Speech Recognition
Software and Vidispine

Tobias Nilsson

April 2, 2013
Master’s Thesis in Computing Science, 30 credits
Supervisor at CS-UmU: Frank Drewes
Examiner: Fredrik Georgsson

UMEÅ UNIVERSITY
DEPARTMENT OF COMPUTING SCIENCE
SE-901 87 UMEÅ
SWEDEN
Abstract

To evaluate libraries for continuous speech recognition, a test based on TED-talk videos was created. The different speech recognition libraries PocketSphinx, Dragon NaturallySpeaking and Microsoft Speech API were part of the evaluation. From the words that the libraries recognized, Word Error Rate (WER) was calculated and the results show that Microsoft SAPI performed worst with a WER of 60.8%, PocketSphinx at second place with 59.9% and Dragon NaturallySpeaking as the best with 42.6%. These results were all achieved with a Real Time Factor (RTF) of less than 1.0.

PocketSphinx was chosen as the best candidate for the intended system on the basis that it is open-source, free and would be a better match to the system.

By modifying the language model and dictionary to closer resemble typical TED-talk contents, it was also possible to improve the WER for PocketSphinx to a value of 39.5%, however with the cost of RTF which passed the 1.0 limit, making it less useful for live video.
# Contents

1 Introduction  .............................................. 1  
  1.1 Report Layout .......................................... 2  
  1.2 Problem Statement ...................................... 3  
  1.3 Goals .................................................. 3  
  1.4 Methods ............................................... 4  
  1.5 Related Work .......................................... 4  

2 Models and Algorithms for Speech Recognition ......... 5  
  2.1 What is Speech Recognition? ............................ 5  
     2.1.1 Speech Recognition systems ......................... 5  
     2.1.2 Requirements for speech recognition ............... 6  
     2.1.3 Difficulties in speech recognition .................. 7  
  2.2 Feature Extraction ..................................... 8  
  2.3 Statistical Models ..................................... 8  
     2.3.1 Acoustic models .................................... 9  
     2.3.2 Language models .................................. 11  
  2.4 Algorithms ............................................. 12  
     2.4.1 Formal Overview ................................... 13  
     2.4.2 Forward Search .................................... 14  
     2.4.3 Viterbi search ..................................... 17  
     2.4.4 A* decoder / Stack decoder ......................... 23  

3 Speech Recognition Systems and Libraries ............. 29  
  3.1 CMU Sphinx ........................................... 30  
  3.2 Julius .................................................. 31  
  3.3 Kaldi ................................................... 31  
  3.4 HTK ..................................................... 32  
  3.5 Windows Speech Recognition ............................ 32  
  3.6 Dragon NaturallySpeaking ............................... 32  
  3.7 SpeechMagic ........................................... 33  
  3.8 Comparison of libraries ................................. 33
4 Evaluation of Existing Systems and Libraries  35
4.1 The evaluation test .......................... 35
4.1.1 Limitations on test ...................... 36
4.1.2 Test Input .............................. 36
4.1.3 Problems .............................. 37
4.1.4 Output ................................ 37
4.1.5 Metrics ................................. 38
4.1.6 Calculation of metrics .................. 39
4.2 Practicalities using the libraries ....... 39
4.2.1 CMU Sphinx ......................... 39
4.2.2 Microsoft Speech API ............... 40
4.2.3 Dragon NaturallySpeaking .......... 40
4.2.4 Kaldi ................................. 41
4.2.5 Julius ................................ 41
4.2.6 Youtube ASR ......................... 42
4.3 Evaluation Results ....................... 42
4.3.1 Produced Words ....................... 42
4.3.2 Word Error Rate ...................... 44
4.3.3 Real Time Factor ..................... 44
4.3.4 Analysis of the results ............... 45

5 Integration of Speech Recognition in Vidispine  53
5.1 Selection of library ....................... 53
5.2 Vidispine ................................ 54
5.3 ASR module ................................ 54
5.4 Prototype application .................... 57

6 Improvements of PocketSphinx  59
6.1 Dictionary ................................ 59
6.2 Language Model ......................... 60
6.2.1 RTF .................................. 61
6.3 Custom LM and Dictionary ............. 61
6.4 T-test .................................. 62
6.5 Confusion Matrix ......................... 63
6.5.1 Interpretation of Confusion Matrix .. 64
6.5.2 Usage ................................ 65

7 Conclusions and Future Work  69
7.1 Restrictions ............................... 71
7.1.1 Evaluation restrictions .............. 71
7.1.2 Software restrictions ............... 72
7.2 Future work ............................... 73
7.2.1 Evaluation ............................ 73
7.2.2 Software ............................... 74
<table>
<thead>
<tr>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>8  Acknowledgements</td>
</tr>
<tr>
<td>References</td>
</tr>
<tr>
<td>A  Evaluation videos</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Overview of the general idea of speech recognition, which is the conversion of a raw audio signal into a sequence of words.</td>
</tr>
<tr>
<td>2.2</td>
<td>An overview of how language models and acoustic models are used in a speech recognizer, based on a figure in [11]. The audio signal is analyzed and represented as feature vectors. The recognizer make use of a language model, acoustic model and a dictionary.</td>
</tr>
<tr>
<td>2.3</td>
<td>HMM of a fictional phoneme. This figure does not represent any real feature numbers, but is only used for demonstrative purposes. Real HMMs are usually more complex, containing more states, and feature vectors instead of just a single number, as well with transitions going from most states to most other states.</td>
</tr>
<tr>
<td>2.4</td>
<td>The word &quot;need&quot; represented as a HMM. Note that this is a simplification where the states contain phonemes instead of feature numbers. This is only for demonstrative purposes. A real HMM representing the entire word would contain too many states and transitions to be reasonable for demonstration.</td>
</tr>
<tr>
<td>2.5</td>
<td>The word &quot;neat&quot; represented as a HMM. Note that this is a simplification where the states contain phonemes instead of feature numbers. This is only for demonstrative purposes. A real HMM representing the entire word would contain too many states and transitions to be reasonable for demonstration.</td>
</tr>
<tr>
<td>2.6</td>
<td>The words &quot;I&quot;, &quot;need&quot;, &quot;the&quot;, &quot;on&quot; represented as HMMs. Note that the HMMs are simplified, in the sense that the states contain phonemes instead of feature numbers. This is done to make it easier to explain the algorithms. Note also that 2.6b is the same as Figure 2.4.</td>
</tr>
<tr>
<td>2.7</td>
<td>The words &quot;I&quot;, &quot;need&quot;, &quot;the&quot;, &quot;on&quot; represented as combined HMM, using bigrams to represent the probabilities between words. Note that &quot;#&quot; represents the beginning of a sentence, meaning that sentences that start with &quot;I&quot;, for example, has the initial probability 0.079.</td>
</tr>
</tbody>
</table>
2.8 Representation of word sequences allowed in an example language. The figure only presents a few words of a language and only four levels deep for demonstration purposes. A complete network of allowable word sequences would be very large.  

2.9 First iteration of the Stack decoding algorithm, based on the example network in Figure 2.8 and with fictional scores. The most likely sentence is "the" at this point.  

2.10 Second iteration of the Stack decoding algorithm continuing the example in Figure 2.9, with continued fabricated scores. The most likely sentence is "i" at this point.  

2.11 Third iteration of the Stack decoding algorithm, continuing the example from Figure 2.10, with continued fabricated scores. The most likely sentence is "i want" at this point.  

4.1 The number of produced words for each video and library. For each video, the results from the different libraries are grouped, the first column in each group being the number of words in the original transcript.  

4.2 Average Word Error Rate for the different libraries and Youtube. The total average is presented in blue, the average for women in red and the average for men in orange.  

4.3 Average Real Time Factor for the different libraries. The total average is presented in blue, the average for women in red and the average for men in orange.  

4.4 Real Time Factor for the separate videos of the tests for the different libraries. The first four bars in each group (library) are the female videos, whereas the last four bars are the male videos.  

5.1 Vidispine module graph. Visualization of how the modules in Vidispine work together.  

5.2 Module overview. Simple overview of the speech recognition module functionality.  

5.3 Illustration of the cut-off problem, where each break in the input data causes a disturbance of the recognition.  

5.4 Illustration of the overlap solution for the cut-off problem. By recognizing some of the data multiple times, and then matching the overlap (darker green), a complete recognition can still be achieved.  

5.5 Screenshot of the prototype application written to demonstrate the possibilities of the model. Here a search for the word "world" was made, having a set of search results to the right, with the chosen video displaying to the left.
6.1 Word Error Rates for PS (standard models) and the different combinations of the custom language model and dictionary (PS DICT, PS LM, PS LM DICT). .............................................................. 61
6.2 Real Time Factors for PocketSphinx using standard dictionary and models (PS) as well as custom language model (PS LM), custom dictionary (PS DICT) and the usage of both the custom language model and the custom dictionary (PS LM DICT) ............................................. 66
6.3 Excerpt of a confusion matrix for the single word ”about”, here represented as a tree. It can be seen that ”about” was recognized as ”about” 72 times, as ”review” 2 times, as ”eleven” 1 time, as ”got” 1 time and as multiple words 40 times ................................................. 67
## List of Tables

2.1 Matrix of output results from the Forward Algorithm for the word "need", based on the HMM in Figure 2.4, showing the probability $P(O|w)$ in the rightmost column on the last row. Each state in the HMM is represented on a single row, and each column represents the observations or time $t$. ................................. 16

2.2 Matrix of output results from the Forward Algorithm for the word "neat", showing the probability $P(O|w)$ in the rightmost column on the last row. States can be seen in the leftmost column. ................................. 17

2.3 Matrix of output results from the Viterbi Algorithm example, using the HMM in Figure 2.7, showing the probability $P(O|w)P(w)$ in the rightmost column on the last row. Each row represents a state in the HMM, where the states are grouped for each word. Each column represents an element in the observation sequence. ................................. 21

3.1 Comparison of libraries. WERs as described in Section 3 are used to make an estimated comparison between the libraries. ................................. 33

4.1 Hardware used in the evaluation ................................................................. 36

4.2 The library names and their abbreviations used for Tables and Figures from this point in the report ................................................................. 42

4.3 Expected word count and produced words for the different libraries and Youtube. ................................................................. 43

4.4 Word Error Rate total average, women average and men averages for the different libraries. Averages for Youtube takes into account that one value is missing. ................................................................. 44

4.5 Real Time Factor total average, women average and men average for the different libraries. ................................................................. 45

4.6 WER results from this evaluation (TED evaluation) and results of Keith Vertanen. Note that the results are achieved using different methods. ................................................................. 48

4.7 The ttest2 results for the difference in WER between women and men for the different libraries ................................................................. 50
6.1 Total average of WER as well as average for women and men for the standard language model and dictionary (PS), custom language model (PS LM), custom dictionary (PS DICT) and finally custom language model and dictionary (PS LM DICT). 59
6.2 Accuracy for specific key-words for PS, PS LM, PS DICT and PS LM DICT. 64
Chapter 1

Introduction

Speech recognition is an area that is being more and more present for the average user. This is something that exists today in smartphones where one of the most known application is Siri for Apple products. Speech recognition will be more and more common in the future as the amount of data grows and devices contain more features.

Having good speech recognition can for example reduce costs for automated customer (telephone) communication, where it is possible for a customer to state his business with his voice and to be automatically transferred to the correct person, or even conduct the business without contact with a real person.

Speech recognition can also help students learn a natural language better. Rosetta Stone\textsuperscript{1} is an example of this, where students can get direct feedback on the pronunciation of words.

The above examples can be considered somewhat simple compared to dictation of speech, since in those cases it is possible to limit the number of words, and thus the search area, that need to be considered. Speech recognition for continuous-speech-dictation produce lower accuracy rates because of the fact that more words and word combinations need to be considered, which is still a large research topic.

The cause for this thesis work is by a request from Codemill AB, which is an expanding IT programming firm located in Umeå with cooperation with the company Vidispine in Stockholm. Codemill has around 25 employees and had a turnover of 12.9 million SEK for the year 2011. Vidispine AB has around five employees with a turnover of 8.4 million SEK for the year 2011. Vidispine AB has developed a media asset platform that deals with storage, handling and transmission of video, and should be seen as a middleware for other applications.

Codemill and Vidispine want a system where it is possible to be able to recognize, from videos, who said what at what time in which video, by using face recognition, speech recognition and voice recognition.

\textsuperscript{1}http://www.rosettastone.eu/
For a potential customer, this could for example be used for videos from seminars or meetings, so that for example possible upcoming trends could be detected. This kind of system could also be used for video surveillance, which could be used by companies both for good and evil.

The work presented in this report is limited to the recognition of speech, i.e. to automatically create a transcript of the spoken dialog in a video, the task being to evaluate the current available libraries for speech recognition in terms of performance and suitability.

The library that is best suitable for the needs of Codemill is used to generate meta-data for videos, containing words paired up with time-stamps. This is done for the Vidispine platform, where this feature is in demand. The accuracy of the recognition for the system is improved where possible, and the results may later be used in a larger system containing voice recognition and face recognition as well.

1.1 Report Layout

Chapter one gives an introduction to why this work was done, the goals with the project, what purposes and goals were set up for the work and how other work is related to this report.

Chapter two gives a brief explanation of what is meant by speech recognition and also explains the most common algorithms used in speech recognition, such as Forward Search, Viterbi Search, A* or Stack Search and also describes how acoustic and language models are involved in these.

Chapter three describes some of the most common libraries and projects involved in speech recognition, such as VoxForge, CMU Sphinx, Julius, Kaldi, HTK, Windows Speech Recognition and Dragon NaturallySpeaking. A quick comparison between the libraries are also made.

Chapter four presents how the evaluation of the libraries was done, the input and output to the test, the metrics used to compare the libraries and how the libraries were used. Furthermore, the results of the evaluation in terms of Word Error Rate and Real Time Factor as well as an analysis of the results can be seen in this chapter.

Chapter five describes how the integration of speech recognition was made in Vidispine, the choice of library, how the module works and how a sample application can make use of the work.

Chapter six discusses some improvements that are made for the selected library, that the language model and dictionary is replaced to improve the speech recognition. It also discusses how a confusion matrix can be used to analyze the most problematic words in the recognition.

Chapter seven discusses the conclusions that can be drawn from the results in this report, and also discusses what could have been done differently and also what can be made in future work.
Chapter Eight is dedicated to thank the people that have been a support during this works.

1.2 Problem Statement

Codemill wanted a module in Vidispine that could automatically generate a transcript of the speech in videos, for which existent speech recognition libraries were supposed to be evaluated. The most suitable library is then used for the Vidispine module.

The module itself should take a video as input, run the speech recognizer on the audio, and then make the generated text available with timestamps so that it is made searchable. It should for example be possible to perform a search on specific words, and the result should be the videos and timestamps where these words are recognized.

The evaluation is only done on libraries that already exist, and in general, material that already exist is used instead of implementing a completely new version. Both open-source and proprietary libraries are interesting for this evaluation. The evaluation of the libraries is limited to a small test set.

The research questions answered in this report are:

1. Using currently available speech recognition software, and with the limitations given below, how accurately can speech be recognized?
   The source audio is taken from TED-talks.
   The speakers are assumed to speak American English

2. Is it possible to increase the accuracy? If so, by how much?

3. Are there similar surveys in the literature? If so, what answers did they arrive at to the above questions?

1.3 Goals

A goal of the project is to find the best performing library for speech recognition and also the most suitable library that can be used to create a module in Vidispine, which should be used for automatic meta-data creation.

Another goal is to create the module for Vidispine, which takes a video as input, creates meta-data for the video and integrates it in the video, or makes the meta-data available in another way. This module shall be seen as a proof-of-concept for future projects.

A third goal is to get as good accuracy as possible within the time limits for the project, so that the meta-data can be as accurate as possible.
1.4 Methods

A study of the most common algorithms used in speech recognition is made so that the structure of the libraries can be investigated, and also to collect information about how it is possible to improve accuracy.

To be able to evaluate the different libraries, tests that others have performed is investigated and there are also separate tests performed, the results of which are compared with the results of others. The evaluation is measured with standard methods as far as possible, so that they can be compared to other similar evaluations.

The results from the evaluation and the information about the libraries are used to determine the best choice for Vidispine. By studying how the speech recognition engine works, an attempt is made to improve its models and, in this way, its accuracy.

1.5 Related Work

Keith Vertanen has done some research in speech recognition [29, 30] and has published some articles where the accuracy (word error rate to be more specific, see Section 4.1.5 for an explanation of this) of different recognizers is measured. Systems included in the articles are CMU Sphinx, Dragon NaturallySpeaking, Microsoft SAPI and HTK. The results of Vertanens work are compared with the results presented in this report. Since Keith Vertanen has compared the different libraries using the same methods for all libraries, it is possible to compare the libraries against each other.

In an article [9] by David Huggins-Daines and Alexander I. Rudnicky, adaptations of language models are made for specific videos of meetings. By making use of the written agendas for the meetings to adapt the language model, they show that it is possible to significantly improve the recognition accuracy.

A master thesis by Boris Guenebaut [8] that deals with automatic subtitle generation for videos using Sphinx4 can be considered to be similar to the final product that is requested for this work.
Chapter 2

Models and Algorithms for Speech Recognition

2.1 What is Speech Recognition?

Speech recognition is in one sentence the translation of a raw audio signal into text as illustrated in Figure 2.1, preferably with as high accuracy as possible. The recognized text can then be used as further instructions in a system or presented directly to the user as text. Speech recognition (and Text-To-Speech) have been in research for many years. Nevertheless, trying to increase the accuracy of speech recognition continues to be an important research topic, because of variations in, e.g., environment, speaker and context [2].

