of statistical language modeling as well as reviewing other attempts at building adaptive language

[1] In the context of this thesis, man is used as a short from of word *human* and does not mean an adult male Homo Sapien.

[2] Sign language is also a natural language.

models. The final part introduces some basic text classification techniques and shows how they are used in language modeling.

Chapter 3 gives an overview of possible interlinked sources, describes how the data needed to build an adaptive language model can be extracted from these sources and what the resulting language model looks like.

A language model on its own is not very useful so chapter 4 elaborates on how the Janus (speech) Recognition Toolkit (JRTk) interfaces with language models and how this new adaptive language model is incorporated into it. This chapter also goes into detail on what *"history"* the language model can adapt to.

The evaluation is performed on a set of recordings from the European parliament where the perplexity and word error rate are measured using different *histories* and language model configurations. These are compared to the values measured on a baseline language model and presented in chapter 5.

Chapter 6 discusses the results of these experiments and finally chapter 7 presents possible future improvements to this language model.

1.1 Importance and Effects of Language Model Adaptation in NLP

Most state of the art language models used in speech recognition and machine translation are statistical language models trained from a large amount of language data. The better the data represents the type of language the model will encounter, the better the language model will be. This means that having more data does not necessarily lead to a better language model. The data has to be relevant to the domain of the language model.

It can be easily seen that language models built for a particular domain will perform better on that domain than a general purpose language model. The previously mentioned language model used to transcribe business meetings will pretty much never have to deal with the sequence of words *"high altitude cerebral edema"* whereas a general purpose language model that could also be used in hospitals and by rock climbers would have to deal with this and many other similar word sequences like it.

An adaptive language model should recognize the domain in which it is being used and adjust itself accordingly. This thesis describes how to build just such a language model by utilizing large data sources with semantic meta information. Latent domains or concepts are extracted and only data from the relevant concepts are used when decoding on a particular domain.

1.1.1 Objectives

To achieve this goal the following objectives are set.

- Identifying and acquiring possible interlinked data sources with the necessary meta information.
- Extracting the latent domains or concepts from the data sources and associating them with the appropriate text data.
- Constructing a system that selects the relevant concepts based on a short history.
- Training and constructing the adaptive language model from the data.
- Implementing a language model decoding module in the JRTk.
- Evaluating the language model.

1.2 History of Automatic Speech Recognition

The great physicist and mathematician Sir Isaac Newton once said:

> "*If I have seen a little further it is by standing on the shoulders of giants.*"

And what was true 300 years ago for Newton is even more true today. Novel and groundbreaking tools and ideas of one generation lay the groundwork for the tools and ideas of the next, which is why this history of automatic speech recognition (ASR) does not start with the first ASR system but with Harvey Fletcher, one of the giants on whose shoulders the field stands. Before anyone could build a machine to recognize speech one first had to understand the physical properties of speech. Fletcher's research in this area showed that the important features of speech are contained within its time-varying frequency spectrum [FotFI22].

Our vocal tract can by considered as an acoustic tube with resonance frequencies called **formants**. The shape of this acoustic tube can be altered by moving the tongue vertically (e.g. **[i:]** in *beat* vs **[e]** in *bet*), horizontally (e.g. **[i:]** in *beat* vs **[u:]** in *boot*) and by changing the shape of the lips

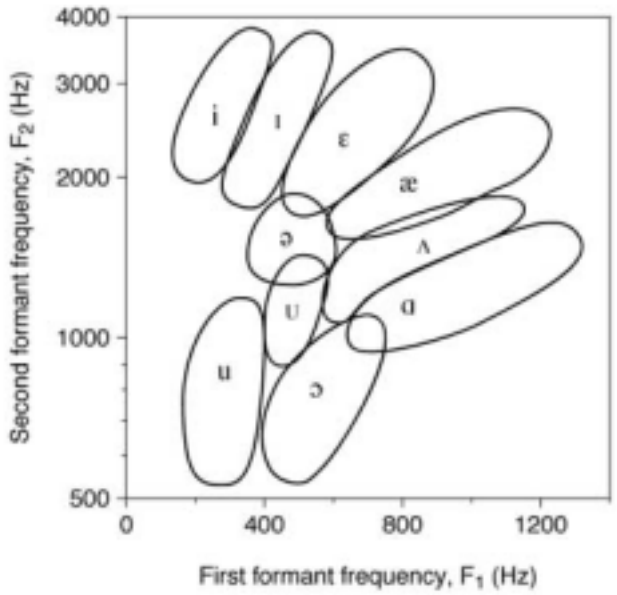

Figure 1.1: The first (F1) and second (F2) formants and how they correspond to vowels. Source: [PB52]

(e.g. **[æ]** in *bat* vs **[o]** in *boat*) which leads to different formants for each voiced sound. In practice only the first two formants F1 and F2 are needed to identify a vowel (see Figure 1.1).

On the basis of Fletcher's research Homor Dudley at Bell Labs built his speech synthesizer VODER (Voice Operating Demonstrator [Dud39]), which allowed the operator to *play* a sentence by adjusting the output level of 10 bandpass filters through which a base signal was passed. Previous attempts to build such machines had been purely mechanical in nature using either acoustic resonance tubes, like Russian scientist Christian Kratzenstein (1773) and Wolfgang von Kempelen from Vienna (1791), or resonators made of leather like Charles Wheatstone (mid-1800's) [JR]). These machines could only produce "speech-like" sounds and not understandable sentences. When Dudley's VODER was demonstrated at the 1939 Word Fair in New York City (Figure 1.2) it was considered revolutionary.

The first attempt to build a machine that could recognize some isolated spoken words of a single speaker was also undertaken at Bell Labs. The machine built by Davis, Biddulph, and Balashek in 1952 could recognize the isolated digits "oh" and "one" through "nine". It assumed that each input utterance was exactly one of these digits and didn't contain any other sounds It measured the formant frequencies and the change in formant frequencies in the utterance and compared this to previously generated reference patterns (Figur 1.3) for each digit [DBB52].

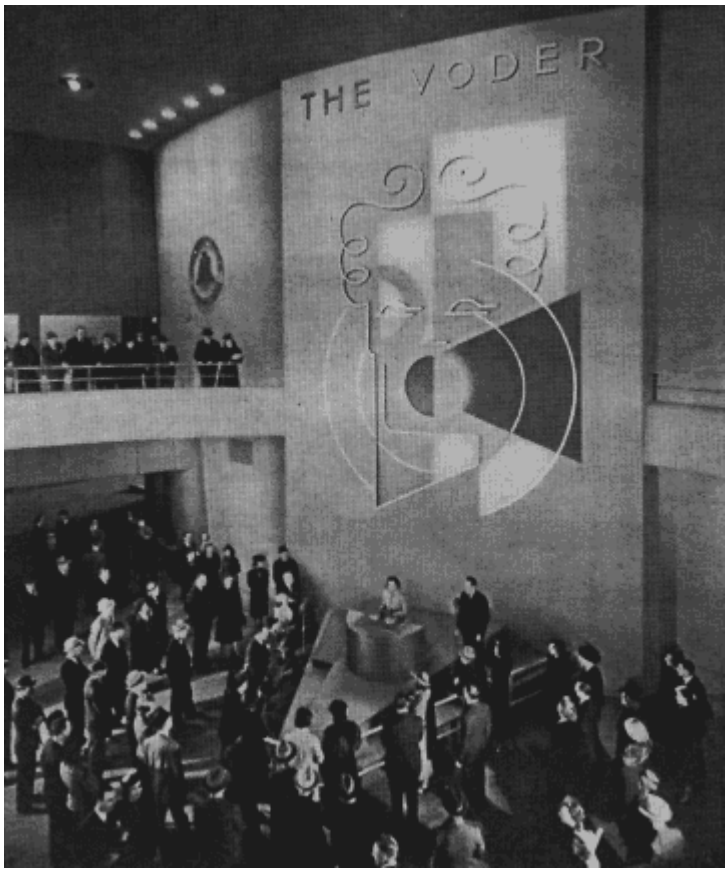

Figure 1.2: Bell Labs engineer Homer Dudley's speech synthesis machine Voder is demonstrated at the 1939 World Fair in New York City.

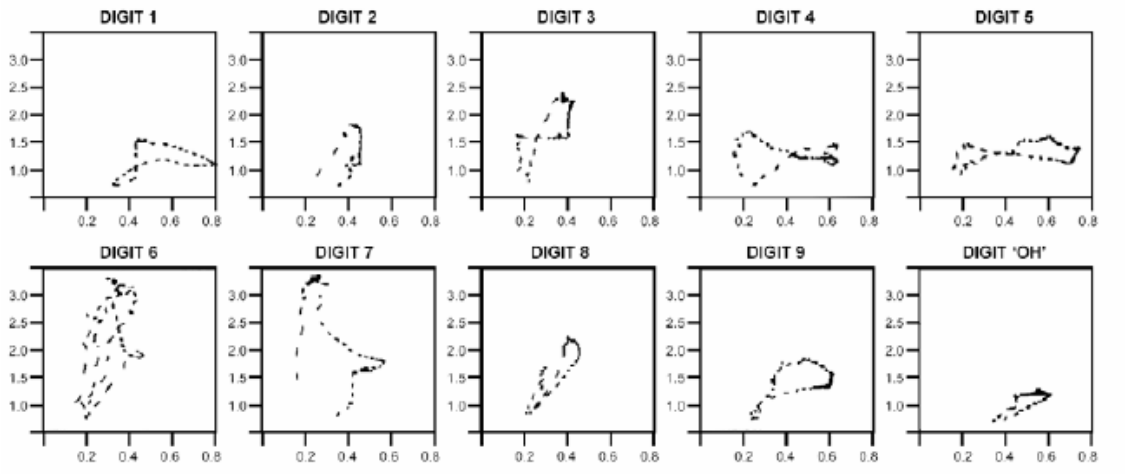

Figure 1.3: Davis' reference patterns. Source: [DBB52].

Similar systems were built by Forgie and Forgie at MIT Lincoln Lab [FF59] and by Suzuki and Nakata at the Radio Research Lab in Tokyo [SN61]. They concentrated on vowel detection and more or less ignored the more difficult to detect consonants. This example from Tanja Schulz [Sch08] shows just how useful consonants are for ASR:

```
_E__ _I__OU_ _O_E__ I_ _O_ _I__I_U__ _O U__E__A__

T_XT W_TH__T V_W_LS _S N_T D_FF_C_LT T_ _ND_RST_ND
```

Fry and Denes at University College in England [FD58] and Sakai and Doshita at Kyoto University [SD64] both built phoneme recognizers that could detect consonants. A **phoneme** is the smallest unit of speech which differentiates the meaning of a word pair. The only difference between *bush* and *push* is the change of the phoneme /b/ to /p/. Phonemes can sound quite different in a different context or when spoken by a different speaker. An example of a phoneme is called a **phone**.

Another problem ASR systems have to deal with is the non-uniform time scale of utterances; the speed with which an utterance is spoken can be faster or slower than its reference pattern and the change in this speed is not necessarily constant. Sometimes one part of an utterance is spoken faster than the reference while other parts are spoken slower. To solve this, so called, alignment problem Russian scientist Vintsyuk proposed using a variant of the minimal editing distance metric, which can be efficiently computed using dynamic programming, to compare the utterance with the hypotheses [Vin68, Vin71].

