Topics in Speech Recognition

by

D. A. Liauw Kie Fa

Delft University of Technology
Mediamatics
Man-Machine Interaction Group

DUTCHEAR B.V.

Thesis Committee

Dr. Drs. L.J.M. Rothkrantz	Delft University of Technology
Ir. F. Ververs	Delft University of Technology
Dr. Ir. C.A.P.G. van der Mast	Delft University of Technology
Ir. P. Wiggers	Delft University of Technology
Drs. H. Jongebloed	DUTCHEAR B.V.

D.A. Liauw Kie Fa
Master's Thesis, June 1, 2006
"Topics in Speech Recognition"

Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Department of Mediamatics
Mekelweg 4, 2628 CD Delft, The Netherlands

DUTCHEAR B.V.
Brassersplein 2, 2612 CT Delft, The Netherlands

Preface

In December 2004, I started my graduation project at DUTCHEAR, which I had found with the help of my thesis supervisor, Leon Rothkrantz. I thought I wanted to do something with designing intelligent dialogs and had started my literature research on the topic. However when I started the internship I found that making an intelligent dialog design practical was frankly being hindered by the speech recognizer's performance. Thus the focus gradually shifted to topics which were more in the direction of improving performance, namely English foreign word recognition and Call Classification. In short a time I managed to restart my literature research and finish the goals set by myself and my supervisors, while in the same adjusting to a new environment and being around interesting people from whom I was able to learn a lot.

For letting me do my project amongst them and for giving me this great learning experience, I'd like to thank the complete DUTCHEAR team, Victor Huisman, Bram Vromans, Sander de Graaf, Hanna Schösler, Els Nachtegaal, Corien Brinkmeyer, my fellow interns with whom I shared a room and had lots of fun, Marko Simsic and Thomas de Bondt, and last but not least my supervisor at DUTCHEAR, Hans Jongebloed, for giving me the freedom to choose my own path and giving me lots of helpful advice in the process. My gratitude again goes to Leon Rothkrantz, for putting up with all the changes and giving me advice in writing this thesis whenever I hit a roadblock. Finally I'd like to thank my friends and family for giving me the opportunity to study and their support through all the years.

Abstract

Automated Speech Recognition has many open problems. In this thesis two well-known problems are researched. The first topic deals with the ever growing phenomenon of English words being used in Dutch colloquial speech. This makes it increasingly expectable that speech recognizers in the Netherlands should be able to recognize these English words as well as Dutch. Dutch people speaking English introduces a number of problems, including non-native speech and lack of training data. Thus a multilingual acoustic modeling approach was attempted using training data from widely available Dutch and English corpora. The second topic deals with the realization of a call classifier which is trained using machine learning techniques. Machine learning techniques algorithms in call classification literature such as BoosTexter generally require a large amount of data before satisfying results are obtained, thus making them unsuitable for use in the beginning stages of a project where human expertise is more suitable. By using the RIPPER rule generating algorithm hopefully a system can be made which accepts human knowledge in the early phases of a project, as well as machine generated rules which can be inspected by the expert in later stages of a project as more data becomes available.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In Arthur C. Clarke's 1968 popular science fiction movie "2001: A Space Odyssey" the notion of humans and computers communicating back and forth perfectly through speech was introduced to the general public. HAL, the space ship's computer, is able to chat interactively and intelligently with crewman Dave Bowman and has no trouble understanding him. Real life systems today are still a long way from this level of interactivity and recognition performance.

The first milestone in speech research was reached in 1936 when Bell Labs researchers were able to demonstrate a machine capable of synthesizing speech. This voice was rather robotic, and this type of voice has been popular in many science fiction movies. In the 1940s the U.S. Department of Defense began funding work in this field. Researchers quickly realized that unrestricted, large vocabulary, speaker independent systems were unfeasible for their time, so they concentrated on speaker dependent, small vocabulary, discrete speech systems. It took until the early 1970s when Lenny Baum invented the Hidden Markov Model approach to speech recognition. Shortly thereafter the first commercial speech recognizers appeared. The Hidden Markov Model approach was and still is the most successful approach to date.

With the realization of Moore's Law and the availability of more and more computational power research has shifted over the years from speaker dependent, small vocabulary and discrete to large vocabulary, speaker independent and unrestricted and is still ongoing.

In December 2004 I started my internship at TNO-ICT and DUTCHEAR, a small innovative company which is a leader on the telephone speech recognition market in the Netherlands, where I was able to gather information to write this thesis, as well as perform research to deepen their knowledge in this field. A lot of insight and experience was gained during the internship which lasted until June 2005.