![Diagram of speech recognition process]

Figure 2.1: Overview of the general idea of speech recognition, which is the conversion of a raw audio signal into a sequence of words.

Creating artificial models of human verbal communication, so that humans and machines can communicate with each other is the main motivation for Speech Recognition [2]. The interesting idea that a human can have a conversation with a computer fascinates many people.

2.1.1 Speech Recognition systems

There are two main categories that speech recognition systems can be divided into. The first is the command and control type of system, where the system
can interpret a predetermined, often very small set of words or sentences, i.e. isolated words. This type of recognition is often used for controlling systems in one way or another, where specific words or phrases are used to activate different things in the system.

A sub-category of the command and control-type of system is interactive voice response systems. These are often used in call centers, where the caller can voice their request and get the information they want.

The other type of systems are dictation systems, where the system is meant to recognize all (as many as possible) words that the user is speaking from continuous speech. These systems require a language model which serves a similar purpose as the limited set of sentences in command and control types of systems.

This report focuses on the latter type, since the input data can be pretty much any video with any contents, and therefore needs to be able to recognize as many words as possible.

**Speaker dependency**

A speech recognition system can be speaker dependent or speaker independent. The former means that the system can be trained to match a specific user as well as possible. This limits the number of users of the same system, since there must exist training data and models for each user. These systems are generally better at recognizing a single person and would perform worse for new or unknown users. New users have to be trained separately.

Speaker independent systems are more tolerant to different types of voices. This may mean that the system is less accurate in general, since it needs to be more tolerant to the voices, and thus produces more false positives.

Speaker dependency is not focus for this work.

**2.1.2 Requirements for speech recognition**

For a speech recognizer to work, there must at least exist an acoustic model and preferably a language model.

The acoustic model is a representation of different raw audio input signals and a mapping of these signals to phonemes. The phonemes can then be mapped into possible words, by having a dictionary consisting of a mapping between phonemes and written words.

The language model contains rules of how the language to be recognized is organized, which basically consists of rules of how the words in the language can be ordered. Language models are used to limit the search space of the recognition, i.e. the most likely word sequences get highest priority. A very simple language model can contain simple and strict rules of sequences of words that can be recognized. A more sophisticated language model contains probabilities of words occurring after one another and a much larger vocabulary. Simple
models are usually used for the command and control types of speech recognition systems, whereas statistical models are normally used for the dictation types of systems.

By combination of the acoustic model and the language model, through the extraction of phonemes from the audio and using the language model to produce the most likely phrase from those phonemes, speech recognition is in work. An illustration of this can be seen in Figure 2.2, where the feature vectors contain recognized features of the raw audio signal. Feature vectors in a certain order can be interpreted as phonemes.

![Diagram of speech recognition system](image)

Figure 2.2: An overview of how language models and acoustic models are used in a speech recognizer, based on a figure in [11]. The audio signal is analyzed and represented as feature vectors. The recognizer make use of a language model, acoustic model and a dictionary.

When using statistical models to represent speech, the generic structure of a speech recognizer can be described as in in Figure 2.2. [11]

### 2.1.3 Difficulties in speech recognition

Speech recognition is complicated by many factors, some external ones that are hard to address as well as some that are possible to improve.

External factors, such as audio quality and noise can have a huge impact on the quality of the speech recognition. Noise, for example city background noise or audience sounds, can sometimes be loud compared to the relevant audio. This causes the sound waves to be altered which causes the recognized phonemes to be different, which will result in falsely recognized words. Noise can also be the result from bad recording equipment.

The size of the vocabulary will affect speed and accuracy of the recognition. A larger dictionary means that more words need to be considered for the recognition, which means it will take longer time to map the input with the best interpretation. A larger dictionary will also mean that the odds of
different word combinations sounding similar increases. If a system is meant
to do limited speech recognition, such as command and control programs, the
vocabulary can often be limited to a very small set of words, and therefore the
accuracy and speed can also be improved. Systems that aim to recognize as
many different words as possible, such as dictation software, will either require
a large vocabulary, making it slower, or have more mismatches. If it is known
that the input phrases belong to a certain domain, for example if the recogni-
tion is aimed at the medical industry, it may be possible to reduce the size of
the dictionary.

The quality of the spoken words is also very important. Words can for
example be spoken in different speeds, in different volumes, different pitches,
different accents, which makes the signal processing more difficult.

In an ideal world, when it comes to speech recognition, everyone should talk
in exactly the same way and use the same build-up of sentences and equipment
that would yield perfect recording quality with no noise. This, of course, is
something that will, most likely, never exist.

2.2 Feature Extraction

The raw audio signal is converted to a spectrum and a cepstrum, often using
the Fast Fourier Transform algorithm. This algorithm essentially converts the
audio signal from the ”time domain” to the ”frequency domain”. From the cep-
strum, feature numbers are extracted and stored into feature vectors. Often,
the Fourier Transform is not used directly but instead the Perceptual Linear
Prediction [17, 11] or Mel Frequency Cepstral [11] is used. Both of these how-
ever work by calculating the feature numbers (vectors) from the audio signal
in chunks, normally 20 to 30 ms in size, with a 10 ms offset at a time.

Each phoneme in a language consists of one or multiple feature vectors
after one another. These feature vectors are stored in the acoustic model that
represents different phonemes in a language. Using statistics, it is possible to
produce a list of feature vectors from an input, and these feature vectors can
then be mapped to phonemes and words.

2.3 Statistical Models

The most common type of speech recognition makes use of statistical models,
which means that the recognition process makes use of probabilistic methods.
Having a statistical database over how different phonemes are normally struc-
tured and how a language is structured, it is possible to analyze the input
signal to determine the most likely sequence of feature vectors and therefore
also phonemes and words. This is done by using previous information about
the input signal to determine the most likely next state.
2.3. Statistical Models

2.3.1 Acoustic models

An acoustic model is a representation of the pronunciation of phonemes of a language. The English language has around 40 phonemes [4].

A lexicon or a dictionary describes the vocabulary items and a representation of the pronunciation of the items, where each item is usually a word and the pronunciation is often the sequence of phonemes that the word consists of. The dictionary can hold multiple entries for one word, meaning that each word can be pronounced different ways, for example to represent different accents.

The phonemes themselves are derived from statistical data, which in turn is generated from a corpus, i.e. a text bank with associated spoken audio. VoxForge is an example that provides such a corpus. The corpus is used to create a statistical representation of the phonemes in the language\(^1\). These statistical representations are very commonly stored as Hidden Markov Models.

The accuracy of the acoustic model is dependent of the consistency of the pronunciations listed in the lexicon [11]. This makes sense, since having a single phoneme representing two different sounds would not cause a very accurate recognition.

Hidden Markov Models

A Hidden Markov Model (HMM) is a weighted automaton \(\lambda = (S, V, A, B, \pi)\), having transition probabilities on the edges between states. It consists of the following parts, based on [23]:

A finite set of states: The N states of the HMM, denoted as

\[ S = \{s_0, s_1, ..., s_{N-1}\} \] (2.1)

state transition probabilities: The probabilities of transitioning from one state to another, denoted as

\[ A = a[s_i, s_j] = P(s_i | s_j), 0 \leq i, j < N \] (2.2)

observation symbols: The possible observation symbols, denoted as

\[ V = \{v_1, v_2, ..., v_M\} \] (2.3)

observation likelihoods: The probability that an observation (symbol) is generated from a certain state, denoted as

\[ B = b[s_i, o_t] = P(o_t = v_k | s_i), 0 \leq i < N, 1 \leq k \leq M \] (2.4)

initial distribution: The initial probability that the HMM will start in a certain state, denoted as

\[ \pi = \pi[s_i], 0 \leq i < N \] (2.5)

\(^1\)http://www.voxforge.org/home/docs/faq/faq/what-is-an-acoustic-model
Having a HMM $\lambda$, it is possible to generate an observation sequence with the following procedure [23]:

1. Choose $q_0$, according to the initial state distribution $\pi$.

2. Set $t = 0$.

3. Choose $O_t = v_k$ according to $b[s_i, o_t]$.

4. Transit to a new state $q_{t+1}$ according to $a[s_i, s_j]$.

5. Set $t = t + 1$. If $t < T$ return to step 3, else terminate.

In the procedure 1-5 above, each state sequence $Q = q_0, ..., q_{T-1}$ of length $T$ has the probability $P_Q = \pi[q_0] \prod_{i=1}^{T-1} A[q_{i-1}, q_i]$. $Q$, in turn, generates with probability $P_{Q,O} = \prod_{i=0}^{T-1} B[s_i, o_i]$ a given observation sequence $O = o_0, ..., o_{T-1}$ of length $T$. The probability that $O$ is generated by the state sequence $Q$ therefore is $P_Q P_{Q,O}$, which we can denote as $\lambda(Q,O)$. Now we imagine that we observe $O$ but do not know what state sequence gave rise to $O$ (The state transitions are "hidden"). The state sequence $Q$ that maximizes the probability $\lambda(Q,O)$ is the most likely explanation to $O$, and this is what we are looking for.

When it comes to speech recognition, HMMs are used to represent the acoustic model (phonemes) of a language. Each phoneme can even have its own HMM, where the states in the HMM are the "feature numbers" for different temporal positions in the phoneme. A simple HMM representing this can be seen in Figure 2.3, which represents a simplified fictional phoneme HMM. Possible sequences of feature numbers in this example could be ”33, 33, 33, 58, 21, 21”, ”33, 57, 21”, ”33, 33, 57, 21, 21”, which all would be recognized as this fictional phoneme. In a real example, a single phoneme could contain many more observed feature numbers, and the HMM would be more complex, having more states as well as consist of feature vectors instead of just a single feature number. The states may even consist of sub-phonemes. The figure is simplified for easier explanation.

A simplified example of a Hidden Markov Model with multiple words in a single HMM can be seen illustrated in Figure 2.7, where each state is represented by a phoneme (a real system would have feature vectors as states).

The HMM is trained before it is be used. The training is done by calculating the transition probabilities and the observation likelihoods for the HMM, based on audio with corresponding transcriptions from a corpus.

One issue with Hidden Markov Models is that it is often difficult to improve the performance by analyzing the errors. Nevertheless the Hidden Markov Model is the most popular choice in speech recognizers today [2].
2.3. Statistical Models

Figure 2.3: HMM of a fictional phoneme. This figure does not represent any real feature numbers, but is only used for demonstrative purposes. Real HMMs are usually more complex, containing more states, and feature vectors instead of just a single number, as well with transitions going from most states to most other states.

Neural networks

Using neural networks for speech recognition may help to keep the models trained as the input changes over time, and may even perform better and better over time.

Neural networks have the advantage that they can be used immediately without training, and that they learn as the system is used. This however assumes that an award-system of some type is activated.

In [31], explorations of neural networks in large vocabulary continuous speech recognition are made, where a word recognition accuracy was achieved of 62.8% and at some cases 83.2%. It is claimed that neural networks are better to adapt to different speech environments compared to using traditional HMMs which are aimed at a certain target area. They are also claimed to be very cost efficient since they does not require as much training data as traditional models.

90.5% accuracy has been reported in [27], where a HMM reached 86.0% under similar conditions, which would indicate that neural networks work better than HMMs.

According to Microsoft, neural networks have reduced the error rates from 25% to below 15% compared to standard HMMs, which also suggests that neural networks may be the future of speech recognition.\footnote{http://www.youtube.com/watch?v=Nu-n1QqFCKg}

2.3.2 Language models

A language model describes the legal sentences in a language, or how words in a language can be ordered to be part of the language. These are used to limit the search space of the recognizer. Without the language model, the recognizer would have to consider every possible word combination in the vocabulary.
N-grams are very common in speech recognition and consist of all legal word sequences of length $N$ which are allowed in the language. Each N-gram contains a probability that represents the probability that the $N$ words in the N-gram are used in that order in the language. When the recognizer tries to estimate the next word, it uses the N-grams to find the most likely word sequence based on the previous N-1 words to find the most likely Nth word.

A formal description of a word bigram (2-gram) model is presented in [24], and can be seen in Equation 2.6. This calculates the probability for a sentence $W = w_1, w_2, ..., w_N$, using the probability of adjacent words. That is, given the previous word $w_{n-1}$ in the sentence, we can calculate the probability for the next word being $w_n$, and by multiplying these probabilities we get the probability for the entire sentence $W$. This, for simplicity, also assumes that $P(w_n)$ is independent of $w_1, w_2, ..., w_{n-2}$, and thus only depends on $w_{n-1}$.

$$P(w_1, w_2, ..., w_N) = \prod_{n=0}^{N} P(w_n|w_{n-1}) \quad (2.6)$$

**Grammar**

A grammar is a type of language model, and consists of rules in which words can occur and is often implemented as a context-free grammar. A grammar can make use of word classifications, where each word in the dictionary is classified as a verb, noun, prepositions and so on. An example grammar can for example contain the word order ‘noun verb preposition noun’. A grammar can be more simplistic than that and can in a very simple form specify the specific words that can be recognized, but can also describe very complex languages. An example grammar can be seen in 2.7.

$$(he|she|it)(eats|drinks)a(soda|meal|burger) \quad (2.7)$$

Language models that use probabilities for word occurrences use estimates of how often those word combinations occur in a corpus. The corpus used for training should represent the language to be recognized. From the corpus, the frequency of each word and each N-gram is calculated, from which it is an easy task to calculate the probability of each word and each N-gram.

### 2.4 Algorithms

There exist different algorithms to turn spoken words into text. According to [11], the ones that perform the best are the ones that use statistical models where Hidden Markov Models is the most dominant.
2.4. Algorithms

2.4.1 Formal Overview

The general problem of finding the most likely spoken sentence using statistical methods, given an observation sequence, is described formally in this section, following the description in [21] and [17].

Given an input $O$, what is the most likely sentence $W'$ out of all sentences in language $L$? More precisely, let

- $o_i$ be a 10 ms slice of the input and
- $O = o_0, o_1, o_2, ..., o_{T-1}$ be a sequence of input slices,
- $w_i$ be a word,
- $W = w_1, w_2, w_3, ..., w_n$ be a sequence of words,
- $L$ be the language used,
- and $V$ be the vocabulary in $L$.

The problem of finding the most likely word sequence $W'$ can in a naive, but accurate, way be found by iterating through all possible sentences $W$ that can be produced from $V$. The sentence $W$ with the highest probability $P(W|O)$, i.e. the probability that $W$ is the correct transcript of $O$, can be considered the best choice for $W'$. This can be seen in Equation 2.8, which also holds for single words.

$$W' = \arg \max_{W \in V} P(W|O)$$

(2.8)

It is generally hard to compute $P(W|O)$ [17]. This is because all possible word sequences $W$ must be tested, which would require much resources, however there is a trick that can be used. Bayes’ rule states that $P(W|O) = \frac{P(O|W)P(W)}{P(O)}$. This means we can replace Equation 2.8 by Equation 2.9.

$$W' = \arg \max_{W \in V} \frac{P(O|W)P(W)}{P(O)}$$

(2.9)

$P(O)$ is hard to estimate according to [17], however it is possible to ignore $P(O)$. For all sentences that are compared in Equation 2.9, $P(O|W)$, $P(W)$ and $P(O)$ will be calculated, where $P(O)$ never changes between the different sentences [17]. The probability $P(O)$ will therefore be the same for all possible sentences. $P(O)$ can therefore be seen as a normalizing factor, which makes it possible to ignore it. This gives the function described in Equation 2.10.

$$W' = \arg \max_{W \in V} P(O|W)P(W)$$

(2.10)

What is left are two terms. $P(W)$ is known as the language model, or prior probability [17], and $P(O|W)$ is known as the acoustic model or likelihood [17]. $P(W)$ is the probability of a word string $W$ occurring, and $P(O|W)$ is the probability that a word string $W$ generates the observation $O$. 
2.4.2 Forward Search

Given a so-called pronunciation network, representing a word $W$, the forward algorithm makes it possible to compute the probability that $W$ matches an observation sequence $O$.

The forward algorithm takes as input a pronunciation network for each candidate word. A pronunciation network is a weighted automaton or Markov chain, where a node consists of a phoneme and the edges contain the probability to transit from one node to another. The algorithm generates the probability matrix called $forward$. This matrix contains probabilities for the possible paths in the pronunciation network. The output is the sum of the probabilities in the final column of $forward$.

To find which of the candidate words is most probable given the observation sequence $O$, we need to compute the product $P(O|w)P(w)$ (see Equation 2.10) for each candidate word.