In 1971 APRA[3] (Advanced Research Projects Agency) launched its Speech Understanding Research program (SUR) with the ambitious five year goal of building ASR systems with 1000-word vocabulary, less than 10% WER[4] and near real time capabilities. The best system built by SUR participents was Carnegie Mellon University's *Harpy* which had a vocabulary of 1011 words and only 5% WER but its runtime was about 100 times real-time [Low76]. Speech input into the Harpy system was first segmented and then the segments were matched to phone templetes using the Itakura distance [Mar89]. The number of hypotheses was constrained by a predefined knowledge system and a graph search algorithm similar to the modern beam search algorithm was used to find the best hypothesis.

Independently from one another in the late 1970s both AT&T Bell Laboratories and IBM introduced statistical methods to speech recognition. Bell Laboratories' goal was to provide telecommunication services to the public and because these services had to work with millions of different

[3] later renamed to Defense Advanced Research Projects Agency (DARPA)

[4] Word Error Rate, see chapter 2.1.4

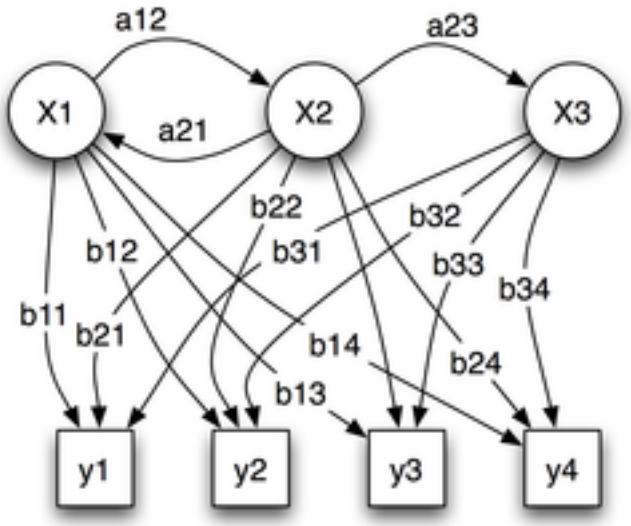

Figure 1.4: Hidden Markov Model example. X: states, y: observations, a:transition probabilities, b: output probabilities. Source: http://en.wikipedia.org/wiki/Image:HiddenMarkovModel.png

people they had to be speaker-independent. Instead of phone templates they used statistical models and later hidden Markov models (HMM) which have since become standard for representing speech units [LRS83]. Hidden Markov models are used to model statistical systems with a set of hidden states ($X_1, X_2, ...$), possible observations ($y_1, y_2, ..$) and parameters. For each state X_i the probability of observing y_j is b_{ij} and the probability of transitioning to X_j is a_{ij} (Figure 1.4). The Baum-Welch algorithm can be used to compute the parameters of an HMM if a large enough set of observation sequences are available. Meanwhile, over at IBM, Fred Jelinek's team were busy trying to build a *voice-activated typewriter* (VAT) to be used in office correspondence [JBM75]. Their VAT was the first ASR system to use a statistical language model, which given multiple hypotheses that fit an input utterance, chooses the hypotheses that is most likely to occur in the language.

1.3 Modern Automatic Speech Recognition Systems

Over the years the hidden Markov models developed at Bell Laboratories have been improved upon and are now a vital part of speech recognition systems. The, so called, acoustic model uses

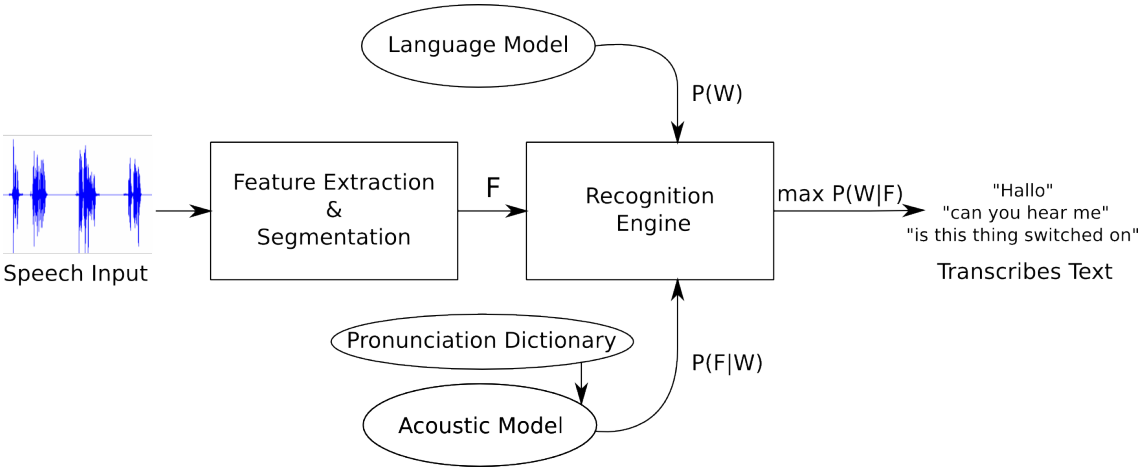

Figure 1.5: Components of a Speech Recognition System.

a probability distribution of features to describe the smallest units of speech, the phones or subphones. The pronunciation dictionary associated with an acoustic model maps words to sequences of phones. This allows the acoustic model to, for a given word w, calculate the likelihood of a sequence of feature vector F and can be written as a conditional probability $P(F|w)$ [You96].

The goal of automatic speech recognition is the exact opposite. After being extracted the sequence of feature vectors F is known and what is wanted is the contional probability $P(W|f)$ of the word sequence $W = w_1 w_2 ... w_m$. Bayes' law gives us.

$$P(W|F) = \frac{P(F|W)P(W)}{P(F)}$$

To be more accurate, the goal of ASR is not to calculate the probability $P(W|F)$ but to find the word sequence W_t with the highest probability.

$$\max_W P(W|F) = \max_W \frac{P(F|W)P(W)}{P(F)} = \max_W P(F|W)P(W)$$

The remaining component $P(W)$ is the probability that a sequence of words will be spoken and is supplied by the language model.

2 Theoretical Background

In this chapter the basic theoretical background knowledge needed to build the proposed adaptive language model is presented. Section 1 covers the basics of language modeling and explains how language models are built and evaluated. The first attempts at building adaptive language models are covered in section 2. Section 3 introduces text classification which is used to build the language models discussed in section 4.

2.1 Basics of Language Modeling

Our goal in language modeling is to create a mathematical model of a given natural language. Natural languages are languages that humans use for general purpose human to human communication These can be spoken, written or even signed. They are made up from a set of symbols (words) which are arranged according to the language's grammar rules to form sentences.

The set of symbols is arbitrarily defined and sometimes ambiguous. The English word *bank* can, depending upon the context, refer to among other things *"a financial institution"*, *"an edge of a river"* or *"the incline of an aircraft*. The same word in German can also mean *"bench"*. Although new words and terms slip into a language all the time, for practical purposes language models are often defined over a finite vocabulary V.

A language model for a particular language calculates how probable a sentence or sequence of words is in that language. In a lot of applications, like speech recognition and data compression, language models are used predictively to calculate the probability of a possible *next word* in a context. For example, if in the English language the context is *"I want to live in a nice"* then the probability of the next word being *house* is a lot higher than the probability of the next word being *mouse*.

Language models often contain special symbols indicating the beginning ($< s >$) and end ($< /s >$) of a sentence. As a word sequence *"$< s >$ I want to live in a nice"* may have a relatively

high probability but its probability as a sentence *"$< s >$ I want to live in a nice $< /s >$"* will be very low.

The probability of a sentence or word sequence can be calculated from the conditional probability of the individual words given their context.

$$P(w_1 w_2 ... w_m) = \prod_{i=1}^{n} P(w_i | w_{i-1}, ..., w_1)$$

The sequence of words $s = w1...w_m$ is called a sentence when w_m is the end of sentence symbol $< /s >$. Both sentences and word sequences have an implied start of sentence symbol $< s >$ at the unwritten w_0 position. Over the set of complete sentences $P(s)$ can be considered a probability distribution.

$$\sum_{s \text{ is a sentence}} P(s) = 1$$

A language model is often said to *score* a sentence or word. Since the word sequence or sentence probabilities are almost always only used to find the most probable word sequences or sentences this comparison can be done faster by comparing the sum of the logarithm of the probabilities (log probabilities).

$$\log P(w_1 w_2 ... w_m) = \sum_{i=1}^{n} \log P(w_i | w_{i-1}, ..., w_1)$$

$$P(s_1) < P(s_2) \Rightarrow \log P(s_1) > \log P(s_2)$$

The log probability returned by such a language model is called a score.

There are two main ways of building language models; the first is data driven, using a large corpus of text to build a statistical model. The other method is to use all the rules and constraints of the language's grammar.

2.1.1 Statistical Language Models (N-grams)

N-gram language models make the assumption that the probability of a word is only dependent on the $n - 1$ previous words. In the context w_1 w_2 ... w_{i-1} the probability of the word w_i occurring is then only dependant on w_{i-1}, w_{i-2}, ..., $w_{i-(n-1)}$. This is written as the conditional probability $p(w_i | w_{i-1}, w_{i-2}, ..., w_{i-(n-1)})$. An n-gram is called a unigram when $n = 1$, a bigram (or sometimes digram) for $n = 2$, a trigram for $n = 3$, four-grams and everything higher don't have special names. The term n-gram can refer to both a language model and a counted sequence of words contained within the language model.

The concept originates from Claude Shannon's work on information theory and data compression. When transferring text through a communications channel the symbol probability is not constant but dependant on the previous symbols. In English, for example, *qu* is more probable than *qj*. In an n-gram language model the symbols are words. So the trigrams in the sentence *"$< s >$ I want to live in a nice house $< /s >$"* are *"$< s >$ I want"*, *"I want to"*, *"want to live"* , ... , *"nice house $< /s >$"*.

Building an n-gram language model doesn't require any knowledge of the language's grammar. All that is needed is a large amount of language data from which the n-gram probabilities are estimated. The n-gram is then said to have been *trained* on that data. This is done by counting all the n-gram and (n-1)-gram word sequences and using a maximum likelihood estimator.

$$P(w_i|w_{i-1}, ..., w_{i-(n-1)}) = \frac{\#w_{i-(n-1)}...w_{i-1}w_i}{\#w_{i-(n-1)}...w_{i-1}}$$

$\#w_x..w_1$ is the number of times the specified sequence of words occurs in the training data. Straightforward n-gram language models like this rely only on n-gram counts found in the training data which can lead to sparsity problems because a lot of possible and likely n-grams will not be contained in the training data.

The non-usage of grammar knowledge is also one of the major criticisms of n-gram language models. This is countered with the observation that with enough data short distance grammatical structures are automatically modeled and, especially in spontaneous speech, grammatical rules are not always adhered to.

Another criticism is that n-grams can't model long distance dependences. The sentence "The *boat* which I just had repainted in blue *sunk*" would have a very low probability in an trigram language model because of the vanishingly low trigram probability of *"in blue sunk"*. N-grams also do not take advantage of the domain context: In a pet store your much more likely to ask to buy a mouse than a house, whereas when talking to a realtor the probabilities will be reversed.

Despite these concerns and problems most state of the art ASR systems and MT system use n-gram based language models. To overcome the data sparsity, these state of the art n-gram language models utilize, among other things, smoothing and discounting methods.

2.1.2 Providing Generalization Capability to N-grams

The training data used to build an n-gram language model will not contain all the possible or even probable n-grams, like the improbable but still perfectly OK previously mentioned trigram *"in*

blue sunk"[1]. Conversely, lots of meaningless and unneeded n-grams will appear once or twice in the training data. **Smoothing** attempts to solve these problems by *smearing* probability mass from seen n-grams to unseen n-grams allowing them to generalize better.