The thesis is built up as follows. In this chapter the current state and problem areas of speech recognition are discussed, which in turn lead to two specific problems or topics which will be worked out in the rest of this work, namely multilingual recognition of English spoken by Dutch, and machine learning for call classification. In Chapter 2 background theory is discussed into the workings of a speech recognizer, multilingual recognition and machine learning. In Chapter 3 the corpora, or huge collections of speech recordings used in this thesis will be covered, and in Chapter 4 the used tools. The two main topics are covered in Chapter 5 and 6 respectively, and finally the conclusion in Chapter 7.

1.1 Problem areas in speech recognition

Automatic Speech Recognition has been around for a long time. Most speech recognizers today are based on HMM (Hidden Markov Model) techniques, and this technique is generic enough to be used on many languages. The first speech recognizers were developed in the United States, but since then speech recognizers have been

made for many other languages: Dutch, Japanese, German, French, Italian etc. Speech recognizers are traditionally trained on a large general corpus for that language, consisting of a lot of utterances from native speakers, usually phonetically rich sentences, so that most of the transitions between sounds are covered, and that a lot of different variations of the same sound made by many people is learned by the recognizer. Utterances are usually transcribed orthographically and sometimes phonetically. An alignment is then needed to determine in what part of the recording a certain phoneme is present. Alignments can be done manually or automatically. The final alignments are used for training the HMM models of the recognizer.

In the Netherlands some examples of successful applications of speech recognition over the years are:

- OVR Time travel:
  Callers can ask the system for the schedule of trains leaving from station to station on a certain day.
- Stocks and shares information line:
  Callers are kept informed about the shares and stocks on the exchange market.
- Postal code line:
  Callers say the place, street name and house number to get the corresponding postal code.

Despite speech recognition's success, there are still areas of research left within speech recognition, for example:

- Large vocabulary recognition and free speech:
  Many applications up until now are limited to short utterances by the users, or system initiated dialogs where the system takes control and asks the questions. This makes the expected answers predictable, short and there is less probability that the user has to correct himself in an utterance. In order to give users more freedom, the application should accept longer sentences and free speech so that he can take more control over the dialog, but this also makes his sentences unpredictable, especially with hesitations, corrections and sentence restarts. Task specific language models have to be made in order to optimally predict possible utterances. This is also important for dictation applications. After the sentence is recognized, there is the problem of interpreting it and taking the next course of action, but speech understanding is another subject which will not be discussed here.
- Prosody and Tonal languages:
  Words can be said with different tones. Saying something in a different tone can for example mean the exact opposite, for example in the case of sarcasm. Even though the sound is the same, the tone is different. Speech recognizers are good at recognizing sounds, but still have a hard time distinguishing the tone. Therefore there is much research being done in order to detect the tone of an utterance and the emotion behind it, being anger, happiness, sadness etc.

  Tones are also important for certain languages, particularly Asian ones, such as Chinese and Japanese. For example, in Japanese, a word with two syllables can have different meanings, depending on whether the emphasis is on the first or second syllable, or no emphasis.
- Noisy environments:
  Speech recognizers are generally trained purely on the speech and most of the time in limited environments. Speech over telephone line, or in a quiet room through a microphone are the most common. With the rise of mobile phones, there is the need to use speech recognition in much more noisier environments than the recognizer is trained on, for example in the car in traffic or in the middle of a shopping center. A lot of research is being done to improve recognition in these environments, but as it is hard for humans to understand people with a lot of background noise, it is even harder at the moment for computers.
- Speaker recognition and verification:
  With the recent terrorists attacks, there is a lot of focus on security, and to verify that people are really who they say they are. One major focus is on biometrics. The idea is that based on a speech sample from a person the system should be able to identify him (speaker recognition), or verify that he really is who he claims to be (speaker verification). Obviously speaker identification is the harder problem.

- Multilinguality and Non-nativeness:
  With globalization and immigration, a lot of people travel over the world, and a lot of people communicate with each other using a common language, which is not necessarily their own. In some countries such as Belgium, Switzerland and Canada there is more than one official language. Many people also use a lot of loan words from other languages, particularly English. Companies want to make speech recognition services available for as many people as possible, so speech recognizers should be able to detect or recognize multiple languages, and recognize the speech of people not speaking in their native language.

In telephone speech there are additional problems and doing recognition over a telephone line is even harder. Comparisons are drawn with dictation systems, because these are the most successful applications, where vendors claim accuracy of 95% or higher, and as such some people (wrongly) come to expect this kind of performance for automated speech solutions over the telephone.