**Pseudo algorithm**

The algorithm described in Algorithm 1 is based on the one in [17], where $a[s,s']$ is the transition probability from current state $s$ to next state $s'$, see Equation 2.2, and $b[s',o_t]$ is the observation likelihood of $o_t$ given $s$, see Equation 2.4.

```plaintext
Data: observations, state_graph

N ← NUMBER_OF_STATES(state_graph)
T ← length(observations)

Create probability matrix $forward[N + 2, T + 2]$ $forward[0,0] ← 1.0$

for each time step $t$ from 0 to $T$ do
  for each state $s$ from 0 to $N$ do
    for each transition from $s$ to $s'$ (specified by state_graph) do
      current_cell_tmp ← $forward[s,t] * a[s,s'] * b[s',o_t]$
      $forward[s',t+1] ← forward[s',t+1] + current_cell_tmp$
    end
  end
  forward_probability ← 0
  for each probability $p$ in final column of $forward$ do
    forward_probability ← forward_probability + $p$
  end
return forward_probability

Algorithm 1: The Forward algorithm
```

The forward algorithm calculates the probability for each of the possible paths through the network. It then summarizes the probabilities of the paths that leads to an accepting node.
For each time step $t$ (representing each observation), for each state $s$ of the network, calculate the probability of getting to all next states $s'$. This is done by multiplying the probability of the current state with the transition probability $a[s, s']$ and the observation likelihood $b[s', o_t]$. The calculated value is stored in $forward[s', t]$. When all time steps are done (all observations are processed), the sum of the last column is calculated and returned.

An example probability matrix for one word 'need', where the probability $P(O|w)$ is calculated, can be seen in Table 2.1.

**Forward Algorithm Example**

The following example is taken from [17].

**Figure 2.4**: The word “need” represented as a HMM. Note that this is a simplification where the states contain phonemes instead of feature numbers. This is only for demonstrative purposes. A real HMM representing the entire word would contain too many states and transitions to be reasonable for demonstration.

**Figure 2.5**: The word “neat” represented as a HMM. Note that this is a simplification where the states contain phonemes instead of feature numbers. This is only for demonstrative purposes. A real HMM representing the entire word would contain too many states and transitions to be reasonable for demonstration.

The words need and neat are represented as two separate HMMs, which can be seen in Figure 2.4 and Figure 2.5. Both of these are simplified in the
sense that the states contain phonemes and not feature numbers as they do in real HMMs.

Notice that both words in Figure 2.4 and Figure 2.5 can be recognized with the phonemes ”n” and ”iy” as input and in that order.

As input to the example we will have the observation sequence $O = ”#”, ”n”, ”iy”, ”#”$, where ”#” denotes start and end of the observation sequence. This can either be recognized as the word ”need” or the word ”neat”, when considering the HMMs in Figure 2.4 and 2.5.

We assume for this example that the probabilities for the words ”need” and ”neat” are already calculated from a corpus and that the probabilities $P(w)$ for the words are the following. These probabilities can be generated by calculating how frequently the words appear in the corpus.

\[
\begin{align*}
P(”need”) &= 0.00056 \text{ (probability of the word ”need”)} \\
P(”neat”) &= 0.00013 \text{ (probability of the word ”neat”)}
\end{align*}
\]

<table>
<thead>
<tr>
<th>$o_0$ = ”#”</th>
<th>$o_1$ = ”n”</th>
<th>$o_2$ = ”iy”</th>
<th>$o_3$ = ”#”</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>n</td>
<td>iy</td>
<td>end</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0 * 1.0 = 1.0</td>
<td>1.0 * 1.0 = 1.0</td>
<td>1.0 * 0.11 = 0.11</td>
</tr>
</tbody>
</table>

Table 2.1: Matrix of output results from the Forward Algorithm for the word ”need”, based on the HMM in Figure 2.4, showing the probability $P(O|w)$ in the rightmost column on the last row. Each state in the HMM is represented on a single row, and each column represents the observations or time $t$.

Table 2.1 should contain the probability $P(O|w)P(w)$ instead of just $P(O|w)$ as shown. The starting probability is 1.0 for time $t = 1$, and for each observation $o_t$, the probability $P(o_0, o_1, o_2, ..., o_t, s_i$ at $t|w)$ is calculated and stored in cell $forward[s_i, t]$, where $s_i$ is the $t$th state.

From Table 2.1 we can see that the probability $P(O|w)$ for the word ”need” is 0.11 and therefore the probability $P(O|w)P(w)$ for the word ”need” is $0.11 \times 0.00056 = 0.0000616$. The equivalent probability for the word ”neat” is $P(O|w)P(w) = 0.52 \times 0.00013 = 0.0000676$. This means that in this case, given the observation $O = ”#”, ”n”, ”iy”, ”#”$ and the state graphs of the words ”need” and ”neat”, the most likely word would be ”neat” because it has the highest probability.

**Problems with the Forward Algorithm**

Using the Forward algorithm as a decoder is inefficient, since all possible words need to be considered and the algorithm must be run on each word to know which word has the highest probability and thus being the best match. When
Table 2.2: Matrix of output results from the Forward Algorithm for the word "neat", showing the probability $P(O|w)$ in the rightmost column on the last row. States can be seen in the leftmost column.

<table>
<thead>
<tr>
<th>State</th>
<th>$o_0 = &quot;#&quot;$</th>
<th>$o_1 = &quot;n&quot;$</th>
<th>$o_2 = &quot;iy&quot;$</th>
<th>$o_3 = &quot;#&quot;$</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>1.0</td>
<td>1.0 * 1.0 = 1.0</td>
<td>1.0 * 1.0 = 1.0</td>
<td>1.0 * 0.52 = 0.52</td>
</tr>
<tr>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iy</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

using 50000 words or more, as is very common in continuous speech recognition, this is not a sufficiently fast method. [17]

The example in the previous section is also simplified, as in a real system, the states in the HMM would represent feature numbers instead of whole phonemes, meaning that the HMM would be much larger in a real system. When considering complete sentences instead of single words as in this example, the inefficiency would be even more noticeable. Another simplification is that the observation likelihood $b[s, o_t]$ is also limited to be either 1 or 0.

The Viterbi algorithm, as described in Section 2.4.3, is a better solution as it makes it possible to consider all words at the same time and also compute the most likely path.

### 2.4.3 Viterbi search

The goal of the Viterbi algorithm is to find the best state sequence $Q = q_0, q_1, ..., q_{T-1}$ given the observed phonemes $O = o_0, o_1, o_2, ..., o_{T-1}$ and a model $\lambda$. The Viterbi Algorithm makes it possible to find the best path through an automaton, as well as the probability of the observation sequence.

In [6], the Viterbi Algorithm is described as finding the shortest path through a graph, formalized as follows.

$\hat{s}(q_t)$ is defined as the survivor for $q_t$ and refers to the shortest path between $q_0$ and $q_t$.

$Length(q_t)$ is the length of the survivor (length of shortest path for $q_t$).

$Length(q_{t+1}, q_t) = Length(q_t + \text{distance}(q_t, q_{t+1}))$ calculates the length for $q_{t+1}$ by taking the shortest distance for $q_t$ and adding the distance between $q_t$ and $q_{t+1}$.

$Length(q_{t+1}) = \min_{q_t} Length(q_{t+1}, q_t)$ finds the shortest path for $q_{t+1}$, by iterating through all states in next timestep ($t + 1$). The length of the shortest path for $q_{t+1}$ is stored in $Length(q_{t+1})$, and the survivor (shortest path) is stored in $\hat{s}(q_{t+1})$. 
By calculating $\text{Length}(q_{t+1})$ over all times $t$, we find the best paths to all states $q_t$, and thus the minimum path through the graph.

The Viterbi algorithm seen in Algorithm 2 is based on the algorithm found in [17] and performs the same calculations as described above with the main difference that we now calculate probabilities instead of finding the shortest path. The same idea applies, with the difference that we are now interested in finding the highest value instead of the lowest. The calculated values are stored in the matrix $\text{Viterbi}$, and the best path, or survivor can be found in $\text{back_pointer}$. The algorithm takes a single weighted automaton and an observation sequence $O = o_0, o_1, o_2, ..., o_{T-1}$ as input, and returns the most probable state sequence $Q = q_0, q_1, q_2, ..., q_{T-1}$ together with its probability.

The weighted automaton is in the form of a HMM, where the transitions (edges) consists of the transition probabilities between the phonemes (see also Figure 5.18 in [17]).

**Pseudo algorithm**

**Data:** Observations $O$ of length $T$, state_graph

$N \leftarrow$ number of states in state_graph

Create a path probability matrix $\text{Viterbi}[N + 2, T + 2]$

For all $s$ and $t$, initialize $\text{Viterbi}[s, t]$ with 0

Create a matrix back-pointer $\text{back_pointer}$

For all $s$ and $t$, initialize $\text{back_pointer}[s, t]$ with $s$

$\text{Viterbi}[0, 0] \leftarrow 1.0$

for each time step $t$ from 0 to $T$

for each state $s$ from 0 to $N$

for each transition from $s$ to $s'$, specified by state_graph do

new_score $\leftarrow \text{Viterbi}[s, t] \times a[s, s'] \times b[s', o_t]$

if new_score $>$ $\text{Viterbi}[s', t + 1]$ then

$\text{Viterbi}[s', t + 1] \leftarrow$ new_score

$\text{back_pointer}[s', t + 1] \leftarrow s$

end

end

end

Backtrace from highest probability state in the final column of $\text{Viterbi}$ using $\text{back_pointer}$ and return path

**Algorithm 2:** The Viterbi algorithm

A state graph (automaton) is often a representation of single phonemes, but to simplify explanation, a state graph represents a word where the states in the automaton are phonemes of the word.

The Viterbi algorithm can solve both the segmentation problem (determination of where words start and end) and the decoding problem (determine
which word produces an observation sequence).

If multiple words are to be recognized, the automata for the words must be combined into one single automaton. To create this combined automaton, bigram probabilities (2-grams) are used to connect two words in the automaton. These bigrams are calculated from a corpus where a bigram probability is the probability that one word follows another word.

The Viterbi algorithm is very similar to the forward algorithm with the difference that back-pointers are saved for each cell and only the maximum of the previous paths is saved, in comparison to the forward algorithm where the sum of the previous paths is saved.

**Example of Viterbi algorithm**

This example is based on an example in [17].

In Figure 2.6 the separate words for "I", "need", "the", "on" can be seen. In Figure 2.7, a simplified HMM of the combined words can be seen that will be used in this example, where the probabilities of the bigrams are also visible, shown as dotted lines between the end of a word and the start of another. For example the bigram "I need" has the probability 0.0016.

![Figure 2.6: The words "I", "need", "the", "on" represented as HMMs. Note that the HMMs are simplified, in the sense that the states contain phonemes instead of feature numbers. This is done to make it easier to explain the algorithms. Note also that 2.6b is the same as Figure 2.4.](image)

The input to use for the example is the observation sequence $O = \#\text{"aa"\"n"\"iy"\"d"\#}$, where "#" represents the start and end of the observation. $O$ represents the input sentence "I need". Note, as in the
forward search, that $b[s', o_t] \in \{1, 0\}$.

Cases where $b[s', o_t]$ is zero are not included in this example. In other words,

Figure 2.7: The words “I”, ”need”, ”the”, ”on” represented as combined HMM, using bigrams to represent the probabilities between words. Note that ”#” represents the beginning of a sentence, meaning that sentences that start with ”I”, for example, has the initial probability 0.079.
only cases for which the total probability is non-zero are presented below. 

\( Viterbi[s, t] \) represents the path probability for a state \( s \) at time \( t \).

\( a[s, s'] \) represents the transition probability between states \( s \) and \( s' \) in the HMM.

States in the HMM are below represented as "phoneme"\_word, e.g. "aa"\_i represents phoneme "aa" in word "i".

Note that for the transition probability \( a[s, s'] \), multiple values may sometimes be necessary if the HMM is represented as in Figure 2.7. This happens for example when one word transitions to another, which includes the N-gram probability into \( a[s, s'] \).

Each step below presents the value of \( Viterbi[s, t] \) (previous path probability), \( a[s, s'] \) (the transition probability between \( s \) and \( s' \)) and the calculated value \( new \_score \) \( (Viterbi[s', t + 1]) \) as presented in Algorithm 2.

<table>
<thead>
<tr>
<th></th>
<th>( o_0 = &quot;#&quot; )</th>
<th>( o_1 = &quot;aa&quot; )</th>
<th>( o_2 = &quot;n&quot; )</th>
<th>( o_3 = &quot;iy&quot; )</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;aa&quot;_i</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;ay&quot;_i</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;aa&quot;_on</td>
<td>0.0016</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;n&quot;_on</td>
<td></td>
<td>0.00077</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;dh&quot;_the</td>
<td></td>
<td></td>
<td>2.3 * 10(^{-8})</td>
<td></td>
</tr>
<tr>
<td>&quot;n&quot;_the</td>
<td></td>
<td></td>
<td></td>
<td>2.8 * 10(^{-9})</td>
</tr>
<tr>
<td>&quot;iy&quot;_the</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;ax&quot;_the</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;n&quot;_need</td>
<td></td>
<td></td>
<td>0.00000256</td>
<td>0.00000256</td>
</tr>
<tr>
<td>&quot;iy&quot;_need</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;d&quot;_need</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.3: Matrix of output results from the Viterbi Algorithm example, using the HMM in Figure 2.7, showing the probability \( P(O|w)P(w) \) in the rightmost column on the last row. Each row represents a state in the HMM, where the states are grouped for each word. Each column represents an element in the observation sequence.
Steps taken to get to the matrix in Table 2.3:
Start probability is always 1.0

\[ \text{time } t = 0, \text{ state } s = "\#", \text{ state } s' = "aa"_i \]
\[ V_{\text{Viterbi}}[s, t] = 1.0 \]
\[ a["\#", "aa"_i] = 0.2 \ast 0.079 = 0.0016 \]
\[ V_{\text{Viterbi}}["aa"_i, 1] = 1.0 \ast 0.0016 \ast 1.0 = 0.0016 \]

\[ \text{time } t = 0, \text{ state } s = "\#", \text{ state } s' = "aa"_{on} \]
\[ V_{\text{Viterbi}}[s, t] = 1.0 \]
\[ a["\#", "aa"_{on}] = 0.00077 \ast 1.0 = 0.00077 \]
\[ V_{\text{Viterbi}}["aa"_{on}, 1] = 1.0 \ast 0.00077 \ast 1.0 = 0.00077 \]

\[ \text{time } t = 1, \text{ state } s = "aa"_{on}, \text{ state } s' = "n"_{on} \]
\[ V_{\text{Viterbi}}[s, t] = 0.00077 \]
\[ a["aa"_{on}, "n"_{on}] = 1.0 \]
\[ V_{\text{Viterbi}}["n"_{on}, 2] = 0.00077 \ast 1.0 \ast 1.0 = 0.00077 \]

\[ \text{time } t = 1, \text{ state } s = "aa"_i, \text{ state } s' = "n"_{need} \]
\[ V_{\text{Viterbi}}[s, t] = 0.0016 \]
\[ a["aa"_i, "n"_{need}] = 1.0 \ast 0.0016 \ast 1.0 = 0.0016 \]
\[ V_{\text{Viterbi}}["n"_{need}, 2] = 0.0016 \ast 0.0016 \ast 1.0 = 0.00000256 \]

\[ \text{time } t = 1, \text{ state } s = "aa"_i, \text{ state } s' = "n"_{the} \]
\[ V_{\text{Viterbi}}[s, t] = 0.0016 \]
\[ a["aa"_i, "n"_{the}] = 1.0 \ast 0.00018 \ast 0.08 = 0.0000144 \]
\[ V_{\text{Viterbi}}["n"_{the}, 2] = 0.0016 \ast 0.0000144 \ast 1.0 = 0.000000023 \]

\[ \text{time } t = 2, \text{ state } s = "n"_{need}, \text{ state } s' = "iy"_{need} \]
\[ V_{\text{Viterbi}}[s, t] = 0.00000256 \]
\[ a["n"_{need}, "iy"_{need}] = 1.0 \]
\[ V_{\text{Viterbi}}["iy"_{need}, 3] = 0.00000256 \ast 1.0 \ast 1.0 = 0.00000256 \]

\[ \text{time } t = 2, \text{ state } s = "n"_{the}, \text{ state } s' = "iy"_{the} \]
\[ V_{\text{Viterbi}}[s, t] = 0.00000023 \]
\[ a["n"_{the}, "iy"_{the}] = 0.12 \]
\[ V_{\text{Viterbi}}["iy"_{the}, 3] = 0.000000023 \ast 0.12 \ast 1.0 = 0.000000028 \]

If we stop after \( O_3 = "iy" \), the highest probability seen in the rightmost column (\( t = 3 \)) will be 0.00000256 representing the sentence "i need".

In the end, there will be different probabilities in the rightmost column. The cell with the highest probability is used for the backtracking. In the example, we stopped after "iy", and there we had the possible words "i need" and "i the", where "i need" had the highest probability, and thus "i need" is the path that is returned by the algorithm.
Simplifications of Viterbi

The Viterbi algorithm described is a simplification of the real algorithm. A real HMM used for the algorithm, would not contain phonemes, but feature vectors (containing spectral and acoustic features). The HMMs would therefore be much larger and more complex than the ones provided as examples. The states would be representations of sub-phonemes.

Since it would be very expensive to consider all the possible word combinations when the algorithm is extending the path from one state-column to the next, in reality the paths are pruned when they are considered to have low enough scores. The path with the best score is extended first, and paths with low scores are pruned, resulting in states being skipped for each time step. This is also known as beam search.

Restrictions of Viterbi

The Viterbi algorithm does not compute the most likely sequence of words, but computes the most likely sequence of states (phonemes). These two may or may not be different [17].

The Viterbi algorithm can be modified to return multiple potential answers (N-best list) and using other algorithms to allow for a higher-level search using higher orders of N-gram models to find the best utterance. This is known as multi-pass decoding.