Smoothing, Discounting and Back-Off Techniques

> *"Whenever data sparsity is an issue, smoothing can help performance, and data sparsity is almost always an issue in statistical modeling. In the extreme case where there is so much training data that all parameters can be accurately trained without smoothing, one can almost always expand the model, such as by moving to a higher n-gram model, to achieve improved performance. With more parameters data sparsity becomes an issue again, but with proper smoothing the models are usually more accurate than the original models. Thus, no matter how much data one has, smoothing can almost always help performace, and for a relatively small effort."*
>
> Chen & Goodman (1998)

A simple smoothing technique is Laplace smoothing, which involves increasing the count of every possible n-gram by one and was first used in Language Models by Lidstone and Jeffays [Lid20].

$$P(w_i|w_{i-1},...,w_{i-(n-1)}) = \frac{\#w_{i-(n-1)}...w_{i-1}w_i + 1}{\#w_{i-(n-1)}...w_{i-1} + V}$$

This has the desired effect of allowing sentences with unseen n-grams to be have non zero probabilities but its effects on well estimated, high count n-grams are drastic. If the nonsmooothed bigram probability of $w_q w_u$ was 0.95 in a 100000 word corpus with a 10000 word vocabulary and 500 occurrences of the word w_q then the bigram probability after applying Laplace Smoothing would be

$$P(w_u|w_q) = \frac{\#w_q w_u + 1}{\#w_q + V} = \frac{475 + 1}{500 + 10000} = 0.0453...$$

This shift of probability mass from well estimated n-grams to unseen n-grams results in a poorer language model performance [GC94]. Good-Turing smoothing solves this problem through discounting. The frequencies of seen events are decreased and the unused probability mass is distributed evenly over the unseen n-grams. This technique of counting $r*$ whenever the real count of an n-gram was $r > r*$ is called **discounting**.

[1] Google actually finds 7 pages containing the 3-gram *"in blue sunk"*, but none for *"in violet sunk"*. As of Mai 2009.

But is it desirable to distribute this free probability mass evenly over all unseen n-grams? According to Jelinek and Mercer a better way would be to utilize the probabilities of lower order n-grams. To do this **Jelinek-Mercer Smoothing** interpolates between all the $n-i$-gram ($i = 0...n-1$) language models that can be built from the training data.

$$\begin{aligned} P_{J-M}\left(w_i|w_{i-1}, w_{i-2}, ..., w_{i-(n-1)}\right) = \quad & \mu_n P_n\left(w_i|w_{i-1}, w_{i-2}, ..., w_{i-(n-1)}\right) \\ & +\mu_{n-1} P_{n-1}\left(w_i|w_{i-1}, w_{i-2}, ..., w_{i-(n-2)}\right) \\ & +... + \mu_1 P_1\left(w_i\right) \\ \sum_{j=1}^{n} \mu_j = 1 \end{aligned}$$

The weights μ_i can be estimated using the expectation-maximization (EM) algorithm on either held-out training data or through cross-validation [BJM90].

Witten-Bell smoothing improves on Jelinek-Mercer smoothing by replacing the static μ_i mixing weights with dynamic context dependent weights which can interpreted as a confidence value of the higher-order n-gram. These new $\mu_j(w_{i-(j-1)}, ..., w_i)$ are set depending upon how often the n-gram $w_{i-(j-1)}, ..., w_i$ was seen in the training data. When its occurrence was high then the original maximum likehood estimate will be good and more weight can be given to the higher order n-gram, otherwise the lower order n-grams will receive more weight [WB89].

In contrast to the interpolation smoothing methods, back-off smoothing techniques like **Katz Smoothing** only use lower order (n-1)-grams to estimate the probability of n-grams with zero counts, n-grams with nonzero counts are still discounted to free probability mass for the unseen n-grams [Kat87]. Let $r = \#w_{i-(n-1)}...w_{i-1}w_i$ be the count of a word sequence then its discounted **Katz Count** c_{katz} is:

$$c_{katz}(w_{i-(n-1)}...w_{i-1}w_i) = \begin{cases} r & r \geq k \\ d_r r & k > r > 0 \\ \alpha\left(w_{i-1}, ..\right) P_{katz}\left(w_{i-1}|...\right) & \text{otherwise} \end{cases}$$

where d_r and the back-off probabilities $\alpha\left(w_{i-1}, ..\right)$ are chosen such that no probability mass is added or lost.

$$\sum_{s \text{is a sentence}} P_{katz}(s) = 1$$

The new recursively defined katz-smoothed probability function is derived from the Katz Counts.

$$P_{katz}\left(w_i|w_{i-1}, ..., w_{i-(n-1)}\right) = \frac{c_{katz}(w_{i-(n-1)}...w_{i-1}w_i)}{\sum_{w_i} c_{katz}(w_{i-(n-1)}...w_{i-1})}$$

What Katz smoothing fails to take into consideration is that words like *"Francisco"* might have a very high unigram probability. There are only a very limited amount of words that they occur after. *"Francisco"* is almost always seen after *"San"* [KN95]. **Modified Kneser-Ney Smoothing** takes this phenomenon into consideration when calculating the back-off probabilities.

An indepth evaluation and empirical study of smoothing techiques for statistical language models carried out be Chen and Goodman concluded that modified Kneser-Ney smoothing performed best [CG96].

Automatically Finding more Relevant Training Data

Building a statistical language model for a particular domain can be very difficult if there is only a small amount of training data available. Sarikaya, Gravano and Gao designed a system that combines the small in-domain language model (D-LM) with a language model built on text acquired from the World Wide Web (Web-LM) [SG05]. Only text that was *similar* to training data was included in the Web-LM. Using this method they were able to reduce the WER from a baseline of 24.3% to 19.1%.

A similar system was built by Sethy, Georgiou and Narayanan who started with a large general purpose topic independent language model (G-LM) and a small topic dependant language model (T-LM) which they used to generate search queries [SGN05]. The retreaded text data was weighted through relative entropy and either incorporated into the T-LM or G-LM, or used to build a rejection language model. Their system achieved a WER of 24% on a test set of 800 medical domain utterances, a relative reduction of 14% compared to the baseline 28% WER.

While both systems were able to improve their small baseline language models by intelligently adding new data, neither of them compared their small in-domain language model plus intelligent web data to a large in-domain language model. These systems also have the disadvantages of having to know the domain in advance and requiring a seed amount of data.

2.1.3 CFG Language Models

The alternative to statistical language modeling is knowledge based language modeling where a language's grammar rules are used to build the model. Since context sensitive grammars are very computationally hard to model, knowledge based language models often approximated them with context free grammars (CFG).

CFG language models are successfully used in command interpreters and dialog systems where only simple sentences are encountered. They are not useful when working with continuous speech.

2.1.4 Language Model Evaluation

Because language models are used in a lot different tasks, they can be evaluated in many different ways. A language model used in a machine translation system could be evaluated with the Bilingual Evaluation Understudy (BLEU) metric[cite Papineni, K., Roukos, S., Ward, T., and Zhu, W. J]. In the field of information retrieval a language model might be evaluated using the F-measure.

In ASR the standard evaluation metric is the word error rate (WER[2]), which is the percentage of word errors in the hypothosis sentence compared to the reference sentence [cite Hunt, M.J., 1990: Figures of Merit for Assessing Connected Word Recognisers]. The number of errors is the minimal edit distance between the hypothosis and the reference. This is the minimal amount of substitutions s, insertions i and deletions d needed to transform the hypothosis into the reference sentence with length l.

$$\text{WER} = \frac{s + i + d}{l} \times 100\% \tag{2.1}$$

Most of these task based evaluation metrics have the disadvantage of not only measuring how good your language model is but also how also how good the other components are. Some of them are also very computationally intensive and time consuming.

A general-purpose task independent language model evaluation metric is *perplexity* which is defined as $2^{H}_{\rho}(T)$, where $H_\rho(T)$ is the *cross-entropy* of the language model on a set of test sentences T, containing $|T|_w$ words.

$$H_\rho(T) = \frac{\sum_{t \in T} \log_2 P(t)}{|T|_w} \tag{2.2}$$

$$\text{ppl} = 2^{H_\rho(T)} \tag{2.3}$$

If the set of test sentences T is representative of the language or domain the language model is designed to model then the perplexity metric can be used to compare different language models.

[2] Although WER is given in percent, because it is in percent of the reference sentence length l, a very bad hypothosis with more than l words, could have a WER of over 100%

Lower perplexity values indicate that a language model is more predictive. It has also been shown that lower perplexity values correlate with lower word error rates [KP02].

2.2 General Techniques and Related work in Statistical Language Model Adaptation

2.2.1 Cache Based Language Models

The first adaptive language model was a cache-based language model designed by Kuhn, De Mori, McGill [KDMUoCS90]. A cache-based language model simply increases the probability of a word whenever it is observed, assuming that words that appear once are more likely to appear again. To cope with a change of speaker or topic the probability increase from observing a word is not permanent. A cache-based language model is composed of two weighted probabilities, the cache part and the n-gram part.

$$\begin{aligned} P_{lm}\left(w_i|w_{i-1}, w_{i-2}, ..., w_1\right) =& \gamma P_{cache}\left(w_i|w_{i-1}, w_{i-2}, ..., w_{i-K}\right) + \\ & (1-\gamma) P_{ngram}\left(w_i|w_{i-1}, ..., w_{i-n+1}\right) \end{aligned}$$

The most basic cache function simply keeps a list of the last K words and counts how often the current word occurs in that list.

$$P_{cache}\left(w_i|w_{i-1}, w_{i-2}, ..., w_1\right) = \frac{1}{K}\sum_{j=1}^{K} I\left(w_i = w_j\right)$$

I is an indicator function, with $I = 1$ when $w_i = w_j$ otherwise $I = 0$. This function has a sharp edge, in that the last K words are all considered equally important and word w_{k+1} is not considered at all.

To more accurately model the observation that, *the more recent an occurrence, the higher the probability of a re-occurrence*, Clarkson and Robinson added a decaying factor to the cache function [CR97].

$$P_{cache}\left(w_i|w_{i-1}, w_{i-2}, ..., w_1\right) = \beta \sum_{j=1}^{K} I\left(w_i = w_j\right) e^{\alpha(i-j)}$$

α is the rate of decay and β a normalization constant such that $P_{cache}\left(w_i|w_{i-1},...\right)$ summed over all the words w_i in the dictionary D equals 1. This also removes the cache size as a variable since words far back in the history have an almost negligible impact on the probability. Tests showed that the regular cache function performed best with a cache size of 500, reducing the perplexity to 144.73 from a baseline of 165.52. With an optimum decay rate of 0.005 the exponentially decaying cache had an even lower perplexity of 141.75.

The cache-based language models have the advantage of not requiring any more data than normal language models to train but then can only improve the probabilities of detecting previously seen words. A cache based language model that observes the word *Obama* will correctly increase the probability of detecting *Obama* again but it will not change the probability of detecting *Barack*[3]. To take advantage of these, other adaptive language models incorporate text classification techniques.

2.2.2 Class-Based Language Models

Class based n-gram language models, first introduced by Brown et al [BMDPL92], attempt to improve the generalization capabilities of language models by mapping words to word classes. The word *"seven"* for example would be mapped to the class *"numbers"* and the word ‘*'poodle"* might be mapped to class *"animals"*. Word probabilities are then dependant on the previous $n-1$ classes and its occurrence frequency within the word class. The word *"seven"* will undoubtably occur more often within the class *"numbers"* than the word *"57"*.

$$P_{class}\left(w_i|c_{i-1},c_{i-2},...,c_{i-(n-1)}\right) = P(w_i|c_i)P_{class}\left(c_i|c_{i-1},c_{i-2},...,c_{i-(n-1)}\right)$$

Class based languages models don't need as much training data as standard n-gram language models and can encode semantic information in the classes which improves speech understanding. In large vocabulary ASR systems cache-based n-grams have failed to reduce the word error rate.