- Speaker independency:
  Telephone services are open to anyone with a phone and who is willing to dial the number. Therefore a whole variety of speakers can be expected to make use of these services. In contrast to dictation systems where the speaker is known beforehand and performance can be optimized for one speaker, the recognition must be as optimal for an as large group as possible. Because of this factors which affect the voice such as race, dialect, non-nativeness, gender, age, health etc. are all variables for which an as wide range as possible should be covered by the recognition system. This makes the problem much harder and the relative performance is therefore lower than for a system for which the speaker is known beforehand. The amount of necessary training data needed is also larger, because a lot of speaker variability should be covered by the training set.

- Noise and low-bandwidthness:
  Humans can perceive sounds of up to around 20 kHz. As this is the limit for humans it makes sense that for speech recognition also up to this frequency should be captured in the speech recordings. This can be done by recording at 44 kHz with a good microphone. However telephony is narrowband and only sounds of up to around 4 kHz can be transmitted over an analog telephone line (and recorded at 8 kHz), meaning that there is a lot of information loss in the higher frequencies. Because of this loss of information the performance of telephony speech recognition systems is less than speech recognition systems which do not have to cope with this limitation and can record under ideal recording circumstances. Analog, humans are also affected since they often need to put more effort into understanding people over a telephone line. Also the quality of the telephone microphones can vary. Training data should therefore consist of recordings made over the telephone line, to prevent the recognizer from relying on frequency information above 4 kHz.

  The second issue is noise. A dictation application is often used in a relatively quiet office and broadcast audio is recorded in a controlled studio environment. The environment that people use their telephones is much more variable. In fixed telephony the noise can be from screaming children, passing cars, sounds from the television or radio etc. With the rise in use of mobile phones extra noise sources are introduced, since phones are used in even more environments. For example, in cars the acoustics are completely different than from the living room environment and the phone is often used in combination with a hands-free set. The hands-free microphone can record sounds from farther away than a traditional telephone microphone and more noises are potentially captured. As noise hinders the human ability to recognize speech, it also hinders the performance of speech recognizers. Speech recognizers should therefore be additionally trained on or adapted to noisy data to improve robustness.

- Language model training:
  It has been shown that language models improve recognition performance of sentences. Dictation applications can use language models from large amounts of text which are widely available, usually newspapers, because the speaking style used is mostly the same. Also in dictation and broadcast transcription systems speakers formulate a complete sentence beforehand or read from a cue sheet, making little or no mistakes.

  For telephony systems language models are much harder to obtain. Telephony systems are often interactive and require that users formulate their answers or questions on the fly. This causes them to make mistakes,

hesitations, make restarts etc. These can occur at any time within the utterance. Furthermore, the automated understanding of such utterances becomes much harder, since the system has to be able to detect and disregard any mistakes the user made in the sentence.

For good language models in telephony, transcriptions of real dialogs should be used. Speech data must therefore be collected and transcribed for each task. This data is often not reusable for a new task unless there is a certain amount of similarity in the domain and type of language used.

- Amount of training data for Dutch:
  As described earlier, the most useful solution to issues in telephone speech (and speech recognition in general) is simply more data. All would be well if this were available. For Dutch there are only a few general purpose use corpora which are recorded over telephone lines. These are Polyphone and Speechdat II. For acoustic models, this amount of data can be regarded as sufficient, but certainly not abundant. Polyphone can be used to train generic acoustic models. Speechdat II can be used for the mobile environment. Large corpora for Dutch such as the Spoken Dutch Corpus do exist, but they are more suited for dictation applications and have limited usefulness for telephony applications because of reasons mentioned earlier.

  Task specific free spontaneous speech corpora also exist, for example OVR Train table Information, but are harder to obtain, and don't have much use beyond the public transport domain. For domains such as stocks, banking, support helpdesks etc. there are no known corpora which can be obtained (freely or commercially). Thus new data must be collected from scratch when creating a telephony application for a new domain which involves free spontaneous speech.

1.2 Specific problem definitions

In the foregoing section several challenges in speech recognition in general and Dutch telephony speech recognition were discussed. Two topics were selected in accordance with the product innovation team at DUTCHEAR for which experiments were carried out in this thesis, as they were considered interesting with the eye on improving products and services or developing new ones in the future.

1.2.1 English foreign name recognition

English words are a common phenomenon in Dutch colloquial speech. Examples of such words are: first and last names, company names, titles of books, movies, CDs, product names etc.

For certain spoken telephone dialog systems it is desirable that these English names are properly recognized. Many telephony services which require adequate recognition of such words, for example: a movie ticket ordering service, a name dialing service where a large number of people are from foreign origin.

Also the handling of requests in multiple languages is of some interest, for example an international help desk where there are callers from many countries and they want to be helped in their own language. This however is not investigated here.

Recognition of English words by a Dutch speech recognizer is not straightforward because English has phonemes which are untypical for Dutch. In addition there is the difficulty that many Dutch people do not have a perfect English pronunciation and have tendencies to borrow the closest Dutch phoneme.