Another solution is to use another algorithm such as the A* decoding algorithm also known as Stack decoding algorithm. [17]

2.4.4 A* decoder / Stack decoder

The A* decoding algorithm, or Stack decoding algorithm, is better than the Viterbi algorithm in the sense that it not only allows for bigram language models, but practically any N-gram language model [17]. Douglas B. Paul [20] has implemented and run the algorithm using trigrams. The A* decoding algorithm is also dependent on the forward algorithm for calculating the probabilities of each word.

The stack decoding algorithm has been used by Julius in version 3.2 and has achieved a recognition rate of 90%3 when using a vocabulary of 20000 words [12].

A* works with scores and not with probabilities to determine the best match. The acoustic probabilities and language model probabilities are converted into a score by using the A* evaluation function: \( f^*(p) = g(p) + h^*(p), \) \( p \) being the path. A more detailed explanation of the computation of scores is given on page 25.

A* builds up a tree of possible paths of words to create the sentence. This tree is built by using the language model and the N-gram probabilities.

\(^3\text{http://www.ar.media.kyoto-u.ac.jp/members/ian/doc/system.html}\)
In Figure 2.8 an example representation of allowable word sequences of a language can be seen. This is very simplified and only covers a subset of the allowed sentences. The entire tree of all possible sentences is never completely generated at the same time, since it would require very much space, but is built up as the algorithm proceeds. A trigram language model that describes the sentences in Figure 2.8 can consist of (among others) the following sample trigrams:

- \{"Start", "I", "can" \}
- \{"Start", "the", "dog" \}
- \{"I", "want", "the" \}
- \{"can", "the", "dog" \}

A* works by having a priority queue of the leaves of the paths considered so far. The queue is sorted after the highest scoring paths. When a new word is to be considered, the path with the highest score will be removed from the queue and expanded. The resulting new paths are added to the queue.

Figure 2.8: Representation of word sequences allowed in an example language. The figure only presents a few words of a language and only four levels deep for demonstration purposes. A complete network of allowable word sequences would be very large.
Pseudo-algorithm

initialize the priority queue with the null sentence (start node)

\begin{algorithm}
\textbf{while not done do}
\begin{algorithmic}
\State get the path $p$ with the highest score from the queue \textbf{if} $p$ \textbf{is end of sentence then}
\State return $p$
\EndIf
\EndWhile
\State Find list of likely next words using \textit{fast-match}
\For {each candidate next word $w$}
\State create new candidate sentence (path) $p' = p + w$
\State compute acoustic likelihood $A$ of $p'$ using the forward algorithm
\State compute language model probability $L$ of $p'$
\State compute score for $p'$
\State insert $p'$ into the queue
\EndFor
\EndFor
\end{algorithmic}
\end{algorithm}

\textbf{Algorithm 3: The Stack Decoder algorithm}

The algorithm described in Algorithm 3 is based on the algorithm found in [17]. The algorithm described in [17] however lacks one small but important detail, that everything after the initialization of the queue must be looped. Another description of the algorithm is found in [20] where this error does not exist, by using a goto at the end of the algorithm. In the algorithm described in Algorithm 3, this is solved with a while-loop with the same result.

Even though the stack decoding uses the forward algorithm, which would be slow if calculating all possible sentences, this is not a problem because of the language model which limits the search field in combination with fast matching and pruning.

Fast Match

$A^*$ uses fast match to find the most likely next words. This can be done by storing the phonemes of words in a tree-structure, which means that words with the same initial phonemes partly share the same path in the tree [17].

Computation of score

The algorithm requires computation of the score of a sentence. This is done by using the $A^*$ evaluation function, $f^*(p) = g(p) + h^*(p)$, $p$ being the current path of words(sentence) where $g(p)$ represents the score from the start of the sentence to the current position, and $h^*(p)$ represents the score from the current position and to the end of the utterance.

The computation of $g(p)$ is calculated from the values of $A$ and $L$ (the acoustic and language model probabilities $P(O|p)P(p)$). Since the score should
be as high as possible, one cannot take the probability $P(O|p)P(p)$ directly since this number will become lower for longer sentences. If that probability were to be used as the score, the best sentence would only contain one word, i.e. the first word since shorter sentences would have higher scores.

The calculation of $h^*(p)$ is still considered an unsolved problem to which only estimates are computed, where one approach is to base the score on the number of words remaining in the sentence [17]. Since the score $h^*(p)$ only is an estimate, the scoring cannot be guaranteed to be optimal. This means that the stack decoder algorithm is not guaranteed to be optimal as well.

Algorithm example

Input to the example is a sequence of observations that corresponds to the sentence "i want pie", using the limited amount of sentences in Figure 2.8.

The fast matching may produce a word list that corresponds to the words in the first column after "Start" in Figure 2.8. The word "Start" following each of these words will now form a path. For each of these paths, a score is calculated as can be seen in Figure 2.9.

![Figure 2.9: First iteration of the Stack decoding algorithm, based on the example network in Figure 2.8 and with fictional scores. The most likely sentence is "the" at this point.](image)

The sentence "the" is the most likely sentence at this point, which means that "the" is the node to be expanded. For all possible words that the fast matching returns (and that the language model allows), new paths are built
and new scores are calculated for the leaf nodes. This can be illustrated as in Figure 2.10 where it can be seen that the highest scoring node has changed from "the" to "i".

Since the leaf node "i" is the sentence with the highest score at this point, "i" is the node that will be expanded next. The result of this expansion can be seen in Figure 2.11 where it can be seen that the highest scoring sentence is "i want". The algorithm continues this way until no more input is available, or an end of utterance is detected. At this point, we have achieved the best path.

Figure 2.10: Second iteration of the Stack decoding algorithm continuing the example in Figure 2.9, with continued fabricated scores. The most likely sentence is "i" at this point.
Figure 2.11: Third iteration of the Stack decoding algorithm, continuing the example from Figure 2.10, with continued fabricated scores. The most likely sentence is ”i want“ at this point.
Chapter 3

Speech Recognition Systems and Libraries

There are some libraries available that are reasonable candidates for the evaluation. The main requirement for the libraries is that they should be able to recognize large vocabulary continuous speech. Relevant libraries are described in this section, together with some information about their performance from other research findings. A comparison of the different libraries can be seen in Section 3.8. The ones used in the evaluation are CMU Sphinx, Microsoft SAPI and Dragon NaturallySpeaking.

A valuable resource for the evaluation is VoxForge\(^1\). It provides a database with audio files and with corresponding text, which makes it possible to generate acoustic models to be used with a recognizer. VoxForge provides acoustic models for CMU Sphinx, Julius and HTK, which can be downloaded from the VoxForge web page.

Most Acoustic Models used by Open Source Speech Recognition engines are closed source, according to VoxForge. These closed source acoustic models do not provide access to the speech audio (the source) used to create the Acoustic Model, since they often are licensed. Often it is possible to get access to these corpora by purchasing a license. One license provider is the Linguistic Data Consortium\(^2\). This is one large reason to why VoxForge exists, to make it possible to have access to a free corpus that can actually be used in any type of product. VoxForge is a good starting point when generating a custom acoustic model.

---

\(^1\)[http://www.voxforge.org/]

\(^2\)[http://www.ldc.upenn.edu/]

29
3.1 CMU Sphinx

CMU Sphinx is an open source speech recognition toolkit, containing both a speech recognition engine and an acoustic model trainer. The speech recognition engine exists in two versions, one written in C (PocketSphinx, Sphinx3) and one written in Java (Sphinx4)\(^3\). The one written in C is chosen for the evaluation. The CMU Sphinx toolkit was developed at Carnegie Mellon University as the name suggests.

CMU Sphinx also contains acoustic model training tools. This can be used to adapt an acoustic model to improve accuracy or to create an entirely new acoustic model, for example for a new language or a new dialect.

Sphinx4 is a port of Sphinx3, and is said to be more flexible, however they perform equally well in word error rate (WER, see Section 4.1.5) according to the documentation\(^4\), which is why it should not be a problem to evaluate Sphinx3 instead of Sphinx4. In the documentation it is claimed that both Sphinx3 and Sphinx4 get a WER of 18.8% using a large vocabulary language model of 64000 words. The language model used in that performance test is based on the corpus HUB4\(^5\), which consists of broadcast news reports. The performance test does not mention anything about which audio was actually tested with to get to 18.8%. One could however assume that the HUB4 test set\(^6\) has been used. The WER are accompanied by the real time factor (RTF, see Section 4.1.5) which is 3.06 for Sphinx3 and 3.95 for Sphinx4.

Keith Vertanen has made some tests [29] using PocketSphinx, Sphinx2 and Sphinx3 using a vocabulary of 60000 words from the English Gigaword corpus evaluated with the San Jose Mercury sentences from the WSJ1 Hub 2 test set, containing 207 sentences. The results from these tests showed a WER of 31.1% for Sphinx3, 39.8% for Sphinx2 and 38.8% for PocketSphinx, which could all be achieved with an RTF lower than 1. Vertanen also managed to lower the WER to around 22%, and suggests that it eventually can be lowered even more, however this caused the RTF to climb to above 1.5 or even up to 2.5 in some cases. Having stated both the training set and the test set, it would be possible to reproduce these results, however comparing the results with the Sphinx4 documentation one can draw the conclusion that the results are consistent with each other. The fact that the RTF lies between 3 and 4 from the Sphinx4 documentation can justify the low WER since this is what Keith Vertanen suggests: if RTF is allowed to be higher than it may be possible to lower the WER.

CMU Sphinx has the advantage of being available on Linux, Windows and Mac OS X. There exist acoustic models, language models and dictionaries for this library that are available for download. It is also possible to create and use other models and dictionaries.

\(^3\)http://cmusphinx.sourceforge.net/wiki/tutorialoverview
\(^4\)http://cmusphinx.sourceforge.net/sphinx4/\#speed_and_accuracy
\(^5\)http://www.ldc.upenn.edu/Catalog/catalogEntry.jsp?catalogId=LDC98S71
\(^6\)http://www.ldc.upenn.edu/Catalog/catalogEntry.jsp?catalogId=LDC2000S88
3.2 Julius

Julius\(^7\) is an open source continuous speech recognition software, which, similar to CMU Sphinx, is available for Linux, Windows and Mac OS X.

The recognition accuracy for Julius mostly depends on the models used in the recognition process, since Julius basically consists only of a decoder. [13]

Julius uses HMMs for acoustic models and for language models, both N-gram probabilistic models and grammars can be used [13]. Algorithms used are the Viterbi algorithm, beam search and stack decoding.

Japanese language models and acoustic models exist for free download from the Julius web page. There also exist English models, however these are not allowed to be used in the development or tests of commercial products\(^8\). Acoustic models are available from VoxForge\(^9\).

Finding performance results for Julius proved to be difficult for the English language. For Japanese however, word accuracy of 58.74% (implies 41.26% WER) have been noted when using a 20000 word vocabulary size [28]. This article focuses on combining the results from multiple systems to improve the confidence level of the recognition. No RTF were published in this report which makes the results a bit meaningless, unless it is assumed that they lie below 1.0.

An article describing an overview of Julius [14] claims that Julius can perform large vocabulary continuous speech recognition in real time "effectively". This does not say much, as effectively is a very subjective word in this situation. An article by Lagatie et al. [10] claims that Julius, HTK and CMU Sphinx perform very similar in accuracy, however does not back this up with any data.

Considering the different sources above, without any actual data to back it up, it is not possible to confirm that Julius performs equally good as CMU Sphinx and HTK for the English language.

3.3 Kaldi

Kaldi\(^10\) is a toolkit for speech recognition, and is primarily used by speech recognition researchers. Kaldi contains methods for training and using neural networks for the acoustic models.

Kaldi as a speech recognizer achieves a WER of 12.5% to 18.3% when trained using the WSJ corpus with a 20000 word vocabulary, and with the test sets WSJ Nov’94 and WSJ Nov’95, and also performs slightly worse than HTK when compared. [22]

\(^7\)http://julius.sourceforge.jp/en_index.php
\(^9\)http://www.repository.voxforge1.org/downloads/Main/Tags/Releases/
\(^10\)http://kaldi.sourceforge.net/about.html
3.4 HTK

HTK\textsuperscript{11} (Hidden Markov Model Toolkit) is a tool for building and manipulating Hidden Markov Models and is primarily used for speech recognition research. HTK provides training tools for acoustic models, which can be used for example in the Julius recognizer.

HTK is allowed to be used for academic and commercial product research, but it is not allowed to redistribute parts of HTK. Any model trained using HTK can be used for commercial products.

3.5 Windows Speech Recognition

Windows Speech Recognition is a feature available in Windows Vista, Windows 7 and Windows 8. These use the Microsoft SAPI (Speech Application Programming Interface)\textsuperscript{12}, with which it is possible to write applications that make use of speech recognition as well as text-to-speech.

The Microsoft Speech API contains a "dictation grammar" which can be used for continuous speech recognition. It is also possible to use custom created language models such as N-gram based language models\textsuperscript{13}.

Microsoft SAPI has achieved a WER value of 32.1\% with a real time factor of 0.55 when run on the San Jose Mercury sentences from the WSJ Hub 2 test set containing 207 sentences and using a language model based on the corpus English Gigaword with 60000 word vocabulary size [30].

3.6 Dragon NaturallySpeaking

Dragon NaturallySpeaking is the successor of Dragon Dictate\textsuperscript{14}, and is one of the most well-known speech recognition software on the market today. DragonDictate made use of Hidden Markov Models, indicating that Dragon NaturallySpeaking may also use them.

Dragon NaturallySpeaking is a speech recognition software developed for Windows operating systems only, which also provides an SDK for developers.

Dragon has achieved a WER of 23.9\% with a real time factor of 0.49 on the San Jose Mercury sentences from the WSJ Hub 2 test set and using a language model based on the English Gigaword corpus with 60000 word vocabulary size [30].

In [5], French dictation using Dragon NaturallySpeaking has achieved a WER of 11.7\% when adaptation of the recognition domain was made to the subject of involvement of Canadian troops in the Iraq war. In [5] it is worth

\textsuperscript{11}htk.eng.cam.ac.uk/
\textsuperscript{12}http://msdn.microsoft.com/en-us/library/ee125663(v=vs.85).aspx
\textsuperscript{14}http://www.cl.cam.ac.uk/a2x-voice/dd-faq.html#What
noticing that the audio input came directly from the mouth of users, which would most likely result in different WER if the same test was done again.

One disadvantage for Dragon NaturallySpeaking is that it costs money. For the most simple version of the software, the cost is $99.99 at the official Nuance online store. The price reaches many hundred dollars for the more advanced versions\(^\text{15}\).

**Algorithms**

A stack decoding algorithm has been developed by Dragon Systems, Inc.\(^\text{[26]}\). This was however in the year 1989 and as Dragon NaturallySpeaking is closed source, it is not possible to know all the details about how the speech recognition actually works.

### 3.7 SpeechMagic

SpeechMagic is a speech recognition platform with focus on healthcare. It was developed by Philips Speech Recognition Systems but is now, as well as Dragon NaturallySpeaking, owned by Nuance\(^\text{16}\).

### 3.8 Comparison of libraries

<table>
<thead>
<tr>
<th></th>
<th>CMU</th>
<th>Julius</th>
<th>Kaldi</th>
<th>Dragon</th>
<th>SAPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Source</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cross-platform</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Models (English)</td>
<td>yes</td>
<td>?</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>Model Customization</td>
<td>yes</td>
<td>yes</td>
<td>?</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>speaker dependent</td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>large vocabulary</td>
<td>yes</td>
<td>41.3% (jap)</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>WER</td>
<td>18.8%</td>
<td>12.5%</td>
<td>23.9%</td>
<td>32.1%</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1: Comparison of libraries. WERs as described in Section 3 are used to make an estimated comparison between the libraries.

In the comparison table seen in Table 3.1, it is worth noticing that Dragon NaturallySpeaking allows for individual speaker training, which does not mean that the library is completely dependent on individual training before it can be used, but rather that the system can be tweaked to better recognize a specific person. It shall therefore be seen as an advantage and not a disadvantage.

Another thing to notice are the WER values. The numbers presented in Table 3.1 are not achieved using the same testing methods. Results for CMU \(^\text{15}\)http://shop.nuance.com/  
Sphinx, Dragon NaturallySpeaking and Microsoft SAPI are basically obtained using the same corpus and testing data. Julius and Kaldi were tested using different training and testing data. The WER for Julius seems may seem high, especially when considering that these results come from a test with a mere 20000 word vocabulary. However these results arose from recognizing the Japanese language instead of English. Therefore it is not fair to do a direct comparison with the other results.

Kaldi having a WER of 12.5% is based on a 20000 word vocabulary, whereas CMU Sphinx, Dragon NaturallySpeaking and Microsoft SAPI were tested with a 60000 word vocabulary, which can explain the lower values for Kaldi.
Chapter 4

Evaluation of Existing Systems and Libraries

The main question to be answered is which speech recognition library is the best for Codemill’s situation and how high accuracy can be achieved using as much currently existing material as possible. The most suitable of these libraries should also be used in Vidispine.

The libraries that were supposed to be in the test were CMU Sphinx, Julius, Kaldi, Microsoft Speech API and Nuance Dragon NaturallySpeaking. Speech-Magic was assessed to focus on a too specific area, which is why it was not considered to be part of the evaluation. The libraries that finally were part of the test are CMU Sphinx (PocketSphinx), Microsoft Speech API, Nuance Dragon NaturallySpeaking and with Youtube Automatic Captions used for comparison. The reason why Julius and Kaldi was dropped from the test is that they were problematic to get starting with, as we will see later in Section 4.2.