2.3 Techniques and Metrics for text Classification

The basic text classification task consists of assigning a category c_i from a set of predefined categories $\mathcal{C}$ to each document d_i in a set of n documents $\mathcal{D}$. The features of d_i which are used in the

[3]Actually, because the probability density remains the same, by increasing the probability of *Obama* the probability of all words including *Barack* is reduced by a tiny amount

classification depends on the classifier used.

$$\tau : \mathcal{D} \rightarrow c_i \quad c_i \in \mathcal{C}$$

This definition classifies each document into exactly one category and is analogous to the real world task of putting the physical document into a particular file (aka non overlapping categories or single-label case) [SM86].

For documents that do not fit into any of the predefined categories, an off-topic category c_x can be added. To allow documents to be classified into multiple categories (aka overlapping categories or multilabel case) the definition can be modified to:

$$\tau : \mathcal{D} \times \mathcal{C} \rightarrow \{0, 1\}$$

If $\tau\left(d_j, c_i\right) = 1$ then the document d_j is in the category c_j.

2.3.1 TFIDF-Metric

Depending upon the application, the documents will have lots of properties that can be useful in deciding how to categorise them. Traditional documents will have an author (or authors), a title, a publication date and so on. Some documents will also contain subheadings, references and links. By far the most important document property is its body text.

Most text classifiers extract the features used for classification exclusively out of a document's body of text. A standard method of of generating a feature vector $\overrightarrow{f_j}$ from a document d_j is to first extract a set of n terms from the sum of the text of all the documents and then weight them according to their occurrence in d_j. For each d_j we have a

$$\overrightarrow{f_j} = (f_{1,j}, f_{2,j}, ..., f_{n,j})$$

where $f_{k,j}$ donates the weight of term k in document j.

The easiest and most common way to extract terms from a body of text is to view each word as a term. This method is often called the *bag-of-words* view of a document.

Once a set of terms $|\mathcal{T}| = n$ has been decided upon then a TFIDF function can be used to generate a document feature vector $\overrightarrow{f_j}$. In its basic variant the function calculates each component of $\overrightarrow{f_j}$ from its term frequency $\mathrm{TF}_{k,j}$ in j and the inverse document frequency of TF_k of k. The term frequency component measures how often a word occurs in a document.

$$\mathrm{TF}_{k,j} = \#(t_k, d_j)$$

The inverse document frequency measures how discriminative a word is. Words in few documents have a high inverse document frequency and words in lot of documents have a low inverse document frequency [SB87].

$$\text{IDF}_{k,j} = log\left(\frac{|\mathcal{D}|}{\#(\mathcal{D}, t_k)}\right)$$

The logarithm of the quotient is used to blunt the effect of extremely rare words, which might only appear in one or two documents. Putting these together gives us.

$$f_{k,j} = \text{TFIDF}(t_k, d_j) = \#(t_k, d_j) \cdot log\left(\frac{|\mathcal{D}|}{\#(\mathcal{D}, t_k)}\right) \tag{2.4}$$

with $\#(t_k, d_j)$ being the number of times t_k appears in d_j and $\#(\mathcal{D}, t_k)$ the number of documents containing t_k. As is, the TFIDF function does not take into account the length of a document. A term appearing once in a short document is more relevant than if it were to appear in a longer one.

One way to solve this is to normalize the vector generated by the TFIDF function.

$$f_{k,j} = \frac{\text{TFIDF}(t_k, d_j)}{\sqrt{\sum_{h=1}^{|\mathcal{T}|} \text{TFIDF}(t_h, d_j)^2}} \tag{2.5}$$

2.3.2 Cosine Similarity Metric

TFIDF vectors are built to be able to compare documents with each other. This requires a similarity metric. The cosine similarity metric is an easy and fast metric to compute. It is defined by the angel τ between the two vectors f_1 and f_2 that are to be compared.

$$\text{cossim}(f_1, f_2) = \cos(\tau)\frac{f_1 \cdot f_2}{|f_1||f_2|} \tag{2.6}$$

Since the TFIDF vectors are often already normalized ($|f_1| = 1$ and $|f_2| = 1$) the denominator part of this definition can be ignored. Also most TFIDF vectors are sparse, leading to very few nonzero terms in the numerator of the definition. This allows the cossine similarity metric to be calculated very fast.

2.4 Adaptive Language Models using Text Classification

As has been previously pointed out, ASR systems usually perform best in a small domain where the total number of possible words is limited. A specialized ASR system designed for either medical transcriptions or tourist information will perform better than an equivalent but more general one that has to work in both domains.

One method for improving language models that follows from this observation is to have specialist LMs for each domain or topic. This is exactly what Lane, Kawahara and Matsui did. They built an adaptive language model consisting of a Generalized Language Model (G-LM) and multiple Topic-dependent Language Models (TD-LM), which when tested on the ATR phrasebook corpus reduces the WER from a baseline of 8.54% to 7.64% and the perplexity from 22.77 to 16.85 [LKM03].

The baseline was a trigram LM trained on the entire training set and later used as the G-LM part of the adaptive LM. The TD-LMs are each trained on the part of the corpus associated with their particular topic. During decoding the G-LM performs an initial pass, this is then used by a topic classifier to determine which TD-LM to use for the second pass. The authors also propose an extension to this system where an intermediate layer of language models is inserted between the G-LM (layer 1) and the TD-LM (layer 3). The layer 2 LMs cover several related topics and are easier to classify.

Compared to cache based language models this adaptive language model has the advantage that the probabilities of the, for the decoder so far, unseen word sequences can be adjusted depending on the dectected context. Unfortunately the number of topics is very low and each topic has to be manually initialized.

Unlike Lane et al's adaptive language model, the Topic-Based Language Model built by Gildea and Hofmann did not require labelled topics. Instead they used the Expectation-Maximization (EM) algorithm to determine both the probability $P(t|d)$ that a particular document d belonged to a latent topic t and the probability $P(w|t)$ that a word w is contained in the latent topic t [GH99]. Their model assumed that

$$P(w|h) = \sum_t P(w|t)P(t|h) \tag{2.7}$$

and interpolated it with an n-gram model to take advantage of short range structures. Unfortunately an improvement in perplexity of 17% did not translate into an improvement in WER when tested on the CSR Hub-4 corpus and broadcast news.

3 Data Sources and Preparation

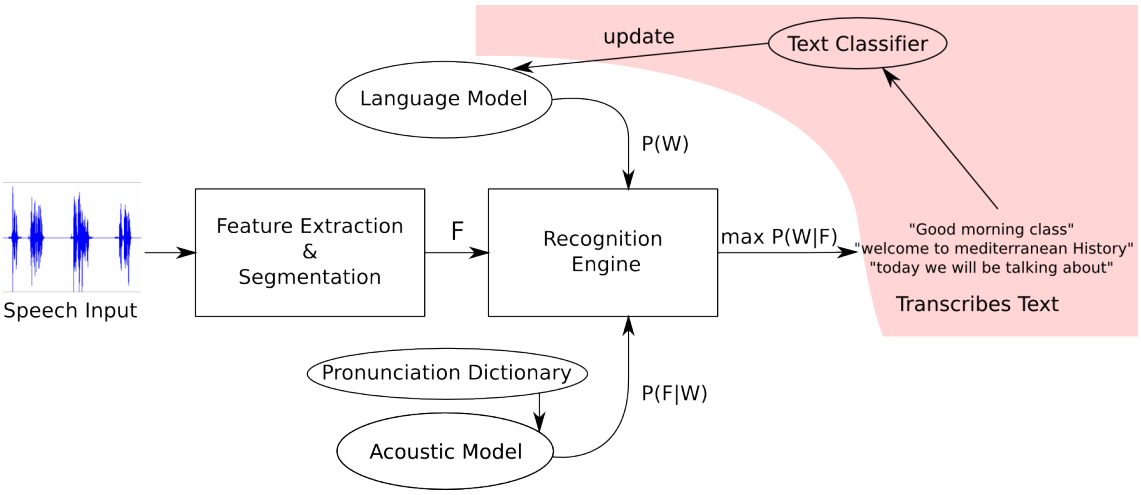

Figure 3.1: Modified Speech Recognition System using a text classification to adapt the language model.

The Speech Recognition System schematic shown in figure 1.5 can be modified (figure 3.1) to work with an adaptive language model by analyzing the text output and using it to adapt the language model. One way of doing this is to identify the domain in which the language model is working and interpolate the original base language model P_{base} with a language model built specifically for the detected domain P_d. Let $h = w_1 w_2 ... w_(i-1)$ be the current word history.

$$P_{adapt}(w_i|h) = \mu P_{base}(w_i|h) + (1-\mu) P_d(w_i|h) \tag{3.1}$$

In practice the text classifier might not be able to accurately identify the domain of the text spoken so far or it might by possible to classify it into two more domains. So let the classifier instead return the n most similar domains, also referred to as *concepts*, $c_1 ... c_n$ and interpolate them based on their similarity weights $\lambda_1 ... \lambda_n$.

$$P_d(w_i|h) = \sum_{j=1}^{n} \lambda_j P_{c_j}(w_i|h) \tag{3.2}$$

where

$$\sum_{j=1}^{n} \lambda_j = 1$$

The big open question here is: Where do these concept language models come from and how are they built? This chapter explains how freely available structured data sources can be used to generate a large set of domain or topic specific language models. Section 1 describes how Evgeniy Gabrilovich had to solve a similar problem when incorporating world knowledge into a textual information retrieval system. Section 2 elaborates on two of the data sources Gabrilovich used and what can be extracted from them to build the concept language models. The preprocessing and *cleaning* of the extracted data is explained in section 3. The final section describes how the concept language models were built and which concepts are used in the final model.

3.1 Feature Generation using World Knowledge

Evgeniy Gabrilovich wanted to design a textual information retrieval system that would recognize that news headlines like ‘*“Demand for Viagra on the Rise”* and *“Pfizer Profits Soar”* were related; or realize that a keynote speech by Steve Balmer might have something to do with Microsoft. In his PhD thesis *“Feature generation for textual information retrieval using world knowledge”* he suggested augmenting the traditional features used in text classification with features generated from external data sources like wikipedia or the open directory project [Gab07].

Numerous concepts were extracted from these data sources. A peace of text to be classified in the information retrieval system was first preclassified and associated with a several concepts which become the additional feature in further classification. This mapping of text into a vector space of concepts is called **explicit semantic analysis**.

3.2 Data Sources

3.2.1 Open Directory Project

The *Open Directory Project* ODP is a collection of over 4.500.000[1] links which have been sorted into over 500.000 relevant categories by human editors. It has a hierarchical structure like the

[1] as of march 09

Figure 3.2: Exert of the ODP XML file.

Yahoo directory. Volunteer editors extend and maintain the directory. The raw data of the ODP can be downloaded from http://www.dmoz.org as a large XML file.

The ODP XML file contains two main top level tags; Topic and ExternalPage (see figure 3.2). The Topic tag's main attribute is its ID (e.g. *Top/Computers/Programming/Languages/Haskell*) and each ExternalPage tag contains a link to a website and is associated with a topic. Topics and ExternalPages are arranged in a tree structure with the topic "Top" as the root node and the external pages as the leaves. The terms topic and node will be used interchangeably. A topic's direct text is the text contained in all the websites linked the ExternalPage tags' directly under it. The text associated with a topic is its direct text combined with the direct text of all its sub topics.