Problem definition and goals

The problem is the following: if a Dutch speech recognizer is used to recognize English foreign words, is the performance sufficient and is there an improvement if training data from other languages, in particular English, is used? What approach can be used to incorporate training data from other languages? Also, would such an

approach cause a degradation in its performance with regard to regular Dutch words? Of course if new models trained additionally with other languages are used they must also be speaker independent, noise robust etc. to be suitable for telephony speech.

The goal for this topic is to find out whether multilingual recognition techniques can be used to improve recognition of non-native English spoken by Dutch people. The focus is specifically on name dialing applications which in the future are hopefully able to handle foreign names and non-native Dutch speakers. The main performance meter of the evaluation is the speech recognizer's performance on single word recognition. The goal is not to build such a system completely, but to find out whether multilingual techniques are a feasible option for recognition, which is the case if there is an improvement in the recognition of English words spoken by Dutch people using multilingual models at little or no cost for the recognition of normal Dutch utterances.

Approach

Because there is barely any training data of Dutch people speaking English to make a speech recognizer in the traditional manner, a multilingual acoustic modeling approach will be tested. Training data will come from various languages and sources so an approach is needed to minimize differences between training corpora. The goal is to recognize English words and phonemes, but reduction in the performance of recognition of Dutch must also remain minimal. A lot of testing will be done to ensure that.

In order to do this, several stages were planned and carried out in this order:

1. A survey was done in literature in order to find the best methods to train and use multilingual acoustic models optimally in this situation.
2. Corpora were selected and prepared for training acoustic models for the speech recognizer using these methods. Not all corpora are equally suited, because of reasons mentioned in the previous section.
3. Several multilingual acoustic models are trained using techniques found to be suitable in the first phase, as well as a baseline Dutch speech recognizer which is needed for measuring the relative performance of the multilingual models. Throughout this part of the work the SONIC speech recognizer is used.
4. Various tests are done which are representative of the kind of utterances that a name dialing application will face. Testing will show whether these techniques are promising or not.

To know if the multilingual models are suitable for the desired use, the following subquestions must be answered with testing:

1. How is the performance of multilingual models on Dutch?
   Tests with a Dutch corpus are done to check whether the performance for Dutch does not degrade, or at least not too much compared to a Dutch only recognizer.
2. How is the performance of multilingual models on native English?
   This is useful to see if recognition of English phonemes is actually working, so a test with an English corpus is done.
3. How is the performance of multilingual models on English utterances spoken by Dutch?
   This is the most interesting. Does performance in fact improve compared to a Dutch only recognizer, and if so how much, under what circumstances and why? To answer this question tests with utterances from Dutch people speaking English are done.

1.2.2 Call classification

The idea of a call classifier is the following:
Traditional IVR (interactive voice response) systems which require callers to press on telephone buttons (DTMF

tones) to navigate through menus to connect to an appropriate agent are commonplace. For small menus these systems perform satisfactory, but when the menus are large, they can become difficult and time-consuming to navigate, often frustrating users. People forget the menu options and have to listen to the options again, and most importantly, the menu does not always correspond to the user's mental model. When users might expect an option in the menu, such option might not exist or might have a different or obscure name. This causes people to select the wrong option only to be turned away or redirected by the agent they are wrongly connected to.

In more recent years machine learning techniques have become popular. Machine learning classifiers learn patterns from the training data. When enough data and a good algorithm is used, the machine learning classifier is capable of successfully classifying unseen data.

The main problem for implementing a call routing system using machine learning methods successfully in the Netherlands is data, or to be more specific, the lack of it. Data is necessary to train good language models and to train the machine learning classifiers. Speech data must also closely reflect the situation where such a system is to be used.

Problem definition and goals

The problem definition for this topic is to find out how to make a good call classifier using machine learning techniques which also performs well in the initial phase of the system's life. Additionally the machine learner should be easy to understand and modifiable by a human expert so he can supervise the system and compensate for shortcomings in the classification early in the system's life. Obviously performance will increase as more data becomes available and the human expert, but it is the initial performance which is of most interest.

The classification system should be able to:

1. Recognize long free speech utterances:
   For this part of the work the SpeechPearl 2000 speech recognizer was used, which is able to recognize sentences.

2. Work using Machine Learning Techniques:
   Since a lot of machine learning classifiers in literature can be somewhat of a "black box" it is desired that the resulting classifiers should be easily understandable and modifiable for a human.

3. Perform comparably to known machine learners:
   Of course any machine learning classifier used must not perform worse than known techniques in the field, and preferably better.