Custom acoustic models or language models were not produced for the tests, but available ones were used.

4.1 The evaluation test

Since the performance of the libraries was to be evaluated, a common test protocol had to exist so that the libraries could be tested on equal conditions. Since PocketSphinx, Dragon NaturallySpeaking and Microsoft SAPI are the main focus on the evaluation and since Keith Vertanen has performed tests on these libraries using the 207 sentences in the San Jose Mercury set from WSJ, the same set would be a good candidate to base the evaluation on. However, to get access to these, a fee of $2500.00\textsuperscript{1} must be payed, which was not available for this project. As an alternative to the San Jose Mercury test set, a new test

\textsuperscript{1}http://www.ldc.upenn.edu/Catalog/catalogEntry.jsp?catalogId=LDC94S13A
could be developed using audio or videos that are available for download, where one alternative would be to use audio from the VoxForge corpus. Considering the goals of the project, using VoxForge could end up with a test containing very varying contents as well as a set of sentences that are incoherent with each other. Instead, the decision to base the test on TED talks was made, since this would result in a test containing more continuous speech, compared to VoxForge where the sentences come in separate files, not necessarily related to one another. The contents of TED talks do of course vary much as well from video to video, however limiting the number of videos in the test could solve this problem. Using TED talks for the evaluation will give results that better represents actual usage, considering the Vidispine platform, compared to the other alternatives, making it the best alternative.

### 4.1.1 Limitations on test

The evaluation only contains videos where the English language is spoken. The accent is american English since this is the most likely target accent of an eventual product.

The videos used for the test are speeches and presentations with mostly one person talking at the same time. This will give a rough idea of how the system will function when used in the real world with the same type of videos.

All tests are run on the same hardware, and are specified in Table 4.1. The operating systems used for the evaluation are Linux Mint 12 32 bit and Windows 7 32 bit.

<table>
<thead>
<tr>
<th>Processor</th>
<th>2 GHz AMD Turion RM-70</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>3072 MB</td>
</tr>
<tr>
<td>Graphics card</td>
<td>ATI Mobility Radeon HD 3450 Graphics</td>
</tr>
</tbody>
</table>

Table 4.1: Hardware used in the evaluation

The hardware is not state-of-the-art by today’s standards, which is why it can be argued that all libraries will perform faster than what can be seen in the evaluation results if modern hardware is used.

### 4.1.2 Test Input

The videos used for the evaluation were chosen so that there were equally many women and men talking, having roughly equal length for women and men. The speech in the videos were also required to be English with an American accent.

The tests were performed using 8 different videos from TED talks. Each video contains one speaker where four of the videos are men talking and the other four are women. The length sums up to a total of roughly 45 minutes for the men, and 37 minutes for the women and contains more than 1800 lines of subtitles of actual speech (approximately 800 sentences), which is more than
4.1. The evaluation test

the 207 sentences used in the tests by Keith Vertanen. This should be enough to determine the quality of the recognition.

The videos used for the evaluation are listed in Appendix A. In this list the video IDs (used by TED), the speaker, the gender of the speaker and the length of the video is presented. From now on, if the videos are mentioned in the report individually, they are addressed by their ID from this list.

The original subtitles for the videos are used when comparing the output of the recognition and calculating the measurement values. Since the original subtitles contain additional information such as timestamps and in some cases notifications of audience laughter and applauds, the extra information is removed before comparison.

Since the speech recognition libraries take audio as input, the videos were converted to audio in a format that the libraries can handle. The audio from the videos was converted to the format PCM mono channel, 16bit, 16000 Hz, which was done by using FFmpeg with the following command.

```bash
ffmpeg -i video.mp4 -f s16le -acodec pcm_s16le -ac 1 -ar 16000 video_audio.raw
```

4.1.3 Problems

A problem that is very obvious concerning the input files is that there are applauds (and other audience-related sounds) that can be heard in most of the TED-talk videos. This can interfere with the recognition process, especially if these noises occur at the same time as the speech. The noise can be interpreted as speech, or it can alter the recognition of the actual speech before and after the noise. This is unfortunate since it most likely has had an effect on the evaluation.

One solution to this could be to cut out the parts where this type of sound exists and to only focus on the clean parts of the video. This however will introduce sentence cut-offs which will also reduce the recognition accuracy, unless only whole sentences are removed with the effect that short noise could cause an entire sentence to be removed, reducing the test set. None of these solutions were applied to the videos.

4.1.4 Output

The output of the recognizers are saved to separate files for each test. By comparing the original text with the output of the recognizer, the values word error rate (WER) and real time factor (RTF) are calculated, which are explained in Section 4.1.5.

One problem with the output of the different libraries is that it is not formatted in a standardized way. For example, PocketSphinx only outputs lowercase letters, whereas Microsoft SAPI and Dragon NaturallySpeaking can also output capital letters. This can cause problems in the calculation of the metrics
used in the comparison. To solve this, all text is normalized by converting all letters to lowercase and by removing special characters if they exist. We do not want to conclude that the words "Hello" and "hello" are different words.

Another output is the time for the recognizer to run, which is measured from the start of the speech recognition to the end of the speech recognition. This will be considered when calculating the real time factor for the tests.

Since no punctuation is present in the output, except for contractions in some cases such as "don’t" and similar, it is not possible to know where a sentence starts and ends. In general this is a difficult problem because of the fact that humans can make very long pauses but still be in the same sentence. The opposite is also true, a very short pause can occur between sentences, which makes it difficult to determine where sentences start and end solely by determining pauses. Another solution to this is to use higher level algorithms to deal with the grammatical buildup of sentences to determine the most likely cut-offs.

### 4.1.5 Metrics

The metrics used to compare the different libraries are discussed here. The most common metric in speech recognition is **word error rate (WER)** and **word accuracy rate (WACC)** [17]. WER is defined in Equation 4.1, where INS is the number of inserted words, DEL the number of deleted words, SUB the number of substituted words and EXP is the number of expected words. WER represents a number that describes the overall proportion of incorrectly recognized words of the recognizer. Since WACC represents the opposite (1 - WER) of WER and WER is more common than WACC, WACC is not included in the results.

According to [16], values for WER when using a large vocabulary (telephony) speech recognition lie around 20% (2012).

State-of-the-art systems have WER of 15% (2005), when a lot of training and manually customized dictionaries are involved [19].

\[
\text{WER} = \frac{\text{INS} + \text{DEL} + \text{SUB}}{\text{EXP}}
\]  

(W.1)

WER should be as low as possible, for the results to be considered as good as possible.

Precision and recall are values that are used sometimes when measuring speech recognition performance, and both of these should be as high as possible for the recognition to be considered as good as possible. Both of these values are excluded in this evaluation since WER is much more common.

The **Real Time Factor (RTF)** is the ratio between the time it takes to process the input, and the actual length of the input and is defined in Equation 4.2, where TTP is the time the recognizer used and LEN is the (temporal) length of the video. This number is a very useful measurement in cases where it is important to be able to do live recognition. If this number is
1 or below, the recognition was done faster than the length of the video, and can therefore be performed on live video.

\[ RTF = \frac{TTP}{LEN} \]  

(4.2)

4.1.6 Calculation of metrics

A script was written that compares the expected output (formatted real subtitles) to the output from the recognizer and calculates the relevant information such as WER. This script makes use of the program \textit{wdiff} in Linux. From this program it is possible to acquire the number of inserted words, the number of deleted words and the number of substituted words. Using these numbers, the number of correct result words are calculated.

The metrics are used in the results of the evaluation which can be seen in Section 4.3.

4.2 Practicalities using the libraries

In this section, how the libraries were used is explained, together with some problems that arose with them.

4.2.1 CMU Sphinx

CMU Sphinx had a couple of candidate test libraries, namely PocketSphinx or Sphinx4. Sphinx4 is written in Java and PocketSphinx is written in C, they are however said to be performing equally well. PocketSphinx was chosen since it would fit better into the system if this were to be chosen to be implemented into Vidispine. For PocketSphinx, a sample program was written in C. The program is based on the code used for the \textit{pocketsphinx-continuous} application that comes with PocketSphinx. The code was modified to simplify the usage of input and output and also for linking the models needed for recognition.

There existed both an acoustic model and a language model, which both were based on the HUB4 corpus, that was used for the test. According to CMU Sphinx\(^2\) the acoustic model that exist are good enough, and should only be adapted if necessary. Should an attempt to produce one from scratch be made, it would probably turn out worse than the one provided considering the time frame for this project.

When doing some random first test of PocketSphinx with the audio files that were generated for the test, the recognizer stopped working for some of the files, or did not work at all. After some error message searching on the internet and on the CMU Sphinx IRC channel, the problem was identified to be caused by the input files. They should have a silent section in the beginning of the file,

\(^2\)\url{http://cmusphinx.sourceforge.net/wiki/tutorialam}
so that PocketSphinx can calibrate for silence. By updating PocketSphinx to the latest version from the repository, this specific problem was eliminated.

Another problem with PocketSphinx, which was related to the "silence-start"-problem, was that in some of the files, no pauses in the speech could be found in the file, and therefore the internal buffer of PocketSphinx would grow too large and therefore cause an overflow. This was easily recognizable from the error messages and since the buffer is cleared each time a recognition is made on the current buffer-data, this problem was solved by forcing a recognition if too much data was pushed to PocketSphinx. Another solution to this problem could be to fine-tune the settings for silence recognition. Using the second solution however could cause other problems, such as the risk of breaking in the middle of a sentence. A long pause in the speech does not necessarily mean a pause between sentences (which is what is most often wanted for best accuracy), it may only be a way for the speaker to emphasize different parts of the speech, or simply just a way to make sure the audience get a chance to reflect on what is being said. A break in the middle of a sentence may cause that the surroundings words of the breaking point are falsely recognized, causing unnecessary recognition errors.

4.2.2 Microsoft Speech API

For Microsoft Speech API, a sample program was written in C# using Visual Studio 2010 in Windows 7. This program made use of the class DictationGrammar, which allowed the program to recognize continuous speech.

Microsoft SAPI can output text that contains capital letters. Therefore the output was normalized by replacing capital letters into lowercase letters.

The class "DictationGrammar" was available to use for the speech recognition. This class is used for free text dictation.\(^3\)

4.2.3 Dragon NaturallySpeaking

For Dragon NaturallySpeaking, no source code was written, but the graphical user interface was used as input of the audio. This was done because there were problems getting the license for Dragon NaturallySpeaking in time. Therefore there were also very limited access to the API at that time. To save time at the end, the user interface was finally utilized instead of writing a program. The input test files were therefore inserted to Dragon NaturallySpeaking through the GUI. The output was transferred to a text editor and finally saved to files. The standard settings were used:

- Language: US English
- Speech Model: BestMatch IV
- Base Vocabulary: General - Large

\(^3\)http://msdn.microsoft.com/en-us/library/ee125663(v=vs.85).aspx
4.2. Practicalities using the libraries

Dragon NaturallySpeaking insists that a user profile is setup before the speech recognition features are used. A real attempt to try to train it to a specific voice was not made since there are eight different speakers in the test, having Dragon NaturallySpeaking start under the same conditions as the other libraries.

In Dragon NaturallySpeaking, there exist different types of accents that can be used, but since the test videos are English with US-accents, the accent chosen was US.

One problem encountered when running the tests, was that the output text sometimes transitioned into strange formats. In the second half of one test, the output became all capital letters. In another test after around half of the test, each first letter of each word was capitalized. In a third test, words were not separated with a space, but just concatenated. These tests were re-run and the output looked normal. This behavior was very odd, and no explanation has been found for this.

4.2.4 Kaldi

An attempt at installing Kaldi was made, but there were problems on the way. First, there were no packages to be downloaded through the package manager in Linux Mint 12, meaning the source code had to be downloaded from the Kaldi web page. The source code was downloaded using svn commands described in the installation guide. When the source code was to be compiled, there were errors that said that ”ATLAS” needed to be installed. When trying to install ”ATLAS” there were further problems, with error messages complaining about CPU throttling.

An attempt to disable CPU throttling was made with the command cpufreq-set, however this only resulted in the same error message. After struggling with this for many hours without any progress, this library was ignored and put to a lower priority and was not included in the evaluation.

4.2.5 Julius

Julius managed to install correctly and an example program was run with a very simple grammar. The sample program could recognize words from an input file that was sent to the program. Testing of Julius looked very hopeful until it became apparent that there are no available English language models for Julius. An attempt was made to create a language model using the steps provided in the HTK Book which is said to produce a language model that is compatible with Julius. The language model was created but there were problems when trying to use it in Julius. Julius produced many error messages of a type that has to do with the loading of the dictionary.

\[^4\text{http://kaldi.sourceforge.net/install.html}\]
\[^5\text{http://htk.eng.cam.ac.uk/docs/docs.shtml}\]
According to some forums, this problem means that words that exist in the language model do not exist in the acoustic model, and that the only solution to this problem is to train an acoustic model.

On the web page of Keith Vertanen\(^6\), a solution to the language model-problem was found and was supposed to work with HTK and CMU Sphinx recognizers. The instructions were pretty straightforward and a model was generated, however similar error messages as mentioned earlier occurred when trying to use them.

After these problems and after much time was spent on trying to generate the language model, and to avoid the risk of wasting more time, Julius was put to a lower priority.

### 4.2.6 Youtube ASR

Youtube have an automatic speech recognition feature, where a video that lacks original transcriptions can still have subtitles. Since most TED-talks exist on Youtube, and these are the types of videos that are used in the test, a comparison with the Youtube ASR was also made.

### 4.3 Evaluation Results

The produced results for the evaluation is presented in the following sections. The different libraries will in the rest of the report have shortened names in the tables and figures because of limited space. In Table 4.2 these names are presented.

Since the video with ID 1592 did not exist on Youtube, no comparison numbers exist for this video in the tables and diagrams.

<table>
<thead>
<tr>
<th>Full library name</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft SAPI</td>
<td>SAPI</td>
</tr>
<tr>
<td>PocketSphinx</td>
<td>PS</td>
</tr>
<tr>
<td>Dragon NaturallySpeaking</td>
<td>Dragon</td>
</tr>
<tr>
<td>Youtube ASR</td>
<td>Youtube</td>
</tr>
</tbody>
</table>

Table 4.2: The library names and their abbreviations used for Tables and Figures from this point in the report.

### 4.3.1 Produced Words

The number of produced words is the sum of all words that the library gave as output, where the optimum would be to generate exactly as many words as in the original transcript, which was a total of 14151 words. It can be seen

\(^6\)http://www.keithv.com/software/
in Table 4.3 that PocketSphinx resulted in the closest number of produced words with only a couple of hundred words off. From the three tested libraries, Microsoft SAPI produced fewest words. On a side note, Youtube produced fewest words for each individual video.

<table>
<thead>
<tr>
<th>Video</th>
<th>Expected words</th>
<th>SAPI</th>
<th>PS</th>
<th>Dragon</th>
<th>Youtube</th>
</tr>
</thead>
<tbody>
<tr>
<td>1584</td>
<td>1428</td>
<td>1323</td>
<td>1535</td>
<td>1168</td>
<td>1107</td>
</tr>
<tr>
<td>1592</td>
<td>657</td>
<td>596</td>
<td>699</td>
<td>504</td>
<td></td>
</tr>
<tr>
<td>1594</td>
<td>3093</td>
<td>524</td>
<td>3169</td>
<td>2559</td>
<td>2281</td>
</tr>
<tr>
<td>1603</td>
<td>865</td>
<td>777</td>
<td>871</td>
<td>707</td>
<td>683</td>
</tr>
<tr>
<td>1604</td>
<td>2161</td>
<td>1919</td>
<td>2126</td>
<td>1631</td>
<td>1336</td>
</tr>
<tr>
<td>1607</td>
<td>1798</td>
<td>1425</td>
<td>1829</td>
<td>1659</td>
<td>1244</td>
</tr>
<tr>
<td>1608</td>
<td>878</td>
<td>713</td>
<td>827</td>
<td>703</td>
<td>545</td>
</tr>
<tr>
<td>1612</td>
<td>3271</td>
<td>2970</td>
<td>3218</td>
<td>2906</td>
<td>2154</td>
</tr>
<tr>
<td>Total</td>
<td>14151</td>
<td>10247</td>
<td>14274</td>
<td>11837</td>
<td>9350</td>
</tr>
</tbody>
</table>

Table 4.3: Expected word count and produced words for the different libraries and Youtube.

In Figure 4.1 the number of produced words can be seen for each video in the test. While most of the test runs resulted in somewhat reasonable numbers, one exception can be seen with Microsoft SAPI for the video with ID 1594, where only 524 words were produced in a video where 3093 words were spoken. See Section 4.3.4 for a possible explanation of this behaviour.

![Produced words](image)

Figure 4.1: The number of produced words for each video and library. For each video, the results from the different libraries are grouped, the first column in each group being the number of words in the original transcript.
Libraries that produced a small number of words could deliberately be more careful in the recognition process and only output words above some limit of confidence. Since PocketSphinx produced most words, this could mean that PocketSphinx gives an answer no matter how the recognition turned out. The opposite can be said about Youtube. Microsoft SAPI and Dragon NaturallySpeaking did not follow the same pattern. Dragon NaturallySpeaking produced fewer words, and got better WER, than PocketSphinx, but produced more words, and still got a better WER than Microsoft SAPI.