The text data is retrieved with the help of a two part program. The *server thread* part of the program reads the ODF XML file and extracts the links while the numerous *client* threads take *parcels* of these links from the server thread and download the websites contents. Nodes with more than an arbitrarily set amount of associated text are used as concepts.

3.2.2 Wikipedia

The free online collaborative encyclopedia *Wikipedia* contains over 2.500.000[2] articles in the English language. Unlike the ODP, Wikipedia is flat containing no strict hierarchy between the articles. Instead its link structure is exploited. Wikipedia provides regularly updated database dumps that anybody can download. They can be downloaded from http://download.wikimedia.org/ as

[2] also as of march 09

very large XML files. Again a minimum size limit is chosen and all larger articles are used as concepts. Their associated text is a combination of their article text plus the texts of linked-to articles.

3.3 Text Cleaning

The text extracted from the ODP links are not raw text files but instead html files which contain a lot of unwanted information. Before anything else can be done the html tags have to be stripped. The same thing is done to the wikicode found in the text taken from Wikipedia[3]. After removing these non-text elements the text is split into sentences [Clo01] and finally all punctuation marks are removed. Figure 3.3 show an excert of an html file linked to by the ODP and figure 3.4 shows the same file after it has been *cleaned*.

3.3.1 Building the Language Model

Two things have to be built for each concept; a language model and an attribute vector. With the help of the *SRI Language Modeling Toolkit* ([Sto02]) and the methods discussed in chapter 2.1 a smoothed n-gram language model is built for each concept.

The attribute vectors are used to compare the concept language models to the word sequences that the language model has to adapt to. The attributes (words) are extracted from the sum of all the text associated with concepts. The high frequency stop-words[4] are removed as well as words with a very low frequency. The concept attribute vectors can then be built using the TFIDF function 2.4.

[3] Wikipedia's recursively defined templates make this very hard.

[4] stop-words are semantically meaningless words like: the, and, ...

```
</td></tr>
<tr>
<td> <a href="/haskellwiki/Merchandise" title="Merchandise">Merchandise</a>

</td></tr>
<tr>
<td> <a href="/haskellwiki/Haskell.org" title="Haskell.org">haskell.org</a>
</td></tr>
<tr>
<td> <a href="/haskellwiki/HaskellWiki:Contributing"
title="HaskellWiki:Contributing">Contributing to this site</a>
</td></tr>
<tr>
<td> Languages: <strong>en</strong> <a href="/haskellwiki/Es/Haskell" title="Es/Haskell">es</
a> <a href="/haskellwiki/Ro/Haskell" title="Ro/Haskell">ro</a> <a href="/haskellwiki/Pt/
Haskell" title="Pt/Haskell">pt</a> <a href="/haskellwiki/Fr/Haskell" title="Fr/Haskell">fr</
a> <a href="/haskellwiki/Ru/Haskell" title="Ru/Haskell"> ru</a> <a href="/haskellwiki/Cn/
Haskell" title="Cn/Haskell">zh/cn</a> <a href="/haskellwiki/Tur/Haskell" title="Tur/
Haskell">tur</a> <a href="/haskellwiki/Cz/Haskell" title="Cz/Haskell">cz</a> <a href="/
haskellwiki/Ko/Haskell" title="Ko/Haskell">ko</a>

</td></tr>
</table>
</td><td valign="top" width="75%" style="text-align:left; line-height:120%">
<p style="text-align:left">
Haskell is an advanced <a href="/haskellwiki/Functional_programming" title="Functional
programming">purely functional
programming</a> language. An open source product of more than twenty years of cutting edge
research,
it allows rapid development of robust, concise, correct
software. With strong support for <a href="http://www.cse.unsw.edu.au/~chak/haskell/ffi/"
class='external text' title="http://www.cse.unsw.edu.au/~chak/haskell/ffi/"
rel="nofollow">integration with other languages</a>,
built-in <a href="http://haskell.org/haskellwiki/Applications_and_libraries/
Concurrency_and_parallelism" class='external text' title="http://haskell.org/haskellwiki/
Applications and libraries/Concurrency and parallelism" rel="nofollow">concurrency and
parallelism</a>, debuggers, profilers, <a href="http://hackage.haskell.org/packages/
hackage.html" class='external text' title="http://hackage.haskell.org/packages/hackage.html"
rel="nofollow">rich libraries</a> and an active community, Haskell makes it easier to produce
flexible, maintainable
high-quality software.
</p>
<table border="0" cellspacing="0" cellpadding="15">

<tr>
<td valign="top" width="65%" style="text-align:left">
<p><b>Events</b>

```

Figure 3.3: A *dirty* html file before cleaning and sentence splitting.

```
Merchandise
haskell.org
Contributing to this site
Languages en es ro pt fr ru zh/cn tur cz ko
Haskell is an advanced purely functional programming language
An open source product of more than twenty years of cutting edge research it allows rapid
development of robust concise correct software
With strong support for integration with other languages built-in concurrency and parallelism
debuggers profilers rich libraries and an active community Haskell makes it easier to produce
flexible, maintainable high-quality software
Events
```

Figure 3.4: The text file shown in figure 3.3 after cleaning.

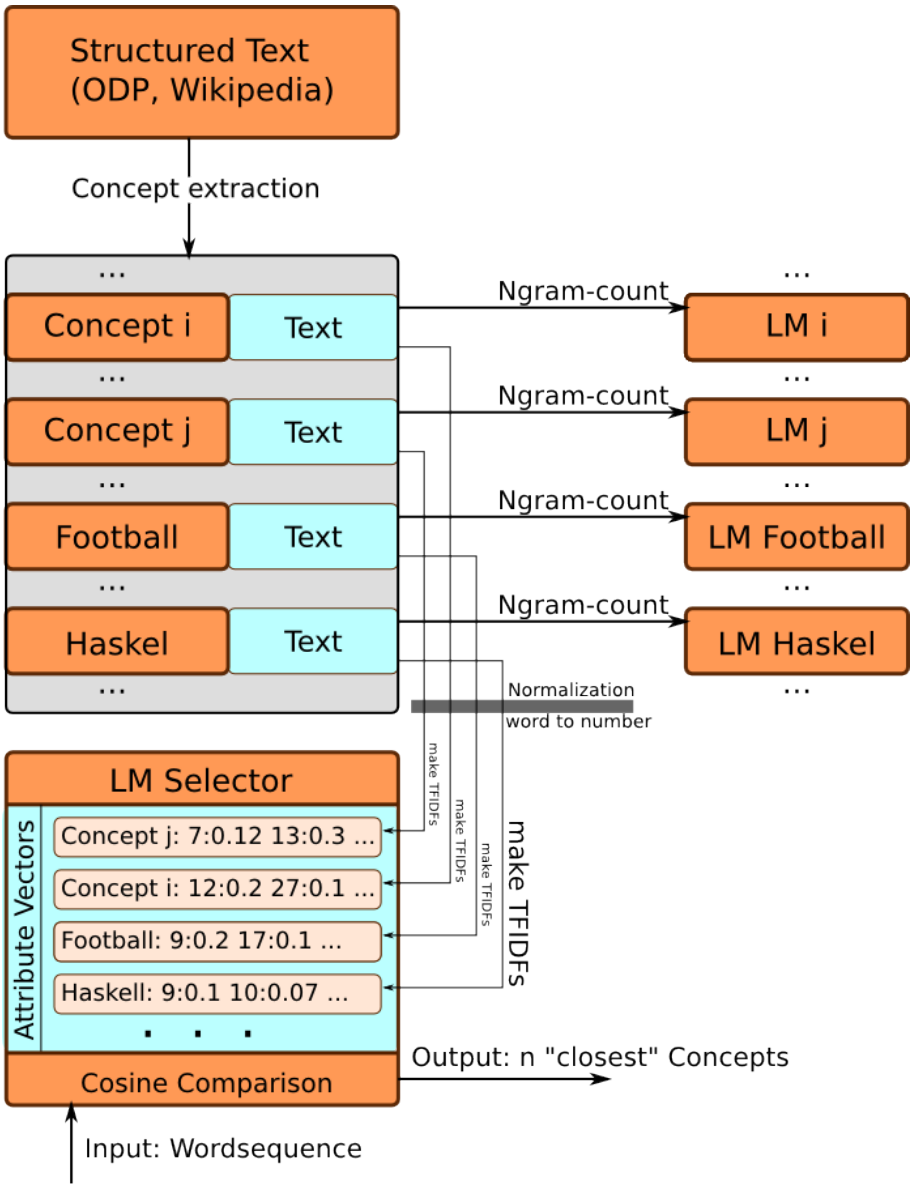

Figure 3.5: This figure shows how the adaptive language model is built. The concepts with their associated text are extracted from the data source(s). The SRI Language Modeling Toolkit is used to build concept n-grams from that text. A concept TFIDF attribute vector for use in the Selector is also built from the associated text.

4 Language Model implementation and incorporation into a decoder

4.1 Overview of the previously existing main components

4.1.1 The Janus Recognition Toolkit

The Janus Recognition Toolkit (JRTk or just Janus) is a general purpose speech recognition toolkit developed at the Interactive Systems Labs in Karlsruhe, Germany and Pittsburgh, USA. Although implemented in C, it includes an object oriented Tcl/Tk interface. Complex tasks are performed with Janus by writing a Tcl/Tk script and running it in Janus. Everything necessary for ASR, from codebooks over language models to the decoder itself is handled as objects in the Tcl/Tk script.

Ibis Decoder

The heart of the JRTk is the Ibis decoder, [cite soltau hagen] a single pass decoder that uses linguistic context polymorphism. This allows it to utilize all available language model information as early as possible. The **dictionary** object provides the decoder with mappings from phone sequences to words. These mappings are not unique because some words can be pronounced in multiple different ways and the same phone sequence can correspond to different words at different times depending upon the context (e.g. there and their). The **SVocab** object defines a subset of the words in the dictionary as the search vocabulary. Only words in the search vocabulary can be recognized by the decoder and appear in the output hypothesis. The words in the search vocabulary are mapped to corresponding words in the Language Model. Since the search tree (**STree**) requires the inverse of this mapping to be surjective, the **SVMap** object maps all words in the search vocabulary that do not have corresponding words in the language model to the *unknown* LM word. The scores of the acoustic model and the language model are combined in the search tree and it is from here that the language model is normally accessed. Instead of just outputting

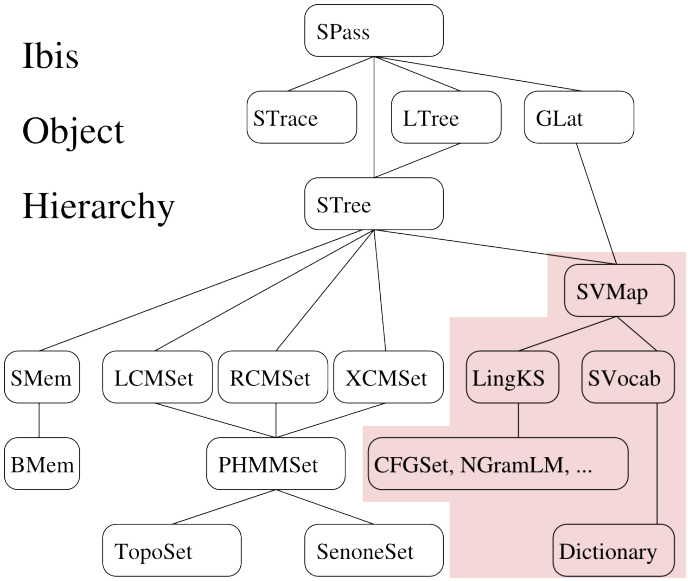

Figure 4.1: Ibis decoder object hierarchy and structure. The red area was changed to make use of the new SelectLM (see figure 4.5)

the best hypotheses, the Ibis decoder can also return a word lattice (**GLat**, see Figure 4.2) that can later be rescored without redoing the entire decoding.