4. Deal with uncertainty:
   Most speech recognizers generate the result with a certain confidence for each recognized word. This is different from the text classification domain where words usually don't have a confidence. When the confidence is low there are probably some misrecognized parts which should be ignored or at least given lesser weight in the decision. It is desirable that the classifiers generated by the machine learners make use of this extra information.

It's not the idea to build a complete system. Contrary to the first topic in this thesis, the focus is not on the speech recognizer or improving recognition performance, but on the machine learner and the classifiers they generate which make decisions based on the output of the speech recognizer.

Approach

Several stages were necessary to find a suitable classifier:

1. A survey was done in literature in order to existing methods in the speech and sometimes text classification domains. The domains are related because the output of speech recognition is a string of text, but not the same, because text classification usually deals with large texts.

2. Machine learners with attractive properties were selected. This would be a known in literature called BoosTexter and another one not previously used for call classification called RIPPER, but with the attractive property of being easily understandable by a human.

3. A corpus of actual calls was collected using a dummy system. The idea was to model the beginning stages of a call classification system, so the amount of calls recorded remained to a minimum, which was around 1500.

4. Experiments were carried with the corpus with the selected machine learning algorithms. These experiments would compare the algorithms, but also different methods of training them.

1.3 Dutchear BV

DUTCHEAR is a new name in the Dutch speech technology market. Over the years it has undergone numerous changes and metamorphoses. Starting as an academic research group in speech technology, DUTCHEAR is now a commercial company.

In the 1970's and 1980's the PTT research and development laboratory, the Dr. Neher Laboratory, started the exploration of speech processing by a computer. In the future computers would be able to listen and talk to humans.

In the 1990's PTT Research was one of the first in the Netherlands to successfully build a software based speech recognizer that could understand the Dutch language over telephone lines and was awarded a patent for the search algorithm. PTT Research also collected large amounts of Dutch spoken texts recorded over telephone lines and these are still used today as training material by all major speech recognition technology suppliers (Scansoft, Nuance etc.).

PTT research changed its name to KPN Research. KPN Research collaborated with major speech technology players of the day, such as Philips Research, Philips Dialogue Systems, Lernhout & Hauspie (latter two are now part of Scansoft) and the University of Nijmegen (now Radboud University Nijmegen). KPN Research was also involved in numerous European research projects which covered speech recognition, speaker verification, multimodal systems (recognition of speech in combination with keyboard or pen input).

The most important projects were the automization of 008 (later 118) telephone number enquiry, train travellers information (OVR 9292), secure services with speaker verification, such as ScopeCard^{TM}and the use of speech recognition in the mobile network, VoiceDialling^{TM}.

In 2000 KPN Research started doing research for parties outside KPN, such as Fortis and ABN AMRO, and as a result more and more speech technology related products and services were developed. On January 1, 2002 the KPN R&D laboratory was transferred to TNO and was renamed to TNO Information and Communication Technology.

TNO Information and Communication Technology created the section DUTCHEAR with the mission to independently explore and exploit the commercial market for speech technology and related services. On January 1, 2005 DUTCHEAR became an independent company with TNO Information and Communication Technology as its majority shareholder.

DUTCHEAR is a speech technology company whose people have expertise in many areas:

- Speech technology and linguistics: to know how speech and speaker recognition works.

- Software and electrical engineering: to know how to build software and integrate them in existing software and telephony systems.
- Psychology and cognitive ergonomics: to know how to make user friendly speech interfaces.

DUTCHEAR is continually developing new products and services to meet market demands and expand its portfolio:

- Customer Call Gateway: Automatically route incoming calls to the correct agent using speech recognition.
- Collega Connect: Call any coworker using your voice and save time and effort by never having to look up phone and extension numbers again.
- Personal Connect: A product which ensures that you never again have to remember a telephone number. Call friends, family and coworkers with your voice. The list of contacts is personalized and can be updated via the web, telephone or Microsoft Outlook.
- Speaker Verification: A product that can ensure that a person is who he claims to be over a telephone line. It works by comparing the person's voice with a known profile in the database.
- Call Scan: A new product that logs and scans incoming call center calls. Call center managers can then search the database using criteria they specify to gain insight in caller behavior.

Chapter 2

Theory

2.1 Introduction

The purpose of speech dialog systems is the exchange of information in a more interactive manner using speech to communicate. The user wants to accomplish something with the system, but in order for the system to do so, it must acquire bits of information from the user. For example, it needs to know the departing and arrival train stations for a trip, and desired arrival time. These bits of information are sometimes referred to as slots. When the information is known by the system the slot is filled, and when it is still unknown it is empty.