### 4.3.2 Word Error Rate

The definition of Word Error Rate can be seen in Equation 4.1, and the results from the tests can be seen in Table 4.4 and in Figure 4.2.

It can be seen that the average word error rate was best for Dragon NaturallySpeaking and worst for Microsoft SAPI. Dragon NaturallySpeaking even beat Youtube in terms of word error rate.

<table>
<thead>
<tr>
<th>Video</th>
<th>MSSAPI</th>
<th>PocketSphinx</th>
<th>Dragon</th>
<th>Youtube</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>0.608</td>
<td>0.599</td>
<td>0.426</td>
<td>0.450</td>
</tr>
<tr>
<td>Women</td>
<td>0.731</td>
<td>0.619</td>
<td>0.485</td>
<td>0.387</td>
</tr>
<tr>
<td>Men</td>
<td>0.486</td>
<td>0.579</td>
<td>0.367</td>
<td>0.497</td>
</tr>
</tbody>
</table>

Table 4.4: Word Error Rate total average, women average and men averages for the different libraries. Averages for Youtube takes into account that one value is missing.

Microsoft SAPI got a bad result on the women videos. This is because of the problems with video 1594. However, looking at the other women results for Microsoft SAPI, the average would still get a worse result than PocketSphinx with 0.34.

In Figure 4.2 it can be seen that Microsoft SAPI performs worst on average because of the women videos, and performed better than Youtube on the men videos.

### 4.3.3 Real Time Factor

Real Time Factor is defined in Equation 4.2 and values below 1 makes it possible to do the speech recognition live on the video. The results from the tests can be seen in Table 4.5 and Figure 4.3.

It can be seen that Microsoft SAPI used least time of all libraries, with under 0.5 in real time factor on average. Dragon and PocketSphinx used about the same time on average, where Dragon was a little bit faster. Youtube does not exist in these results since the subtitles for the videos already exists and are stored on Youtube’s servers, and thus does not need to be generated. The time required to generate Youtube automatic subtitles is therefore unknown.
4.3. Evaluation Results

Figure 4.2: Average Word Error Rate for the different libraries and Youtube. The total average is presented in blue, the average for women in red and the average for men in orange.

All libraries except Microsoft SAPI used more time for the women videos compared to the men videos.

4.3.4 Analysis of the results

Results from research of different algorithms are often discussed around the test results without confirming that the difference is significant [7]. To be able to tell if two tests reveal a significant difference is something that is ignored in many benchmarking tests according to [7], where two methods to determine this significance is presented. The first, McNemar’s test, is best suitable for isolated words, whereas the second, Matched pairs test (t-test), is better suited for connected speech. The matched-pairs test performs better for a larger sample size.

<table>
<thead>
<tr>
<th></th>
<th>SAPI</th>
<th>PS</th>
<th>Dragon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>0.43</td>
<td>0.68</td>
<td>0.66</td>
</tr>
<tr>
<td>Women</td>
<td>0.38</td>
<td>0.77</td>
<td>0.76</td>
</tr>
<tr>
<td>Men</td>
<td>0.48</td>
<td>0.60</td>
<td>0.57</td>
</tr>
</tbody>
</table>

Table 4.5: Real Time Factor total average, women average and men average for the different libraries.
A paired t-test was run in Octave\textsuperscript{7}, using the WER from the recognition results as sample data. The difference between the WER is assumed to follow a normal distribution.

When running the paired t-test on the WER for all videos for libraries PocketSphinx and Dragon NaturallySpeaking, the results were $H = 1$, and $p < 3.2308 \times 10^{-4} < 0.01$, meaning we can reject the null hypothesis. Thus, the difference is significant with a confidence interval of \(99\%\). Running Dragon NaturallySpeaking against Microsoft SAPI also produces $H = 1$, but with $p < 0.0085 < 0.01$, which means that we can in this case also reject the null hypothesis.

Running the same paired t-test for PocketSphinx and Microsoft SAPI, the results were $H = 0$ and $p = 0.8716$, meaning we can not reject the null hypothesis. In other words, we cannot say that there is a significant difference between the two libraries with a confidence interval of \(99\%\).

From this is it possible to say that Dragon NaturallySpeaking is significantly better in the regard of WER for the specific test, compared to the other two libraries.

Microsoft SAPI had some problems with video 1594. This can be clearly seen in the result data. In number of produced words (Figure 4.1) it can be seen that only around one out of six expected words were outputted. The test aborted prematurely, even after a restart of the test. The RTF for the same video show a decrease as well as can be seen in Figure 4.4, indicating that

\footnote{http://octave.sourceforge.net/nan/function/ttest.html}
4.3. Evaluation Results

![Real Time Factor](image)

Figure 4.4: Real Time Factor for the separate videos of the tests for the different libraries. The first four bars in each group (library) are the female videos, whereas the last four bars are the male videos.

very little time was used for running the test. The reason for this abortion is unknown. The break happens at time 4:05 in the input file where a pause in the speech is made, and after almost a second of silence the speaker makes a kind of smacking sound with the mouth. This might be interpreted by the recognizer as a microphone being disconnected or something similar, causing the recognition to stop. However, similar sounds occur earlier in the video, which are processed without problem, which makes this theory farfetched. It would be strange if it was a problem with the actual audio file since all other files are formatted in the same way and the file worked in the other recognizers without problems. The most likely explanation is that the recognizer interpretes the input data as faulty somehow.

Dragon NaturallySpeaking had good results in general with best average word error rate and with a low RTF. Dragon NaturallySpeaking most likely has the best algorithms and techniques for speech recognition out of the libraries tested, considering the results. The average WER was actually lower than that of Youtube, which is impressive considering that the tests were run on a laptop and that no special voice training was done. Dragon NaturallySpeaking does have the advantage of experience from other systems such as the Philips Speech Recognition Systems, and it also has the funding to keep specializing in speech recognition, considering Nuance’s revenue of $1,118.9 million for the fiscal year 2010.
Comparison to Keith Vertanen’s tests

The TED test is limited to 8 speakers in total with a total audio length of 83 minutes and contains more sentences than the test used by Keith Vertanen, meaning that the test size should be sufficient to provide a good estimate of how good the recognizers perform.

The main difference between the San Jose test set compared to the TED test set is that the former contains mostly news reports from the Wall Street Journal, whereas the TED videos contain a more diverse set of speech. In the San Jose test set, there are most likely none or very little noise as compared to the TED videos.

In [30], Keith Vertanen focused on the differences between hyperarticulation and normal speech for Microsoft SAPI, Dragon NaturallySpeaking and HTK, and does not compare the libraries against each other. In [29], he compares the different Sphinx libraries and mostly discusses the differences in speed, but mentions that there is a marginal difference between the libraries.

In Table 4.6, a comparison between the results that were achieved in this report, using the eight videos described in Appendix A, and the results that Keith Vertanen got using the San Jose Mercury sentences from the WSJ1 HUB 2 test set, is presented. While these two tests cannot be compared directly, because of different setups, it is possible to draw some conclusions about the two. First we can see that the WER results from the TED evaluation are around 18.7 – 28.7 percentage points higher than the results of Keith Vertanen. This is most likely because of the test sets used and the fact that Keith both trained and tested the systems with the Wall Street Journal (WSJ) corpora. This makes the system more specialized in news speech recognition, in contrast to especially Microsoft SAPI which contain a more general dictation model. TED talks can vary more in context from video to video which can explain the worse results, since the training data and testing data can contain text with different types of content, whereas news stories tend to have more reoccurring words and sentence buildup.

<table>
<thead>
<tr>
<th></th>
<th>SAPI</th>
<th>PS</th>
<th>Dragon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keith Vertanen</td>
<td>32.1%</td>
<td>38.8%</td>
<td>23.9%</td>
</tr>
<tr>
<td>TED evaluation</td>
<td>60.8%</td>
<td>59.9%</td>
<td>42.6%</td>
</tr>
</tbody>
</table>

Table 4.6: WER results from this evaluation (TED evaluation) and results of Keith Vertanen. Note that the results are achieved using different methods.

Considering that PocketSphinx was tested on the broadcast news HUB4 acoustic model in Vertanen’s tests, having access to that same test data would most likely have given better results. It may also have helped to have had the TED test set, but to have trained the acoustic models with other TED videos. This however is not a guarantee that the recognition will be better since there are so many different talkers in TED videos, and the subjects of the videos are
so diverse. It may however have helped to recognize certain words that are specific to the TED talks-domain.

The sound quality may be more consistent in the San Jose Mercury sentences than that of the TED videos, which could affect the outcome of the recognition. A high quality source with as little unwanted noise as possible would be ideal. In the TED videos there are some applauds, laughter and other audience noises which can be falsely recognized as speech by the recognizer. This is of course a problem for this type of videos which can explain a bit about the results.

When comparing the actual output from the recognizers it was noticed that the recognized words for each video starts with ”it and new soon” using PocketSphinx. It is safe to assume that this is because of the identical start sequence of TED videos are being recognized as words. The same pattern could be seen with Microsoft SAPI where all recognitions started with the word sequence ”he had planned protest proof”, which indicates that Microsoft SAPI also identifies the starting jingle as words. For the results from Dragon NaturallySpeaking this pattern was not as clear as the other two. Four of the videos (in fact all female speaker videos) started with the word sequence ”he is are you”, which is definitely a pattern. Another video started with the word sequence ”he he is a you”, which is basically the same but with an extra ”he” in the very beginning. Another video started with the word sequence ”he is are you” which is very similar as well, and one video started with the word sequence ”he is”. When looking at the Youtube results, it was found that three of the videos started with the word ”four”, but that was all obvious resemblance that could be found.

From these observations it can be concluded that noises in the video indeed affects the outcome of the results from the recognizers. Even Dragon NaturallySpeaking indicated problems with this. Youtube seems to have this problem under control and these observations can explain the relatively low word error rates. In spite of this Dragon NaturallySpeaking managed to yield a better WER.

Noise can therefore be seen as a large problem since it seems to interfere a lot with the recognition. The presence of this type of noise would indicate that TED videos should get worse results than the WSJ test set, since the recognizers become confused from this.

We can see in Table 4.6 that Dragon NaturallySpeaking is the winner in both Vertanens as well as the TED-test case, whereas PocketSphinx performed worst if looking at Vertanens test. Microsoft SAPI was last in the TED test but was not much worse than PocketSphinx. Here it is worth mentioning that the WER for Microsoft SAPI is increased by the problems with the video with ID 1594, where the recognition stopped prematurely, which is why Microsoft SAPI would have showed off better in the results. Considering this, the results actually show the same winner and loser for both tests, with Dragon as the winner, Microsoft SAPI at second place and PocketSphinx at last place, but with a difference of roughly 20 percentage points.
Women vs men

Looking at the average WER in Figure 4.2, it can be seen that the videos with female talkers, the result is worse than for the videos with male talkers. Looking at Figure 4.3 we can also see that PocketSphinx and Dragon show higher RTF for the women. This means that, for these libraries, it took longer time to process the videos with female talkers, nevertheless getting worse results. In [1] it is suggested that women voices are easier for ASR systems to recognize, having better (up to 7% better) WER. These tests were made on broadcast news in both English and French and also concluded that female speakers speak more clearly and that male speakers more often use careless pronunciation.

To determine if the difference between men and women are significant, Octaves $ttest2$ was run for the WER of the men-videos and the women-videos for each library. This test is suitable when we want to know if there is a significant difference between two different sample sets (men vs women). The results from these tests can be seen in Table 4.7.

<table>
<thead>
<tr>
<th>Library</th>
<th>h</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS</td>
<td>0</td>
<td>0.4792</td>
</tr>
<tr>
<td>Dragon</td>
<td>0</td>
<td>0.1011</td>
</tr>
<tr>
<td>SAPI</td>
<td>1</td>
<td>0.016</td>
</tr>
<tr>
<td>Youtube</td>
<td>1</td>
<td>0.0404</td>
</tr>
</tbody>
</table>

Table 4.7: The ttest2 results for the difference in WER between women and men for the different libraries

The results seen in Table 4.7 suggests that we cannot say that the difference between female vs male recognition is significant for PocketSphinx and Dragon NaturallySpeaking. We can however say that for Microsoft SAPI and Youtube, the difference is significant with a confidence interval of 95%. This means we can say that for Microsoft SAPI, the speech recognition works better for men using the TED test videos defined in Appendix A. For Youtube we can say that the speech recognition works better for women with a confidence interval of 95%. The results get more reliable with more data and we assume that the WER difference between men and women follow a normal distribution.

One thing that is worth mentioning about the tests are that the women talker videos actually are a few minutes shorter in total than the videos with men talking, however this does not affect the real time factor. It will however affect the word error rate a little bit, considering that the TED jingle in the start of each video will be a greater part of the total recognition, having the same number of videos, with equally long jingles but with shorter total time.

The average WER says that Youtube recognizes women better than it recognizes men, and the best female video even beat all the men videos for Youtube. Youtube may have algorithms or models that work better with female voices, or quite possibly multiple algorithms that are run in parallel. It is possible that many techniques are used side by side to achieve the results that Youtube got.
4.3. Evaluation Results

Because there are differences in female and male voices, to be able to improve the recognition it is useful and also common to separate the models used for male and female speech recognition [11]. This however requires that male and female voices can be distinguished or the gender of the speaker is known to the system. It is likely that Dragon NaturallySpeaking and Youtube use this kind of technique to get better accuracy. The same argumentation can be said for having separate models for different types of videos.

When listening to the quality of the sound of the videos, it can be heard that video ”1594” has generally worse sound quality than the other videos. This could be the reason why Microsoft SAPI fails on that specific video. For PocketSphinx, this video has the highest RTF and for Dragon NaturallySpeaking this video has the second highest RTF. This could indicate that the recognizers struggle very hard for this video. However, looking at the other female videos, Dragon NaturallySpeaking has almost equal RTF for all of the female videos, while most of the male videos have much lower values. Since the RTF is so stable over the female videos for Dragon NaturallySpeaking, this can be interpreted as the quality of the audio does not matter a lot for that specific library. Requiring more time but having less accurate results could mean that the specific female voices of the test videos are harder to recognize.

Women in general have higher pitch voices and this may be a cause for differences in the recognition. The training data for the acoustic models could contain more male voices than female. If the models are trained more with male voices, intuition says that male voices should be more easily recognized.

If we look at the speed of the talks, women talked at about 162 words per minute, and the men talked at about 176 words per minute. It can therefore be assumed that the speed of the talk does not matter very much, unless talking slower actually means that recognition becomes worse. If this were the opposite, one could have argued that talking faster could mean more problems for the recognizer, since more information would have to be squeezed into the same space of time.

Time

Looking at the RTF of the recognition for the different libraries could tell a lot about the efficiency of the algorithms and models. Microsoft SAPI had the lowest RTF. This is somewhat affected by the fact that one of the tests did not finish properly. PocketSphinx and Dragon NaturallySpeaking had about the same RTF with a small advantage to Dragon NaturallySpeaking. Considering this and the fact that Dragon NaturallySpeaking got a better WER than PocketSphinx, one can draw the conclusion that Dragon NaturallySpeaking gives a better result, using roughly the same amount of time.
Chapter 5

Integration of Speech Recognition in Vidispine

In this section, the implementation of the prototype to use with Vidispine is explained together with why the specific library was chosen and how the library is integrated into Vidispine.

5.1 Selection of library

After looking at the results from the test results it can be seen that from the three tested libraries, Dragon NaturallySpeaking is the library that performed the best when it comes to using standard settings and models and in terms of WER. Despite of that fact, the choice of library for Vidispine went to PocketSphinx. There are several reasons for this, the first one being the fact that PocketSphinx is open source and does not cost money, whereas Dragon NaturallySpeaking is the complete opposite. The other reason is that the system, in which the recognizer was to be implemented, was mainly written in C++ and mainly runs on Linux operating systems, which gave a big advantage to PocketSphinx, since Dragon NaturallySpeaking and Microsoft SAPI only runs on Windows operating systems. After a short discussion with Vidispine, it was decided that PocketSphinx seemed like a good choice. PocketSphinx can also be used in commercial products1.

As we will se later in Section 6, it is possible to improve the language models to match the specific area of the videos with the result that the accuracy becomes much better. The custom language model used for PocketSphinx resulted in a lower WER than using standard settings in Dragon NaturallySpeaking.

1http://www.speech.cs.cmu.edu/sphinx/doc/Sphinx.html
5.2 Vidispine

Vidispine can be seen as a middleware, meant to be used by other applications. Vidispine contains many different modules that interact with each other. One part of Vidispine is the transcoder, where most of the implementation for this project takes place.

In the Vidispine transcoder, a job is defined as an XML file that describes which filters and methods shall be used on a video. Vidispine then sets these filters and modules up and makes sure that they are run in the order they are required. A graph representation of this can be seen in Figure 5.1. The module written can be seen at an endpoint in this graph with the name ”Automatic Speech Recognizer”.

Vidispine supports conversion between different audio and video formats, so it is assumed that the audio is delivered in the correct format when the library is integrated into Vidispine.

5.3 ASR module

PocketSphinx is integrated into the transcoder. This is done by adding a new module called AutomaticSpeechRecognizer which can be seen in Figure 5.1. The module gets an audio stream as input and does not alter the stream in any way, which is something that is normal for other modules to do, especially if they act as a filter. However, the meta-data that is generated is saved to a subtitle file, as well as uploaded to a Vidispine-server where it becomes searchable.