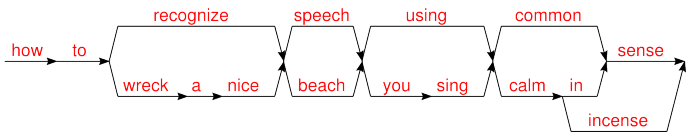

Figure 4.2: An example word lattice.

Language Model Interface

To easily be able to change language models, all language model objects in Janus, often referred to as linguistic knowledge sources (**LingKS**) implement a common interface. Its main functions are: [cite Janus Manuel and efficient handling]

LingKS.createLCT(): returns a linguistic context, containing only the start of sentence word.

LingKS.scoreArray(LCT): scores all the words in the language model at the given linguistic context and returns them in an array.

LingKS.extendLCT(LCT,LVX): creates and returns a new linguistic context by extending the given existing linguistic context by word.

LingKS.score(LCT,LVX): scores a word in a linguistic context.

The linguistic context (**LCT**) is what the language model knows of its place in a sentence. For an n-gram the linguistic context would be the previous n-1 words and for a CFG it would be its current state. In these functions the term LVX refers to a word in the language model vocabulary.

4.2 Structure of the Adaptive Language Model Framework

The Adaptive Language Model Framework was built in two stages and consists primarily of two separate Objects, the Selector, a stand alone program and the SelectLM Janus object as well as a collection of scripts and programs used to extract and prepare the data. The Selector loads a list of attribute vectors corresponding to concepts with previously built sub language models and responds to queries consisting of word sequences with the n most appropriate language models and their mixing weights. The SelectLM Janus object implements the standard language model interface, connects all the sub language models as well as a base language model together and interpolates their scores based on the weights returned by the Selector. The base language model (BaseLM) is a general purpose language model trained on a large data set.

4.3 Selector

The Selector was originally only implemented as a stand alone program to help detect bugs but due to its large memory requirements it was often necessary to run it on a separate machine from the one running the modified Janus. Since doing this slows down decoding considerably, future versions of the Adaptive Language Model Framework will probably incorporate the Selector into Janus as an object.

4.3.1 Overview

The initialization of the Selector requires a port to listen on and two files, an attribute vector list file and the dictionary file used to build the attribute vectors. The vector in line j of the vector list file corresponds to concept j and the sub language model j in the SelectLM object. The dictionary

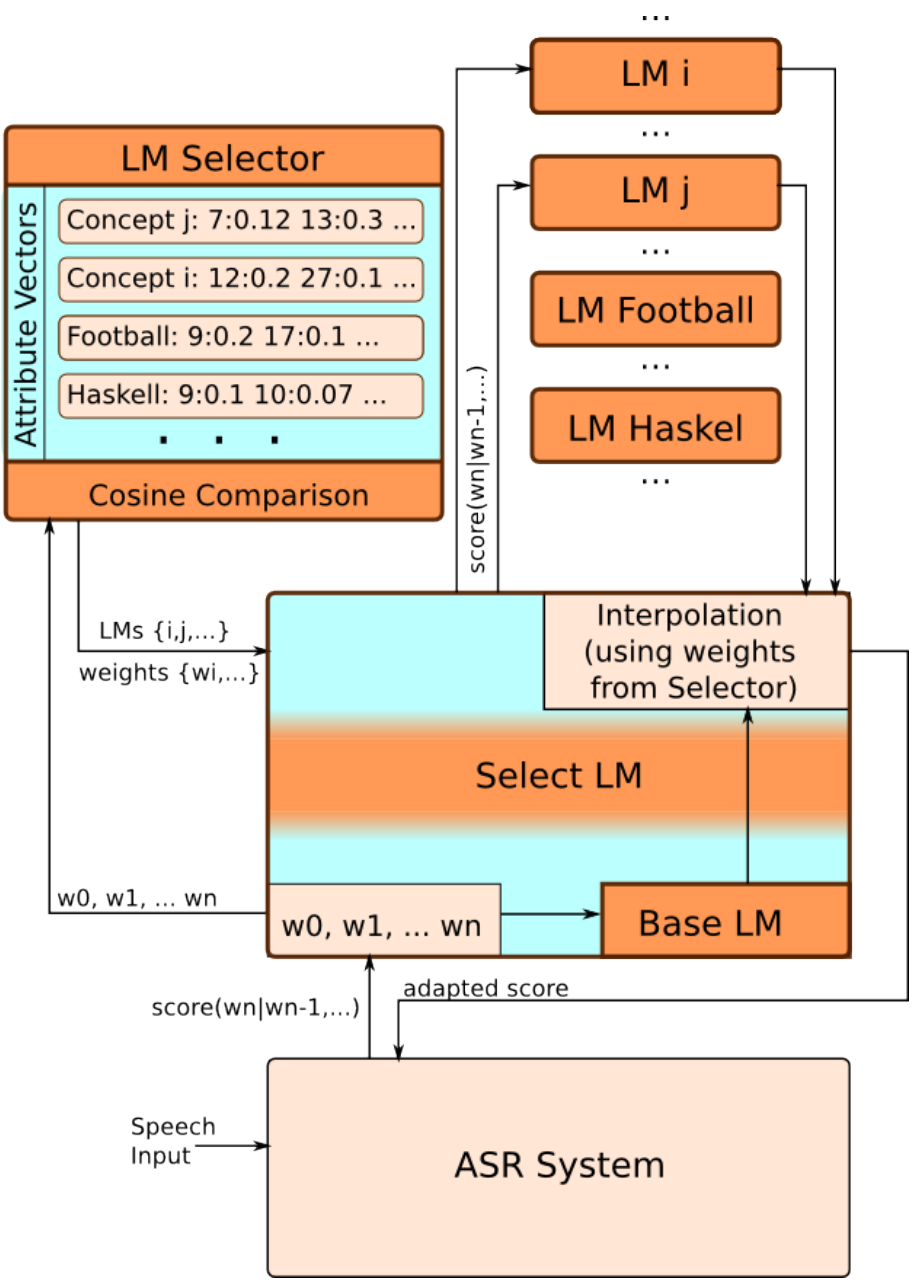

Figure 4.3: Adaptive Language Model in use

ID	Weight	Concept
409	0.184406	Regional/Europe/United_Kingdom/Scotland/Recreation_and_Sports/Golf
123	0.194006	Regional/Europe/United_Kingdom/Scotland/Aberdeenshire/Society_and_Culture/History
360	0.196899	Regional/Europe/United_Kingdom/Scotland/Moray/Glenlivet/Travel_and_Tourism
473	0.210764	Regional/Europe/United_Kingdom/Scotland/Travel_and_Tourism
417	0.213925	Regional/Europe/United_Kingdom/Scotland

Table 4.1: The top 5 concepts returned by the Selector to the query: *"this summer the world will be coming to scotland i hope a million people to come to scotland and behave themselves"*.

file is used to build an attribute vector out of a word sequence. After the dictionary and attribute vectors have been loaded a specified port is opened and the selector waits for connections from Janus.

4.3.2 Query Response

An incoming query to the Selector is a sequence of words which is converted to an attribute vector k. Using the cosine similarity metric the n concepts with loaded attribute vectors most similar to k are found.

$$\text{top}^n = \{s_{t_1}, ..., s_{t_n}\} = \text{max}^n_{v_i \in V}\{\frac{k \cdot v_i}{|k||v_i|}\} \tag{4.1}$$

V refers to the set of all loaded attribute vectors, $t_1, ..., t_n$ the n concepts with the highest similarities $s_{t_1}, ..., s_{t_n}$. Since the term $|k|$ is constant, it is not required to compare the similarities and can be omitted.

$$\text{top}^n = \{\hat{s_{t_1}}, ..., \hat{s}_{t_n}\} = \text{max}^n_{v_i \in V}\{\frac{k \cdot v_i}{|v_i|}\} \tag{4.2}$$

The weights are calculated by normalizing the similarities $\hat{s}_{t_1}, ..., \hat{s}_{t_n}$ of the top n concepts so that $w_{t_1} + ... + w_{t_n} = 1$. The Selects responds to the query by sending back the top n concepts and their weights.

4.4 SelectLM

SelectLM is a new Janus object implementing the language model interface. It reuses and is based open the InterpolLM written by Christian Fügen, which interpolates a set of language models. As

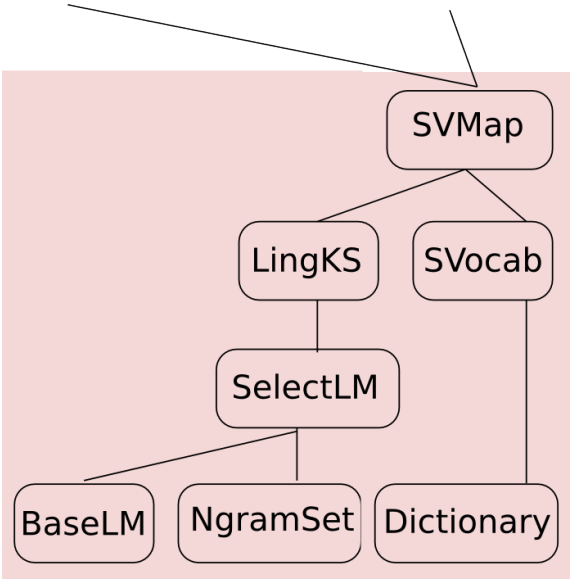

Figure 4.4: New Ibis

can be seen in figure 4.5 the SelectLM also uses the language model interface to communicate with the baseLM and the set of concept associated language models, referred to as sub language models. It also acts as vocabulary mapper from its internal vocabulary to the vocabulary of its sub language models.

4.4.1 SelectLM Description

The SelectLM object is initialized by separately loading all the sub language models and connecting them to the SelectLM object, which builds up the vocabulary mapping table. The vocabulary of the SelectLM is the union of all its sub language model vocabularies. Words in the SelectLM vocabulary that are not contained in a particular sub language model are mapped to that language model's unknown word. This mapping table is necessary because of the way the n-gram language model stores the scores. They are stored in memory as a large array which uses the wordnumber as its index. If all the wordnumbers were the same in all sub language models then they would all take up the same amount of space in memory as the largest one. The location, hostname and port of the Selector and how often it is queried also have to be configured. Before running the decoder the starting context, to which the language model should adapt, is set. This provides 5 adaptation strategies.

Adapt to history The m previously decoded hypotheses are loaded into the language models starting context and the Selector is queried once per hypothesis.

Adapt to base hypothesis The baseLM is used to generate a hypothesis which is loaded into the starting context for a second decoding pass using.

Adapt to base hypothesis + history The m previously decoded hypotheses and the hypothesis generated by the baseLM on the first pass are used as the starting context.

On the fly The starting context is left empty and the Selector the queried every time the a linguistic context is extended.

On the fly + history Here again the m previously decoded hypotheses are used as the starting context and the Selector is queried every time a linguistic context is extended.