Speech dialog systems can be user-initiated or system-initiated, meaning the user and system respectively have control over the dialog and determine in which direction the dialog will go. System-initiated dialogs have the advantage of being very predictable, meaning that the dialog designers can limit the number of expected utterances by users by carefully formulating a direct question. Users new to the system will generally have little confusion as to what to do, because the system explicitly instructs them when and what to say. Information slots are usually filled one by one.

On the other hand user-initiated dialogs are less predictable less structured and more dynamic. They give the user more freedom. The dialog usually starts with an open question to invite the user to use his own words as much as possible. Instead of asking for information one slot at a time, the user might give the information to fill more slots or even all slots in one utterance. It is up to the system to determine which slots can be filled, and based on the slots which are still empty it will continue the dialog. User initiated systems allow for more natural dialogs and are usually better for more experienced users who already know what the system needs to know. The downside is that the user utterances are far less predictable, thus making successful recognition and understanding of the utterance many times harder.

A typical speech dialog system consists of several components. Figure 2.1 shows a general schematic of the information flow in a dialog system. A dialog is usually opened by the system, for example a greeting followed by a request by the system. This can be an open question such as “How may I help you?”, a direct question or command like “Name the train station from where you wish to begin your trip”. The user responds by speaking into the microphone (telephone). The speech recording is processed by the speech recognizer. The speech recognizer consists of an acoustic model, a lexicon and a language model or grammar. The acoustic model has information about which sounds belong to which phonemes, the lexicon information about how sequences of phonemes form words, and the grammar or language model information about the sequences of words which are allowed. With the information in these 3 components, the speech recognizer will return a hypothesis of the most likely utterance by the user.

The dialog manager has two tasks. First, based on the speech recognizer output it must be “understood”. Second, it must determine how the system will respond. Understanding a user utterance can be very easy if questions are

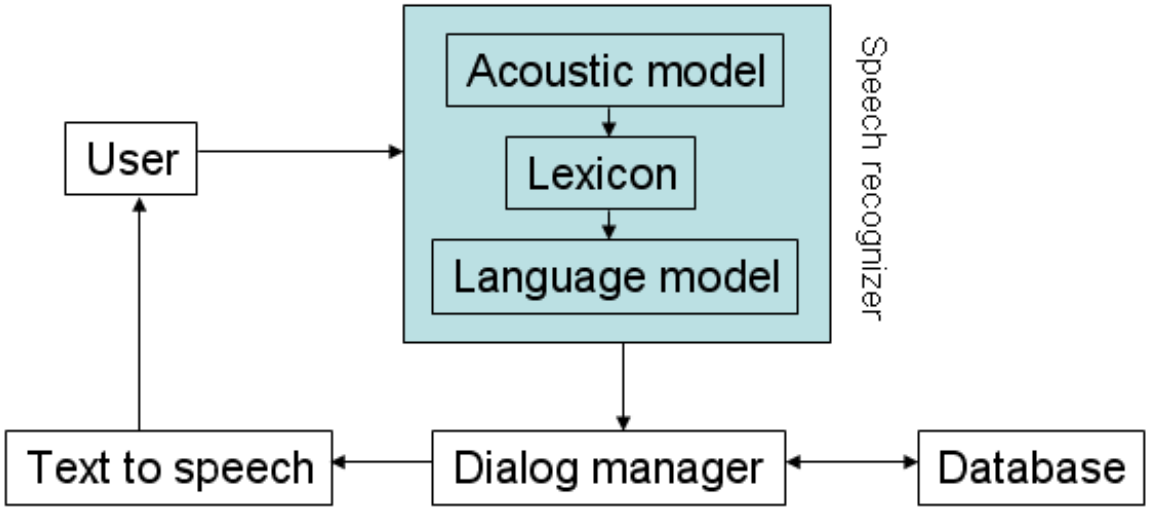


Figure 2.1: A schematic of an automated speech dialog system

asked very direct. For example in the case of “Name the train station from where you wish to begin your trip?”, the expected answers are very short, like “Delft” or “Amsterdam”. In this case responses can be simply understood by matching the response with a list of train stations. If the question was very open like “How many I help you?” the response could be “I want to go from Delft to Rotterdam and I want to arrive there at 2 o’clock”, or “I need to be in Rotterdam at 2 o’clock and I am leaving from Delft” and many other variations. Traditionally grammars are used to to cover most of the sentence patterns used, for example “from <departure-station> to <arrival-station>”, “arrive at <arrival time>”, “leaving from <departure-station>”. Grammars can work for a great deal, but they generally do not cover all possible sentences and there are other phenomena which need to be solved, such as references.