![Figure 5.1: Vidispine module graph. Visualization of how the modules in Vidispine work together.](image)

The written module has one main method that takes audio data as input. The input is assumed to be in the correct format, that is 16000Hz, 16 bit, mono channel. This method is called automatically from Vidispine and the data is then processed by the module. The module sends the incoming data to the PocketSphinx library. When a certain amount of data has been transferred, the function get_hypothesis is run on the PocketSphinx object, which activates the speech recognition for the buffered data. The string that is returned is saved to a subtitle file of type SRT (SubRip), and also sent as metadata to a Vidispine server, where it becomes searchable.

A sub-class called Utterance was written as a helper class to deal with the breakdown of the video clip. This class also has a buffer, into which the incoming data is put. The length of an Utterance is determined by the number
of buffers that are allowed per utterance. The number of buffers is determined from an input parameter called bufferLength, which contains the maximum length of video data to be recognized at any time. When the buffer is full, it is sent to the PocketSphinx recognizer. When the number of buffers required for an utterance is reached, a request for the recognized sentence is made from PocketSphinx. This sentence is then saved to a .srt file together with a timestamp and utterance number. The sentence is also sent to the Vidispine server together with the timestamp.

Data: inputdata
utterance ← new utterance()
if inputdata is start then
  utterance ← newutterance()
end
for each sub_data in data do
  if current buffer is full then
    Add buffer to PocketSphinx
    if maxBuffers is reached then
      finalize utterance
      get and save recognizer result
      utterance ← new utterance()
    end
  end
end
buffer ← sub_data
if data is endOfData then
  finalize utterance; save recognizer result;
end

Algorithm 4: Vidispine speech recognition module algorithm

Algorithm 4 describes how the module for Vidispine works on a high level.

![Module overview. Simple overview of the speech recognition module functionality.](image_url)
The recognition works by taking a small chunk at a time, running the recognizer on that chunk, and then moving on to the next chunk. This means that between all these chunks, there is a “restart” of the recognition, which causes the last data of the previous recognition to not be considered when running the recognizer on the current data. This means that for all those breaks, there will be a decrease in accuracy. By having a decent enough silence detection, this could be avoided by assuming that a silenced part is the end of an utterance. By buffering this way, there will be natural breaks instead of breaks that would occur in the middle of utterances, or even in the middle of words. This problem can be seen illustrated in Figure 5.3.

![Figure 5.3: Illustration of the cut-off problem, where each break in the input data causes a disturbance of the recognition.](image)

Another solution to this problem is to run the recognition without silence detection, but with an overlap of the chunked recognition. By overlapping it is possible to map the results together and thus remove this problem. This has the cost of requiring that the same data will be recognized multiple times, which may not be acceptable in situations where time is limited, since this would require more time for the entire recognition process. With small enough chunks this would mean that in worst case the recognition would require twice as much time, since all data must be recognized twice. It would also require a higher-level algorithm that can match the overlap with each other as to avoid double information. This solution is illustrated in Figure 5.4.

![Figure 5.4: Illustration of the overlap solution for the cut-off problem. By recognizing some of the data multiple times, and then matching the overlap (darker green), a complete recognition can still be achieved.](image)

A third solution would be to buffer as much as possible and to let Pocket-
Sphinx run on as much data as possible. This would limit the recognition of live video, since many minutes could be buffered at any given time, but would eliminate the problems with sentences being cut.

To be able to search for words in a video, the spoken words must be associated with a timestamp. This timestamp is generated from the number of bytes that have passed through the module, converted into the number of seconds that have passed. Since we know the format of the input data, this is not difficult to calculate, and can be seen in Equation 5.1. The metadata was at first stored as a .srt file because this was easier to start with (just saving the information to a file). Later, the data was also generated as XML. The metadata consists of a timestamp interval and the spoken words between those timestamps. As soon as an utterance is finalized, the meta-data is stored locally and sent to the server.

\[
\text{timestamp} = \frac{\text{videoframe}}{\text{videoframespersecond}}
\]  

(5.1)

5.4 Prototype application

The module described above will be used in the middleware Vidispine and functions as backbone code only. A simple prototype client application that makes use of the generated data from the module was also written. To make a prototype that can search for meta-data, the following criteria must be fulfilled.

- The meta-data must contain references to the video, which interval in the video it relates to and what was spoken in that interval.

- The generated meta-data must be stored and made accessible.

The module generates the meta-data and stores it on a server. The communication with the server is done by using HTTP POST, PUT and GET. This is utilized in two simple stand-alone programs where one is made for uploading of a video and the other is made for searching meta-data. These two programs serve as a proof-of-concept, which shows that it is possible to generate the meta-data and to have it searchable.

The upload program sends a video URI to the transcoder which causes the created module to be run on that video. This generates the meta-data, which is sent to the server and then becomes available on the Vidispine server.

Since the meta-data can be accessed using RESTful communication, the communication with the server is not limited to a single programming language. Since Java is easy to work with, this was chosen as the language to use for the prototype application. This application is a proof-of-concept that provides an example of how the meta-data can be used. This prototype has a simple GUI that contains a search field together with a search button. With this it is possible to do a search for a specific word. Below this is a list of search results, that shows a video ID together with a start time and an end time. On the side
of the search field and the list is a video player. If a search result is double-clicked, the video at the start time will be displayed in the video player. This can be seen in Figure 5.5.

When a search is made, an XML string is created containing the search word information that was written in the text field. The prototype establishes a connection to the meta-data-server, and the XML is sent to the server using HTTP (REST). The server returns an XML structure containing the results of the search. From the results, it is possible to get the path to the video files where the word exists, and also the specific timestamps in the videos.

The actual video files paths that are returned from the server are local paths for this prototype but can easily be modified to deal with any URI.

Figure 5.5: Screenshot of the prototype application written to demonstrate the possibilities of the model. Here a search for the word “world” was made, having a set of search results to the right, with the chosen video displaying to the left.
Chapter 6

Improvements of PocketSphinx

Since PocketSphinx was chosen as the library to use for Vidispine (see Section 5.1), the improvements are done solely on PocketSphinx. In order to improve the accuracy for the recognition, the language model was the main target, but it is also possible to modify the acoustic model. From this chapter and forward the different changes are named as follows. PocketSphinx with the standard models as stated previously in the report is abbreviated as PS, the replacement of the dictionary is abbreviated as PS DICT, the replacement of the language model is abbreviated as PS LM, and the replacement of both the dictionary and the language model is abbreviated as PS LM DICT.

The WER values for the different changes can be seen in Table 6.1.

<table>
<thead>
<tr>
<th></th>
<th>PS</th>
<th>PS DICT</th>
<th>PS LM</th>
<th>PS LM DICT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>0.599</td>
<td>0.610</td>
<td>0.398</td>
<td>0.395</td>
</tr>
<tr>
<td>Women</td>
<td>0.619</td>
<td>0.626</td>
<td>0.420</td>
<td>0.415</td>
</tr>
<tr>
<td>Men</td>
<td>0.579</td>
<td>0.594</td>
<td>0.376</td>
<td>0.376</td>
</tr>
</tbody>
</table>

Table 6.1: Total average of WER as well as average for women and men for the standard language model and dictionary (PS), custom language model (PS LM), custom dictionary (PS DICT) and finally custom language model and dictionary (PS LM DICT).

6.1 Dictionary

An attempt to add new words to the dictionary used in PocketSphinx was done to determine if it is possible to increase the accuracy on the TED test set. The dictionary consists of a word together with the phonemes that describes that
The Sphinx Knowledge Base Tool\(^1\) was used to create the entries in the dictionary. This tool makes an estimate of the pronunciation of the words in the utterances, depending on spelling and possible subwords (e.g. ”key” in ”keyboard”). Sentences from TED transcripts that did contain words not in the standard dictionary were sent to the tool. The generated dictionary was then downloaded and merged with the original dictionary. The merged dictionary contained an additional 1917 words that were not in the original dictionary, compared to the first dictionary which contained 133000 entries. It is possible to manually change the phonetic structure of every word in a dictionary, but this was not done.

The new larger dictionary was run with the same acoustic model and language model as the original test. The WER results from this test can be seen in Table 6.1 and illustrated in Figure 6.1. RTF values can be seen in Figure 6.2. It can be seen that only replacing the dictionary actually did not make things better. WER values became worse and the RTF increased, suggesting that to only update the dictionary is a bad idea. One reason for this could be that, in some cases, the new dictionary words are considered a better match than words already in the dictionary, even though they are incorrect. The new words do not have N-gram entries in the language model, making it more difficult for the recognizer to confirm that the new words fit better compared to the old words. In other words, when the recognizer estimates that a new dictionary word is the best match, considering the phonemes of that word, there are no N-grams that can be used to confirm that it is a good match. Therefore the new word might have a perfect acoustic probability, resulting in a higher combined acoustic and language model probability than the combined probability for the old word.

### 6.2 Language Model

A custom language model was created using the available tools for PocketSphinx. The language model was created by using subtitles from TED-videos, the corpus of which consists of around 160 000 utterances. Here it is important to note that the intersection of the set of videos that is used for the test and the set of videos that are used for the language model are disjoint. That is, videos used in the test are not used to train the model. Before generating the language model, the sentences were normalized to have no punctuation, only lower case letters and other simplifications. By following the CMU Sphinx language model tutorial\(^2\), a language model in the ARPA text format\(^3\) is generated and can be used by PocketSphinx.

When using the custom made language model for PocketSphinx, WER decreased with roughly 20 percentage points, which can be seen in Table 6.1 and Figure 6.1. The WER (0.398) actually became better than the WER of Dragon

\(^1\)http://www.speech.cs.cmu.edu/tools/lmtool.html
\(^2\)http://cmusphinx.sourceforge.net/wiki/tutoriallm
\(^3\)http://www.ee.ucla.edu/~weichu/htkbook/node243_ct.html
NaturallySpeaking (0.426), which is a big improvement.

### 6.2.1 RTF

When the custom made language model was used, the RTF for PocketSphinx was increased to an average of 1.035, with a peak of 1.313, which is a lot higher than the 0.662 for Dragon NaturallySpeaking. This suggests that the recognition is really on the verge of being able to run on live video. Here it is also important to note that all tests has been run on a laptop with two cores with a total of 2GHz. With better hardware, the RTF in general will decrease. Looking at male voices, the average RTF was below 0.9 and for female voices the average RTF was 1.17, meaning female voices cannot be used for real time recognition on the hardware used for this test.

![Figure 6.1: Word Error Rates for PS (standard models) and the different combinations of the custom language model and dictionary (PS DICT, PS LM, PS LM DICT).](image)

### 6.3 Custom LM and Dictionary

The custom dictionary and the custom language model were also run together in a test, where the outcome was that the average results actually were slightly improved compared to when only using the custom language model, as can be seen in Figure 6.1. As we can see in Figure 6.2, this small improvement took a lot of extra time to accomplish. Since the average RTF was as high as 1.27,
this can not be considered to run in real time, at least on the hardware used for these tests.

For the men videos the WER actually became slightly worse (0.3755 vs 0.3762) when using both the custom language model and dictionary, compared to when only using the custom language model.

### 6.4 T-test

In [9] improvements were made by using adapted language models and the significance of the differences were calculated from a paired t-test. This improvement was highly significant, having $p < 0.01$.

Octaves `ttest` function was used to determine whether or not there is a significant difference between the results of the standard PocketSphinx models compared to using the customized model. The WER from all eight TED videos were used in this test.

Running `ttest` on the WER from the standard PocketSphinx test and the custom dictionary test, the outcome was $h = 0$ and $p = 0.1863$, from which it is not possible to determine if there is a significant difference between the results.

When running `ttest` on the WER from the standard PocketSphinx test and when using the custom language model, the test returned $h = 1$ and $p = 3.6438\times10^{-4}$. From this we can conclude that there is a significant improvement when using the custom language model compared to when using the standard language model with a confidence interval of 99.9% ($p < 0.001$).

Running Octaves `ttest` function on the WER from the standard PocketSphinx test and the WER from the custom LM together with the custom dictionary, the result was $h = 1$ and $p = 4.2451\times10^{-4}$, from which we can conclude that there is a significant difference between the results with a confidence interval of 99.9% ($p < 0.001$). When running ttest on the WER from PocketSphinx LM against the PocketSphinx custom LM and custom dictionary, the result came out as $h = 0$ and $p = 0.6122$, from which it is not possible to determine that there is significant difference.

Considering the results from the ttest, and seeing the big difference in RTF between using the custom language model and using both the custom language model together with the custom dictionary, we can not say that this combination is better than just using the custom language model.

Comparing the models with the best averaging WER with Dragon NaturallySpeaking, which performed best in the standard test, we can see that the average of the improved PocketSphinx perform better than Dragon NaturallySpeaking (see Table 4.4 and Table 6.1). Running a t-test for these two results however does not conclude that there is a significant difference between the two.
6.5 Confusion Matrix

A confusion matrix is a comparison matrix containing the original values on each row and the values to compare on each row. This leaves a matrix where the cells in the diagonal contains the count of correctly entities, and all other cells contain the number of false combinations.

Confusion Matrices are often used on a phonetic level, however they can be used on word level as well. This is however often done only when the vocabulary is small, for example if the vocabulary only consists of digits (0-9), as in [3], which talks a bit about generating a word confusion matrix for the french numbers with focus on connected digits. In [3] they found that the most errors occurred with the vowel /e:/, making it possible to improve the recognition for those words containing that vowel.

A Confusion matrix could help to see which words are most difficult to recognize. By identifying the words that are most often recognized, it is possible to adapt the acoustic model and language model to eliminate or at least reduce this false recognition. It is also possible to use this information to introduce higher-level algorithms to help the recognition. By looking at this manually (through the eyes of a human), this matrix can also help in finding words that may not be falsely recognized. For example one may find that one word that has different, but very similar, spellings, is treated as different words by the statistics, but should in fact be treated as the same word. Or one may find that a word is actually misspelled in the original subtitle, producing an error that really is not an error. This type of error should however already have been found by investigating the original subtitle.

To be able to see what types of words that were most falsely recognized in the evaluation, word confusion matrices were generated for the different combinations of custom dictionary and custom language model.

The implementation for the matrices was simplified into basically a list of list of words to save space. For each word in the original transcript, a list with recognized words together with the words frequency were generated. An example entry in the confusion matrix can be seen in Figure 6.3 for the word ”about”.

Each word that was correctly recognized is added as an entry in the matrix. When there were word sequences that had equal number of words in the original transcript as there were recognized words, each original word was assigned to the respective recognized word. When there were word sequences with different amounts of recognized words compared to the original words, no direct mapping can be done. In this case the original words were assigned with an ”unknown” recognized word.

The confusion matrix has the flaw that very similar words, where sometimes it’s just a difference of singular or plural, is treated as false recognition. This classification is of course technically correct, however in a system where the recognition of key words is more important than the exact grammatical structure, this should not be classified as a false recognition.
6.5.1 Interpretation of Confusion Matrix

Words that are more correctly recognized than falsely recognized is not very interesting to look at in terms of recognition improvement. It is more interesting to look at words that are more falsely recognized than they are correctly recognized. The most falsely recognized words for the PocketSphinx evaluation were small common words like "in", "of", "so", "you" and "are". To get a better sense of how specific key-words in the videos perform, a small set of key-words that exist in some of the TED-videos used for the evaluation were selected so that some conclusion about this can be made. Examples of key-words that exist in the original transcript are "veteran(s)", "organization", "disaster", "afghanistan", "iraq", "military", "marine", "uniform" and "neurons". In addition to these words, some words that did not exist in the standard dictionary but did exist in some of the test videos were chosen, namely "neuron", "open-source", "facebook", "iphone" and "skype". All words were basically chosen on random. Words like "facebook" can be treated as a concatenation of the words "face" and "book", and should in that case not exist in the confusion matrix at all, however this is all dependent on the goals with the recognition and how the representation of the confusion matrix is done. A summary of the the accuracy for the specific words were made using the information in the confusion matrix and can be seen in Table 6.2.

<table>
<thead>
<tr>
<th>Word</th>
<th>PS</th>
<th>PS LM</th>
<th>PS DICT</th>
<th>PS LM DICT</th>
</tr>
</thead>
<tbody>
<tr>
<td>afghanistan</td>
<td>0.0%</td>
<td>50.0%</td>
<td>0.0%</td>
<td>83.3%</td>
</tr>
<tr>
<td>disaster</td>
<td>100.0%</td>
<td>100.0%</td>
<td>85.7%</td>
<td>100.0%</td>
</tr>
<tr>
<td>iraq</td>
<td>85.7%</td>
<td>85.7%</td>
<td>28.5%</td>
<td>57.1%</td>
</tr>
<tr>
<td>marine</td>
<td>50.0%</td>
<td>50.0%</td>
<td>50.0%</td>
<td>50.0%</td>
</tr>
<tr>
<td>military</td>
<td>100.0%</td>
<td>100.0%</td>
<td>100%</td>
<td>100.0%</td>
</tr>
<tr>
<td>neurons</td>
<td>0.0%</td>
<td>100.0%</td>
<td>0.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>organization</td>
<td>100.0%</td>
<td>80.0%</td>
<td>100.0%</td>
<td>60.0%</td>
</tr>
<tr>
<td>uniform</td>
<td>33.3%</td>
<td>66.6%</td>
<td>33.3%</td>
<td>66.6%</td>
</tr>
<tr>
<td>veteran</td>
<td>50.0%</td>
<td>75.0%</td>
<td>25.0%</td>
<td>75.0%</td>
</tr>
<tr>
<td>veterans</td>
<td>62.5%</td>
<td>75.0%</td>
<td>87.5%</td>
<td>75.0%</td>
</tr>
<tr>
<td>facebook</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>62.5%</td>
</tr>
<tr>
<td>iphone</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>50.0%</td>
</tr>
<tr>
<td>neuron</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>open-source</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>skype</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

Table 6.2: Accuracy for specific key-words for PS, PS LM, PS DICT and PS LM DICT.