4.4.2 SelectLM Data Structure

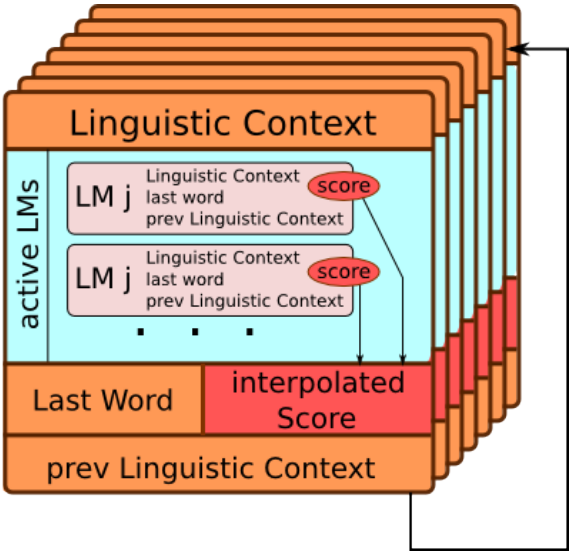

Figure 4.5: The data structure of the SelectLM linguistic context.

A large part of the SelectLM Data Structure is its internal vocabulary mapper. Instead of dealing with strings and words directly the decoder uses numbers correspond to words. Language models take advantage of this by storing the word scores in such a manner that this number can act as an index. Unfortunately, language models with different vocabularies will have a different number to word mapping. This is why the SelectLM object needs to have both a globally defined number to

word mapping and a large look-up table to find a word's number in a particular concept language model.

The other big part is the linguistic context set. While decoding a hypothesis the Ibis decoder scores lots of different hypotheses and can come to them through different paths (see figure4.2). Each extention of a part-hypothesis by a further word creates a new linguistic contex containing all the information need to return a context dependant word score should the decoder require it. For the SelectLM that means the concept language models used in this context and their mixing weights.

5 Experiments

Speech recognition tests to measure possible WER improvements were performed on a subset of the 2006 TC-STAR development data which is composed of European Parliament plenary speeches. This chapter explains how the adaptive language model was used in the ASR system designed for the 2006 TC-STAR ASR evaluation. The first two sections introduce the 2006 TC-STAR development data and the baselines used. The next section explores how the different ways of building an adaptive language model affect its performance. In section 4 the various adaptation methods are analyzed and compared. The following section discusses the interpolation weights used at different stages and the final section shows how the best system performed on the evaluation subset.

5.1 Test and Evaluation Data

The 2006 TC-STAR development data consists of 901 utterances spoken by multiple people. It is derived from recordings of European Parliament plenary speeches. Before any components were tested the data was split into two subsets: a small development set containing 144 utterances and a large evaluation subset containing the remaining utterances. All parameter tests were performed on the small development set and only the language model with the best set of parameters was run on the evaluation set.

5.2 Establishing a Baseline

Two different baselines are needed to measure the effect of the adaptive language model in different situations. **Base1** is a general purpose 3-gram language model not tuned to any specific domain. **Base2** is original hand tuned 4-gram language model designed to work best on the testset domain. Base1 was built from the sum of all the text associated with a concept and the SRILM tool using modified Kneser-Ney smoothing. Base2 was built by hand from language models derived

from multiple sources including a large web crawl, the gigaword corpus and parliamentary debate transcripts. These language models were interpolated with weights derived through PPL measurements (see chapter 2.1.2). All other ASR components were the same as the system built for the 2006 TC-STAR evaluation. Although the development set contained 144 utterances they spanned

	Base1	Base2
WER	21.5%	18.2%
PPL on Reference Text	277.1	100
PPL on best Baseline Hypothess	385.4	147.2

Table 5.1: Baseline measurements

299 sentences in the reference text. Perplexity measurements were carried on both the reference data *"PPL (R)"* as well as on the hypotheses from the best system in 2006 TC-STAR evaluation *"PPL (B)"*. That fact that the reference sentences and utterances are not aligned meant that the *history experiment* could not easily use the references for PPL measurements and that calculating the WER required an alignment preproccessing stage.

5.3 Testing the Effects of Different sets of Concept LMs

In this first experiment the effects of different numbers and types of concepts used in the language model were tested. When building language models, only concepts with more than 500 lines of associated text were used. To get the most out of the concept language models, different smoothing techniques were used for different amounts of concept text. Modified Kneser-Ney smoothing was used on concepts with more than 10000 lines of associated text and Witten-Bell smoothing was used on concepts with between 500 and 10000 lines. Language models use up a lot a memory and even loading only 1000 of the 89823 ODP concept language models requires over 20GByte of RAM when decoding a 45s sentence. Since nowhere near all 89823 language models can be loaded at once only concepts that are subconcepts of *"Top/Society/Government/"* or *"Top/Regional/Europe/"* are used. These 9971 concept LMs are sorted by their size and the largest and most general 20 are removed. To test the presumtion that more concepts should lead to better results, four adaptive language models are built from the remaining concepts. The interpolation weights for the base language model and the adaptive part are both set to 0.5 and the baseline hypothesis is used for adaption.

Adapt10 : The 10 largest remaining concepts.

Adapt100 : The 100 largest remaining concepts.

Adapt500 : The 500 largest remaining concepts.

Adapt1000 : The 1000 largest remaining concepts.

LM	PPL (R)	WER
Base1	277.137641284	**21.5%**
Base1+Adapt10	274.718	22.6%
Base1+Adapt100	237.35	21.3%
Base1+Adapt500	203.054	21.5%
Base1+Adapt1000	183.676	**20.5%**

Table 5.2: PPL and WER tested on language models with different numbers of concept LMs

These were tested on the development subset using Base1 as the baseline language model. Table 5.2 clearly shows that the presumtion was correct, adding more concepts improved the performance of the language model. Adaptive language models using 100 or more concepts performed better than or equal to the base line language model, with the Adapt1000 language model reducing the WER of the baseline language model by 1% absolute.

To demonstrate that this improvement doesn't just stem from using more data a further language model was built from the text of all the concepts that are subconcepts of *"Top/Society/Government/"* or *"Top/Regional/Europe/"*. This language model built from these selected concepts is then interpolated with the base language models to give:

Base1Mix : 50% Base1, 50% LM built form selected ODP concepts

Base2Mix : 50% Base2, 50% LM built form selected ODP concepts

The results of these language models as well as the results of Adapt1000X can be seen in Table 5.3. Adapt1000X is similar to Adapt1000 with the sole difference being that only the direct text and not the associated text is used to build the concept language models. Neither Base1Mix nor Base1+Adapt1000X perform as well as the Base1+Adapt1000 language model. Table 5.4 shows that when using Base2 as the base language model the adaptive language model still outperforms the interpolated language model Base2Mix But this time using the adaptive language model, with a WER of 18.6%, does not improve on base language model with a word error rate of only 18.2%.

LM	PPL (R)	WER
Base1	277.137641284	**21.5%**
Base1+Adapt1000	183.676	**20.5%**
Base1+Adapt1000X	258.74	45.6%
Base1Mix	-	24.1%

Table 5.3: PPL and WER tested on language models with different sets of Concept LMs using Base1 as the baseline language model.

LM	PPL (R)	WER
Base2	100.004	18.2%
Base2+Adapt1000:	85.421	18.6%
Base2Mix	-	20.5%

Table 5.4: PPL and WER tested on language models with different sets of Concept LMs using Base2 as the baseline language model.

5.3.1 Evaluation Time

The experiments were run on a cluster of 12 high powered computers with 2 AMD Opteron 2352 quad-core Barcelona 2.1GHz processors and 32 to 64 GBytes of RAM each. A high amount of parallelization was achieved by having different utterances decoded on different machines. Using only one quad-core Opteron processor on only a single node the baseline measurement of Base1 on the development subset was completeted in about 6 hours. The same task required about 19 hour when using Base1+Adapt1000 as the language model. Adaptive language models with fewer concepts were slightly faster: 13 hours for Base1+Adapt10 with 10 concepts and 16,5 hours for Base1+Adapt500 with 500 concepts. The adaptive language models wasted a lot of time initializing unneeded components and re-initializing large parts of the language model after decoding each utterance. This re-initializing process was necessary to keep the memory usage down but it could and in future should be done a lot more efficiently.

5.4 Adapting to Different Histories

The next parameter to test is the history to which the language model is adapted. As was discussed at the end of the last experiment, the time to measure the word error rate of single parameter configuration is very high. Therefore the WER was only measured on some configurations. The Adapt1000 language model from experiment one is used and adapted to different histories. In the first set of measurements it is adapted to the baseline hypothesis and the baseline hypotheses of the past 0 to 8 utterances. The second set of measurements only adapts the language model to the baseline hypotheses of the past 1 to 8 utterances.

	Base1 + Adapt1000		Base2 + Adapt1000	
History length	WER	PPL (B)	WER	PPL (B)
0 + B	20,5%	253.73	18.6%	118.54
1 + B	20,5%	256.73	18,4%	120.09
2 + B	-	259.22	-	120.70
3 + B	-	262.46	-	121.94
4 + B	-	263.31	-	122.08
5 + B	-	263.70	18,4%	122.36
6 + B	-	262.50	-	122.40
7 + B	-	261.81	-	122.24
8 + B	-	262.38	-	122.16
Base1, no adaptation			21.5%	285.5
Base2, no adaptation			18.2%	147.3

Table 5.5: Perplexity measured on the baseline language hypotheses and word error rate, adapting to different history lengths and the baseline hypothesis

At first glance it appears that according to table 5.5 using more history to adapt to actually makes the language model worse but that was to be expected in this particular case. The PPL is measured on the baseline hypotheses and this table shows that adapting the language model to only the baseline hypotheses reduces the perplexity the most. Adding more history to adapt to only adds more less relevant data. Table 5.5 indicates that when the baseline hypothesis is not available for adaption, increasing the amount of history the language model adapts to can result in PPL reductions.

	Base1 + Adapt1000		Base2 + Adapt1000	
History length	WER	PPL (B)	WER	PPL (B)
1	21.0%	277.91	19.5%	124.65
2	-	272.99	-	123.96
3	-	272.112	-	124.06
4	-	271.49	-	123.99
5	-	270.21	-	123.96
6	-	270.40	-	124.17
7	-	269.07	-	123.91
8	-	268.87	-	123.76
Base1, no adaptation			21.5%	285.5
Base2, no adaptation			18.2%	147.3

Table 5.6: Perplexity measured on the baseline language hypotheses and word error rate while adapting to different history lengths

5.5 Evaluating Interpolation Parameters

All the previous experiments have so far interpolated the base language model and the adaptive part with equal weights. To find out whether or not this parameter is correctly set or could be tweaked, the word error rate of the language model Base2+Adapt1000 adapting to the base LM hypothesis, is calculated using different interpolation weights. It can clearly be seen in table5.7 that the original interpolation weights of 0.5 and 0.5 were incorrectly chosen. Weighting the baseline language model with 0.8 and the adaptive part with 0.2 decreased the WER to 17.9%. This is also below the 18.2% WER of the baseline.

5.6 Examining the Concept Interpolation Weights

The interpolation between the base language model and the adaptive part is not the only interpolation that has to be examined. The concept language models returned by the Selector are interpolated to form the “adaptive part”. The interpolation weights used are calculated by normalizing the similarties of the cosine similarity metric. For pratical purposes only the 10 highest similarties

Base2 Weight	Adapt1000 Weight	WER
0.5	0.5	18.6%
0.6	0.4	19.7%
0.7	0.3	18.0%
0.8	0.2	**17.9%**
0.9	0.1	**17.9%**
Base2 baseline		
1.0	0	**18.2%**

Table 5.7: WER measured using different interpolation parameters

are used and all others are set to 0. Unfortunately this parameter could not be easily adjusted[1]. To evaluate how good these weights are the Base2 language model is examined more closley. As mentioned at the beginning of this chapter, Base2 was built by interpolating 8 existing language models based on weights derived from PPL measurements producing an optimal language model for the domain. A new adaptive language model **Base2split** is designed using each of these language models as concepts and the text used to build them is used to make attribute vectors for the concepts. The interpolation weights are calculated dynamically each time this language model *adapts* to something. For this language model its interpolation weight with the baseline language model Base2 was raised to 0.9. A WER of 18.2% for Base2split adapting to the baseline hypothosis was measured on the development subset and equaled the 18.2% WER of Base2 with standard mixing weights. This result shows that the cosine similarity interpolation weights are on par with the original ppl optimization ones.