The system’s response is returned spoken by using a text to speech (TTS) system, or sometimes using prerecorded utterances. There can be actions associated with the response, such as routing a call or placing an order. The response by the dialog manager is usually determined by the information which is still unknown. If nothing is recognized, the dialog manager might choose to repeat the question. If something is recognized the results are parsed and interpreted by the system. If the dialog manager thinks it can fill a slot, it might ask the user for confirmation. If all slots are filled and confirmed, the dialog manager has all the information it needs and can perform a query on the database and return the result to the user. If there are still missing slots the dialog manager will proceed to ask questions to fill them.

Dialog design and formulating the right questions is an art and there are many theories and strategies to get for example the best user experience, the best recognition results, or the best dialog completion rates. Strategies can be from highly structured, to very dynamic, switching from user to system-initiated on the fly, but this falls out of the scope of this thesis.

In this chapter some background theory is given in the area of speech recognition. First a general introduction to the nature of speech and how speech recognition works. Then some literature will be discussed about each of the two main topics which will be described later in Chapters 5 and 6. The first topic of English words spoken by Dutch is a problem which is for the most part located in the acoustic model, because additional sounds which are not typical for Dutch must be recognized. The second problem of call classification is a problem located in the dialog manager, because, on the basis of what the user says, the speech recognition result must be interpreted and a decision must be made where to send the caller to.

2.2 Speech recognition basics

In this section several aspects of the workings of speech recognition will be discussed. First some information about what speech actually is and how it is stored and processed on the computer. Then some theory about the technology making it all possible, Hidden Markov Models, and the procedures which need to be done in order to train a speech recognizer.

2.2.1 What is speech?

Speech recognition systems focus on sounds that distinguish one word from another within a language. Those sounds are called phonemes. The words “seat”, “meat”, “beat”, “cheat” are different words, because in each case the initial sound (“s”, “m”, “b”, “ch”) is recognized as a separate phoneme in English.

For speech recognition to work, there must be a representation for it. There are several approaches to represent speech [Markowitz96]:

- Articulation: The analysis of how speech sounds are produced.
- Acoustics: The analysis of the speech signal as a stream of sounds.
- Perception: The analysis of how speech is perceived by a human listener.

Articulation

The focus of articulation is on the vocal apparatus consisting of the throat, mouth and nose where the sounds of speech are produced. The throat contains the vocal cords whose vibration produces voiced phonemes, like the sound of “ee” in the word “speech”. The mouth and nose are called resonating cavities because they reinforce certain sound wave frequencies. The resonating cavity of the nose is used for speech when the soft palate is lowered and air is allowed to flow into the nasal cavity. This is the way in which nasal phonemes such as /m/ and /n/ are produced. [1]

Phonemes are categorized by the way they are produced. Two main phoneme categories exist: vowels and consonants. Consonants are characterized by a total or partial blockage of the vocal tract. The effect of the obstruction is to produce noise. Noise can be an explosion of stop consonants like /p/ in “pat” or the hiss of of fricatives such as the /s/ in “see”. The noise is present even when the consonants are voiced such as the /b/ in “bat” or the /z/ “zee”.

As Figure 2.2 illustrates, consonant classification uses the

- Place of articulation (top edge of the consonant chart)
- Manner of articulation (left edge of the consonant chart)
- Presence or absence of voicing (within a cell unvoiced is the one on the left, voiced is the one on the right)

Vowels are characterized by strong harmonic patterns and relatively free passage of air through the vocal tract. Semi-vowels such as the /y/ in “you” fall between consonants and vowels. They are often classified as a type of consonant because they contain partial blockage of the vocal tract, but they also have strong harmonic patterns like vowels.

[1] Throughout this work, X-SAMPA notation is used for phonemes instead of IPA notation, mainly because of the ease of use, and more importantly because X-SAMPA like notations was used in the speech recognition software. A mapping table can be found later on in Table B.1.