When using only the custom dictionary, an improvement could only be seen for the word "veterans", but none of the additional words in the dictionary were recognized, as in the case with the standard models.
When using the customized language model, it can be seen that only one word had decrease in accuracy. Only the word “organization” was misinterpreted one time of the five it was mentioned in the video, whereas the rest of the words had better accuracy or no change.

When combining the custom dictionary and the custom language model, there were an increase for the word ”afghanistan”, but also a decrease for the words ”organization” and ”iraq”. What is the most interesting is that in this case, the additional words started to become recognized, where none of the two separate methods could gain any improvements in accuracy for these words, suggesting that if specific key-words should be recognized, they must exist both in the dictionary as well as in the language model.

The lower part of Table 6.2 shows words that does not exist in the original dictionary, but is added in the custom dictionary. It can be seen that the words got pretty good accuracy as soon as they were added to the dictionary.

### 6.5.2 Usage

If using the confusion matrix ”backwards” so that a recognized word can be classified with a most likely original word, it may be possible to improve the performance of a recognizer.

For example, we can take the entry seen in Figure 6.3 and construct a simple example where the word ”review” is never correctly recognized, and we also construct the example so that the word ”about” is the only word that is falsely recognized as the word ”review”. If we have access to the confusion matrix for this specific example, we can make the fair assumption that whenever the word ”review” occurs in the output, we can substitute it with ”about”, since no other word was interpreted as ”review”. Using more sophisticated methods, this may be expanded to all words in the recognition process with the help of higher-level algorithms.
Figure 6.2: Real Time Factors for PocketSphinx using standard dictionary and models (PS) as well as custom language model (PS LM), custom dictionary (PS DICT) and the usage of both the custom language model and the custom dictionary (PS LM DICT)
Figure 6.3: Excerpt of a confusion matrix for the single word ”about”, here represented as a tree. It can be seen that ”about” was recognized as ”about” 72 times, as ”review” 2 times, as ”eleven” 1 time, as ”got” 1 time and as multiple words 40 times.
Chapter 7

Conclusions and Future Work

To answer the question of which library performs the best, the answer is without doubt Dragon NaturallySpeaking, considering the WER and RTF for the different libraries. The results of the tests performed in this report as well as the results of Keith Vertanens tests indicates that Dragon NaturallySpeaking is the best alternative. This library however requires that a Windows operating system is used.

Dragon NaturallySpeaking has the advantage that it does not require a lot of training before it can be used with acceptable performance. For PocketSphinx and Microsoft SAPI, the standard models did not perform as well as Dragon NaturallySpeaking. Even though Dragon NaturallySpeaking had best accuracy, it was not a good choice for the targeted system Vidispine. PocketSphinx was a better fit since it is written in C and has support for Linux. By customizing the language model and the dictionary it was possible to achieve results that performed better than Dragon NaturallySpeaking in terms of WER, however with a big impairment of the RTF. If corresponding adjustments were to be made for Dragon NaturallySpeaking we might however see that Dragon NaturallySpeaking outperforms PocketSphinx. Since the improvements only were made on PocketSphinx, future work could focus on comparison between using different trained models for different libraries.

Even though Dragon NaturallySpeaking is marketed as a speaker dependent speech recognition software, it works very well with the different voices in the test videos. Dragon did have the highest average precision out of the libraries. The reason for this is that Dragon NaturallySpeaking is a very competent application, being developed by a company with lots of experience.

A difference between women and men could be noticed in the evaluation, which was significant for two of the libraries. This is interesting, since the results contradicts other studies which claims that women are easier to recognize,
compared to men. One explanation for this could be uneven amount of training data for the models between men and women. One could also argue that the number of videos in the evaluation are too few, making the results skewed.

A lot of time went to understand how to setup the libraries to get them to work. More time than expected was spent to setup the libraries with their dependencies and to setup the working environments. This was especially true for the libraries Julius and Kaldi, which caused so much problems that they were excluded from the evaluation. The evaluation would have been more interesting if all intended libraries were part of the evaluation. It may be possible that Julius or Kaldi could have performed better than PocketSphinx, however since these were not part of the evaluation, it is not possible to determine if this was the case.

When the library dependencies were setup, it also took some time to realize how the libraries worked and how they could be used. In the end, both the code for the evaluation and for the integration in Vidispine did not take a significant time to implement and did not require many lines of code.

The creation of the custom language models took some time to complete, and was really worth it considering the results. It was possible to improve the accuracy for the recognition when the custom language model was used, which makes it possible to increase the recognition accuracy if the subject of the video is known.

Since the Dragon NaturallySpeaking license and the access to the API were accessed so late, it became easier to use the GUI of Dragon NaturallySpeaking than to explore how to setup the code to get recognition starting. Using the GUI instead of the library functions should not matter much, since the GUI also use the same library functions.

Since there were not enough available resources to get a hold of the WSJ test set, that test set was not used for the evaluation. Using the WSJ test set, a direct comparison could be made with the results of Keith Vertanen, to confirm or disprove his results. Instead, TED videos were used as basis of the evaluation and showed similar relationships between the libraries as the results from Keith Vertanen.

Because of the problems with the libraries Kaldi and Julius, they were pushed aside and was not used in the test. In the end there was not enough time to resolve the problems and thus no time to run the tests on these libraries. Kaldi, which has support for neural networks, would have been interesting to evaluate, since there are reports showing results that gives an advantage to the neural network over Hidden Markov Models. It does seem reasonable to think that a neural networks would improve its accuracy, the more data it is trained with, and may very well be the future successor of HMMs.

One problem with the TED test set is the fact that the audio does not only contain speech, but contain the TED-intro in each video as well as audience noise such as applauds and laughter. This affected the results of the recognitions. It could be seen that the TED-intro jingle was very often recognized as spoken words which is why a more carefully chosen test set would have been
7.1. Restrictions

7.1.1 Evaluation restrictions

If more time was available another test would most likely have been done, containing more videos, so that the libraries can be tested as much as possible. If there were enough funds available, the WSJ test set that Vertanen used would have been interesting to examine, since the results could then be compared more directly. Since the evaluation test was not a standardized test, it would be interesting to get a hold of a copy of the WSJ test set. Similar results were however achieved but with an offset of about 20\% percent units, so the results could be related to each other.

Unfortunately, not all libraries were tested, which caused the test to be smaller than expected. If more time was available, it may have been possible to get the other libraries to work as intended to get a more comprehensive test. It would have been nice to test Julius and Kaldi to see how those libraries compared as well, but since there were no English language models that could be directly used without any problems, Julius could not be tested. In the Kaldi case, it is possible that the technical problems that occurred with the operating system (Linux Mint 12) may have worked better with another operating system. It may also have been possible to resolve the issues, had more time been dedicated for this. The Julius case should have been possible to find a solution to if enough time was spent on debugging the problem.

Because some libraries were not evaluated, this made the evaluation space smaller, causing the focus to slightly move more towards the improvement of PocketSphinx used for Vidispine.

The dictionary is limited in the number of words it represents, as are the N-grams. Using (N+1)-grams can be very resource-demanding compared to using N-grams since the size of the N-gram database increases exponentially with N, which is a problem if a higher level of N-grams is wanted. It would give more accurate results, however with the cost of more processing power. It
is possible to adapt the acoustic model so that it matches the targeted talkers and audio environment better. If one knows that the videos that are to be recognized contain talker(s) that are recurring in multiple videos, then it may be worth adapting the acoustic model to match the talkers to improve accuracy. This could be done by having different acoustic models for each talker.

As estimated in the beginning of the project, there was not enough time to look into voice recognition, i.e. detection of which person is talking. This would require more time for the evaluation of appropriate libraries.

The test only contained videos from TED-talks. Another test set that may have given more accurate results could have been videos from news broadcasts, since the original models for PocketSphinx are trained on data from that kind of corpus. Using other types of videos can also show how important the specific training of the acoustic models and language models really are.

The testing of Dragon NaturallySpeaking was done through the graphical user interface, which was not the ideal way to go. The best would have been to programmatically write test cases which can be started automatically, since this is the way that a potential program will be used in the future. The difference between the two methods should however not be significant, since they both use the same libraries in the end. The biggest difference would be that it would be easier to perform the automated tests, and of course simplify for future tests and implementations. Other settings could have been used in the Dragon NaturallySpeaking tests to achieve a more broad evaluation. Other accents could for example have been tested, both in the settings of Dragon NaturallySpeaking, but also in the input videos for all libraries. However, considering Vidispine, the most common accent to be used by a potential system would be the American English.

The hardware used for testing is of course important, at least for the RTF, however when comparing different libraries the most important thing is that they are tested on equal conditions. It is more important that the libraries are tested on the same hardware than it is for them to be tested on high-performance hardware. In any case, the main goal for this study was to find the differences between the libraries, making the choice of hardware irrelevant.

Tests where specific key words is the target for the accuracy is an area that could have been investigated more thorough. This however requires a more comprehensive test. Achieving an overall better WER has however proven to improve the recognition of some specific key words, making a general improvement of the WER an alternative to a complete focus to the improvement of the key words. The confusion matrix can help to improve the accuracy for specific key words in the evaluation.

### 7.1.2 Software restrictions

The largest limitation of the current state of the implementation is the lack of proper silence detection, a feature that exists in PocketSphinx, however only when reading directly from a file. To make use of this feature in Vidispine was
something that was spent some time with, and was not solved. The speech recognition works anyway, however detection of where silence exists does not work as good as it may work. This causes the problem that utterances can not be detected properly, which forces the application to do manual utterance breaks which can occur in the middle of sentences or even in the middle of words. This causes the recognition accuracy to drop at these breaking points.

The proof-of-concept client application does not provide an easy-to-read search results list. Considering it is just a proof-of-concept application, this is not a big problem, however if it is meant to be used for demonstrations of how the backbone actually can be used, it may be non-descriptive in it self.

Since the development of the Vidispine module and client application was done in Linux, no time was spent on trying to get it to work under Windows as well. PocketSphinx can run under Windows, so it should be possible to get it to work there as well.

7.2 Future work

It will always be possible to improve the models used for the speech recognition. Even if we manage to recognize the phonemes 100% correctly, this does not guarantee that the right words are recognized. Taking examples like ”Recognize speech” vs ”wreck a nice beach”, which both contain the same phonetic structure, it will in practice be impossible to distinguish between the two. Sure, with a language model and with common sense we can determine that ”recognize speech” is more likely than ”wreck a nice beach”, but it is never a guarantee that the phonemes were meant to be interpreted that way. A human can for example not distinguish between the two if they are spoken quickly enough.

7.2.1 Evaluation

When it comes to what can be done in the future regarding the tests, the first step would be to include Kaldi and Julius in the evaluation, so that the evaluation contains a larger set of possible candidates. It is also possible to produce language models and/or acoustic models for other libraries than the ones evaluated in this report.

An attempt to improve the acoustic model can be made, to see how (if) the results can be improved by specializing the model. By specializing the acoustic model it may be possible to improve the accuracy for videos containing a special accent or to improve the accuracy for videos with specific talkers. For TED videos some work has been done in creating an acoustic model\(^1\), where a recognizer based on CMU Sphinx was used.

\(^1\)http://www.academia.edu/1587789/TED-LIUM_an_Automatic_Speech_Recognition_dedicated_corpus
As can be seen in the case of PocketSphinx, the recognition performance is dependent of the language models, as is also confirmed by looking at articles such as [25], where the use of different corpora produces significantly different WER. To train a language model to fit the task assumes that we know something about the videos that are to be recognized. For PocketSphinx in this report, the language model was trained on a set of texts that were representative for the domain (TED-talks) of the test cases, even though the talks can be very varying in contexts, it proved to be a better model for the test set than the original based on news broadcasts. If the language model for example would have been trained on texts targeted at children, the outcome from the TED test set would probably have become worse. It should be possible to reduce the language model and dictionary to improve RTF with little impact of the WER by removing odd word combinations or irrelevant words for the specific area of recognition.

Allowing for larger, or additional, tests containing additional types of videos, such as news reports or recorded meetings could allow for a larger understanding of how the recognizers perform in different environments. Testing the recognizers on different hardware should be a natural next step, making sure that it is possible to make use of the custom language models on live video, since the results of this evaluation showed that the custom models required additional time and eventually exceeded an RTF of 1. The hardware is the limitation that sets the boundaries of the number of dictionary entries and N-gram entries in the language model with the current algorithms used.

### 7.2.2 Software

The software does not store the generated meta-data in the actual video file, but only on a server. This is something that may be fixed in the future, so that the text always exists in the same location as the video.

The meta-data that is stored separately on the server only contains an URI to the video file. This should be more standardized by storing an identification ID of the video instead, so that the ID can be used to access the video file. This way, the path does not need to be stored in multiple locations.

To increase the accuracy even more, it is possible to look into automatic speech understanding to make sure that sentences are more correctly structured and makes more sense. It is possible to post-analyze the generated text or make this an integral part together with the current recognition methods. One project related to this is the "Open Mind Common Sense"[^2] which collects "common sense"-facts, of the type "norway is a country in europe" and "humans are likely to be found in norway". Using these "common sense"-facts it is possible to find likely patterns in the recognized speech to increase the accuracy. In [15], it is suggested that this can improve the accuracy of speech recognition compared with N-grams, since it may be a closer representation of

how humans process speech. We humans make assumptions about the previous context in a conversation, making certain words more likely than others.

Another feature to look into is voice recognition. If it is possible to recognize who is talking, and if this talker occurs often in videos, it may be possible to have a specially trained acoustic model (and language model) that activates as soon as this talker is recognized. Having separate models for each talker might help in increasing the accuracy rates.
Chapter 8

Acknowledgements

I would like to thank my supervisors Frank Drewes and Petter Ericson for giving me input and valuable thoughts during the work.

I want to thank Codemill for providing the initial idea for the thesis, and I also thank everyone at Codemill for helping me to get started using their systems and for the helping when I was in need.

I want to thank Michael Minock and Mats Johansson for helping with the Dragon NaturallySpeaking licensing, even though the communications with Nuance did not went as smoothly as one could have expected.

I want to thank the other thesis students at Codemill, especially Fredrik and David, that have brightened the atmosphere in the room we shared.
References


Appendix A

Evaluation videos

1. Ted talk ID: 1584¹
   Talker: Ruby Wax
   Title: What’s so funny about mental illness?
   Link: http://www.ted.com/talks/ruby_wax_what_s_so_funny_about_mental_illness.html
   Gender: Female
   Length: 08:44

2. Ted talk ID: 1592²
   Talker: Melissa Marshall
   Title: Talk nerdy to me
   Link: http://www.ted.com/talks/melissa_marshall_talk_nerdy_to_me.html
   Gender: Female
   Length: 04:34

3. Ted talk ID: 1594³
   Talker: Heather Brooke
   Title: My battle to expose government corruption
   Link: http://www.ted.com/talks/heather_brooke_my_battle_to_expose_government_corruption.html
   Gender: Female
   Length: 18:57

¹http://www.ted.com/talks/ruby_wax_what_s_so_funny_about_mental_illness.html
²http://www.ted.com/talks/melissa_marshall_talk_nerdy_to_me.html
³http://www.ted.com/talks/heather_brooke_my_battle_to_expose_government_corruption.html
4. Ted talk ID: 1603
   Talker: Hannah Brencher
   Title: Love letters to strangers
   Link: http://www.ted.com/talks/hannah_brencher_love_letters_to_strangers.html
   Gender: Female
   Length: 04:52

5. Ted talk ID: 1604
   Talker: Gary Greenberg
   Title: The beautiful nano details of our world
   Link: http://www.ted.com/talks/gary_greenberg_the_beautiful_nano_details_of_our_world.html
   Gender: Male
   Length: 12:06

6. Ted talk ID: 1607
   Talker: Matt Killingsworth
   Title: Want to be happier? Stay in the moment
   Link: http://www.ted.com/talks/matt_killingsworth_want_to_be_happier_stay_in_the_moment.html
   Gender: Male
   Length: 10:16

7. Ted talk ID: 1608
   Talker: Jake Wood
   Title: A new mission for veterans – disaster relief
   Link: http://www.ted.com/talks/jake_wood_a_new_mission_for_veterans_disaster_relief.html
   Gender: Male
   Length: 04:59

---

4http://www.ted.com/talks/hannah_brencher_love_letters_to_strangers.html
5http://www.ted.com/talks/gary_greenberg_the_beautiful_nano_details_of_our_world.html
6http://www.ted.com/talks/matt_killingsworth_want_to_be_happier_stay_in_the_moment.html
7http://www.ted.com/talks/jake_wood_a_new_mission_for_veterans_disaster_relief.html
8. Ted talk ID: 1612\(^8\)

Talker: Jeff Hancock
Title: The future of lying
Gender: Male
Length: 18:31

\(^8\)http://www.ted.com/talks/jeff_hancock_3_types_of_digital_lies.html