5.7 Evaluation Subset Results

Using the results of this chapter a final test was set up on the evaluation subset. The baseline was Base2 which achieved a WER of 10.5 on the evaluation subset. The following parameters were chosen for the adaptive language model:

Composition : The Adapt1000 from experiment one was used.

History : The history incompased the baseline hypothosis and the previouse 3 hypothoses.

[1] This might not be necessary since the similarity seems to drop off very fast.

Interpolation Weights : The Base2 interpolation weight was set at 0.8 leaving 0.2 for the adaptive part.

Using five nodes from the cluster described in section 1 this evaluation took just under 28 hours to run. The end result was a disappointing WER of 10.8%.

6 Conclusions and Observations

The last chapter demonstrated that under some conditions the adaptive language model can outperform a baseline language model. When the general purpose language model Base1 is used then the adaptive language model is able improve the WER from 21.5% to 20.5%. This tested the applicability of the language model to unseen domains. Neither Base1 LM nor the adapt1000 LM used any domain knowledge. From this we can conclude that the adaptive language presented in this thesis might be useful when dealing with a new domain.

Although it should also be noted here that running a decoder with the adaptive language model increases the decoding time by about threefold. A Testset that can be decoded in 6 hours with the base language model take about 19 hours to decode with the adaptive language model.

The fact that with increasing history lengths the perplexity actually got worse in table **??** may seem surprising at first but on closer examination this is exactly what should be expected. With a history length 0 the language model was adapting only to the baseLM hypothosis which was exactly on what the PPL was being measured. Increasing the history length meant that the language model was adapting to something less accurate. The, up until a point, expected slight decrease in PPL as the history length gets longer can be seen in table **??**.

Unfortunately, this thesis has not been able to show that the adaptive language model can improve the WER of a hand built and optimized domain specific language model. Although some parameter configurations on the development subset did in fact reduce the WER compared to the original Base2 language model, these reduction were not mirrored in the results on the evaluation subset. Perhaps implementing some of the possible features described in chapter 7 will accomplish this.

The dynamic calculation of the language model interpolation weights in the Selector has been shown to be on par with optimizing them using ppl measurements this might lead to a new method of interpolating language models on the fly.

7 Future Work

This thesis has only scratched the surface of what could be done with adaptive language models built from interlinked data sources. As has already been alluded to in chapter 4, a future version of this language model should integrate the selector directly into the decoder. That is not the only possible improvement to the adaptive language model.

7.1 Optimize and Speed-up

As has already been mentioned, the adaptive language model slows down the decoder a lot. The time to decode the testset went from 6 for the baseline measurement to 19 hours when using the Adapt1000 language model. A future version might be able reduce even further the amount of "baggage" and "crud" produced by the currently unused n-grams.

7.2 Decay Parameters for Improved Dynamic Interpolation

The history window used is very basic. When adapting to a history the last m hypotheses are all considered equally important, while the hypothesis just before them is not considered at all. Similar to the method used in cache language models a decaying factor could be added to weight the hypotheses in the history.

7.3 Try Different Metrics and Classifiers in the Selector

The selector only used the cossim metric when comparing attribute vectors built from input word sequences to the concept attribute vectors. Instead more sophisticated classifiers like support vector machines or decision trees could be tested.

7.4 Dynamic Vocabulary

Right now all concept language models use the same vocabulary as the base language model. A potential improvement to this language model would involve using a larger total vocabulary. To begin with, only the base language model's vocabulary would be loaded and used in the search vocabulary. Each concept language model would also contain a set of extra words which would be dynamically added to the search vocabulary.

7.5 Reduce Memory Requirements

The current SelectLM internal vocabulary mapper requires an enormous amount of space. Even loading only 1.000 language models and with a search vocabulary of only 50.000 words the vocabulary mapper bloats up to an array with 50.000.000 cells each using up 2 bytes of memory. To load a vocabulary larger than 65.535 words the cell size has to be increased to 4 bytes. So 10.000 concept language models with a total vocabulary 100.000 words require 4.000.000.000 bytes of memory.

There are two possible ways of dealing with this problem: the first is modifying the way n-gram word numbers are assigned so that a vocabulary mapper is no longer required. This has the downside of meaning that every n-gram has to have a score for every word. The other solution is to compress the mapping table.

7.6 Incorporate more Data

The websites linked to by the ODP contain links to other most likely related websites. More html data could be gathered by spidering down a level or two. Not only website data but also text from pdfs and other files could be used. Most Wikipedia articles contain links to external websites and documents that could easily be mined for more data.

7.7 Incorporate more Meta Information

Wikipedia articles are often assigned to several categories. This category meta information might be used to build further concepts with their associated text being the text contained in all articles

tagged with that category.

7.8 Use this Language Model in Machine Translation

Machine translation systems face similar problems to speech recognition systems, so although this language model was built with ASR in mind it might also be useful in MT. If concepts existed pairwise in both languages then the language model could adapt to the input text.

Bibliography

[BJM90] L.R. Bahl, F. Jelinek, and R.L. Mercer. A maximum likelihood approach to continuous speech recognition. 1990.

[BMDPL92] P.F. Brown, R.L. Mercer, V.J. Della Pietra, and J.C. Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479, 1992.

[CG96] S.F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 310–318. Association for Computational Linguistics Morristown, NJ, USA, 1996.

[Clo01] P. Clough. A Perl program for sentence splitting using rules, 2001.

[CR97] PR Clarkson and AJ Robinson. Language model adaptation using mixtures and an exponentiallydecaying cache. In *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on*, volume 2, 1997.

[DBB52] K. Davis, R. Biddulph, and S. Balashek. Automatic Recognition of Spoken Digits, JASA. *Bd*, 24:637–642, 1952.

[Dud39] H. Dudley. The vocoder. *Bell Laboratories Record*, 17:122–126, 1939.

[FD58] DB Fry and P. Denes. The solution of some fundamental problems in mechanical speech recognition. *Language and Speech*, 1(1), 1958.

[FF59] J.W. Forgie and C.D. Forgie. Results Obtained from a Vowel Recognition Computer Program. *The Journal of the Acoustical Society of America*, 31:844, 1959.

[FotFI22] H. Fletcher and Journal of the Franklin Institute. *The Nature of Speech and Its Interpretation.* JB Lippincott, 1922.

[Gab07] E. Gabrilovich. Feature generation for textual information retrieval using world knowledge. In *SIGIR FORUM*, volume 41, page 123, 2007.

[GC94] W.A. Gale and K.W. Church. A program for aligning sentences in bilingual corpora. *Computational linguistics*, 19(1):75–102, 1994.

[GH99] D. Gildea and T. Hofmann. Topic-Based Language Models Using EM. In *Sixth European Conference on Speech Communication and Technology*. ISCA, 1999.

[HCL07] A. Heidel, H. Chang, and L. Lee. Language Model Adaptation Using Latent Dirichlet Allocation and an Efficient Topic Inference Algorithm. In *Proc. of Interspeech*, 2007.

[JBM75] F. Jelinek, L. Bahl, and R. Mercer. Design of a linguistic statistical decoder for the recognition of continuous speech. *Information Theory, IEEE Transactions on*, 21(3):250–256, 1975.

[JR] BH Juang and L.R. Rabiner. Automatic Speech Recognition–A Brief History of the Technology Development. *Encyclopedia of Language and Linguistics, Elsevier (to be published)*.

[Kat87] S. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 35(3):400–401, 1987.

[KDMUoCS90] R. Kuhn, R. De Mori, McGill University, and School of Computer Science. A cache-based natural language model for speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(6):570–583, 1990.

[KN95] R. Kneser and H. Ney. Improved backing-off for m-gram language modeling. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, 1995.

[KP02] D. Klakow and J. Peters. Testing the correlation of word error rate and perplexity. *Speech Communication*, 38(1-2):19–28, 2002.

[Lid20] G.J. Lidstone. Note on the general case of the Bayes-Laplace formula for inductive or a posteriori probabilities. *Transactions of the Faculty of Actuaries*, 8:182–192, 1920.

[LKM03] IR Lane, T. Kawahara, and T. Matsui. Language model switching based on topic detection for dialog speech recognition. In *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*, volume 1, 2003.

[Low76] B.T. Lowerre. The harpy speech recognition system. 1976.

[LRS83] SE LEVINSON, LR RABINER, and MM SONDHI. An Introduction to the

Application of the Theory of Probabilistic Functions of a Markov Process to Automatic Speech Recognition. *The Bell System Technical Journal*, 62(4), 1983.

[Mar89] FA Marvasti. An iterative method to compensate for the interpolation distortion. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37(10):1617–1621, 1989.

[PB52] G.E. Peterson and H.L. Barney. Control Methods Used in a Study of the Vowels. *The Journal of the Acoustical Society of America*, 24:175, 1952.

[SB87] G. Salton and C. Buckley. Term weighting approaches in automatic text retrieval. 1987.

[Sch08] Tanja Schulz. Multilinguale mensch-maschine kommunikation lecture notes, 2008.

[SD64] T. Sakai and S. Doshita. The Automatic Speech Recognition System for Conversational Sound. 1964.

[SG05] R.G. Sarikaya and A.Y. Gao. Rapid Language Model Development Using External Resources for New Spoken Dialog Domains. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings.(ICASSP'05). IEEE International Conference on*, volume 1, 2005.

[SGN05] A. Sethy, P.G. Georgiou, and S. Narayanan. Building Topic Specific Language Models from Webdata Using Competitive Models. In *Ninth European Conference on Speech Communication and Technology*. ISCA, 2005.

[SM86] G. Salton and M.J. McGill. *Introduction to modern information retrieval*. McGraw-Hill, Inc. New York, NY, USA, 1986.

[SN61] J. Suzuki and K. Nakata. Recognition of Japanese vowels–preliminary to the recognition of speech. *J. Radio Res. Lab*, 37(8):193–212, 1961.

[Sto02] A. Stolcke. SRILM-an extensible language modeling toolkit. In *Seventh International Conference on Spoken Language Processing*. ISCA, 2002.

[TS05] Y.C. Tam and T. Schultz. Dynamic Language Model Adaptation Using Variational Bayes Inference. In *Ninth European Conference on Speech Communication and Technology*. ISCA, 2005.

[TS07] Y.C. Tam and T. Schultz. CORRELATED LATENT SEMANTIC MODEL FOR UNSUPERVISED LM ADAPTATION. In *Acoustics, Speech and Signal*

Processing, 2007. ICASSP 2007. IEEE International Conference on, volume 4, 2007.

[Vin68] TK Vintsyuk. Speech recognition by dynamic programming methods. *Russian Kibernetika*, 1(4):81–88, 1968.

[Vin71] TK Vintsyuk. Element—by—element recognition of continuous speech composed of the words of given vocabulary. *Russian Kibernetika*, 2:133–143, 1971.

[WB89] I.H. Witten and T.C. Bell. The zero frequency problem: Estimating the probabilities of novel events in adaptive text compression. 1989.

[You96] S. Young. A review of large-vocabulary continuous-speech recognition. *IEEE Signal Processing Magazine*, 13(5):45–57, 1996.