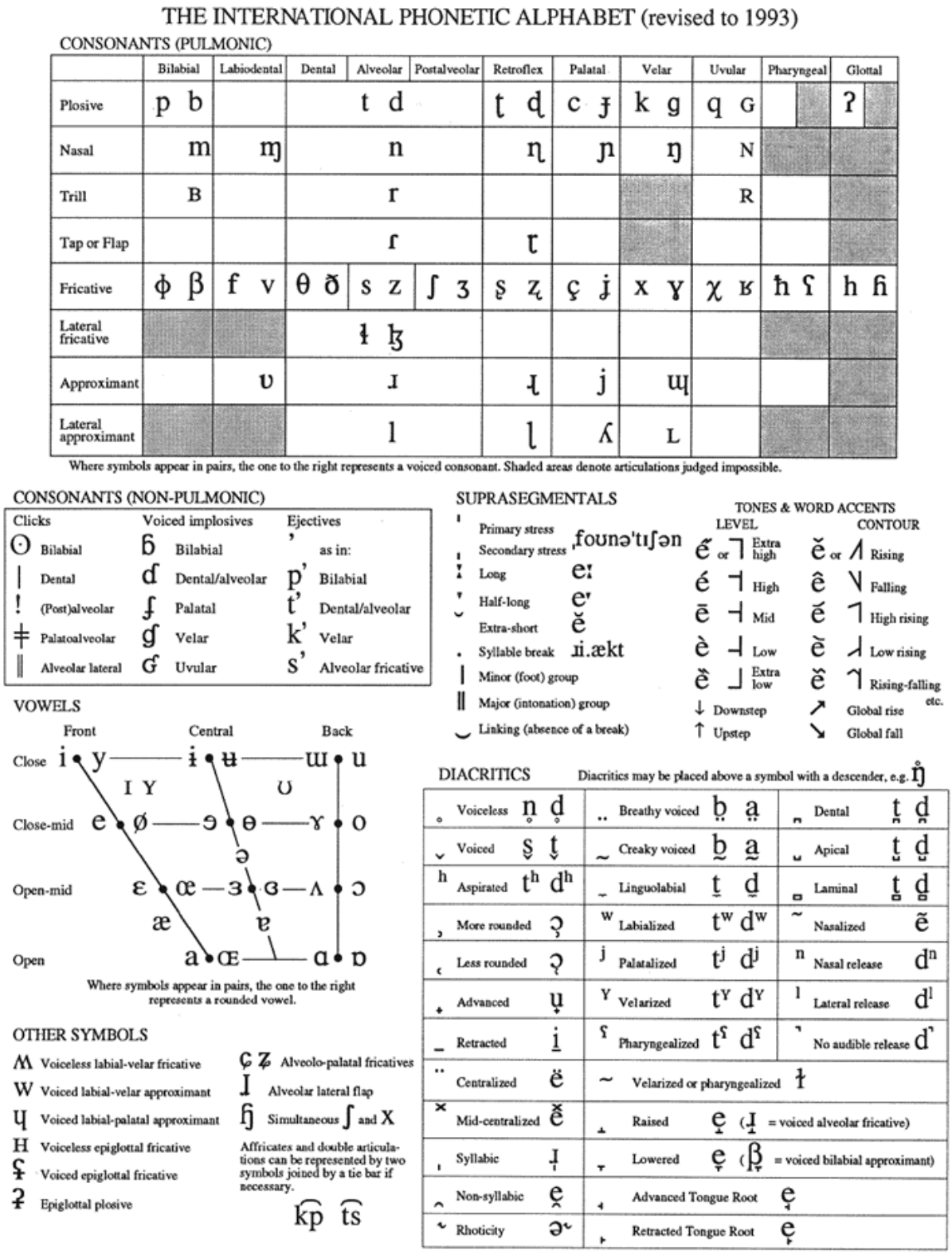


Figure 2.2: IPA chart

This can also be seen in the vowel triangle in Figure 2.2. The triangle represents the position of the tongue, horizontally from front to back, and vertically from close to the roof of the mouth to open where the tongue is as far away as possible from the roof. There are two vowels per tongue position. The one on the left is with lips neutral, while the one on the right is with lips rounded.

Most modern speech recognizers can recognize a lot of words, as long as they are transcribed in phonemes. This means that a word is described as the sequence of mouth positions that produces the word as spoken. A list of words with such corresponding transcriptions is called a lexicon.

Acoustics

Articulation is useful to know how speech is produced, but a speech recognition system does not receive information of the movements of the mouth. Speech recognition systems use the stream of speech itself as data.

Speech is an analog signal: a continuous flow of sound waves and silence. Four important features of acoustic analysis of speech are:

- Frequency
- Amplitude
- Harmonic structure (tone vs. noise)
- Resonance

Sound is produced by movements of air molecules. These moving air molecules come into contact with the eardrums which in turn vibrate. The vibrations of the eardrums are then interpreted by the brain which are interpreted as sound.

The simplest of “pure” sounds are produced for example by a tuning fork. A tuning fork vibrates back and forth at a single steady rate in a sinusoidal manner. This causes the air molecules around the fork to move and vibrate. The vibration spreads throughout the air and eventually reaches the eardrums.

The rate at which the fork moves back and forth is called the frequency. The maximum deviation of the fork from its “neutral” position (when it’s not vibrating) is called the amplitude. High pitch sounds are produced by high frequency. Loud sounds are produced by high amplitude.

Pure tones in speech are rare. Often there is a fundamental frequency that corresponds to our perception of the pitch of the sound, overlaid with secondary frequencies. For speech, the fundamental frequency is the rate at which the vocal cords flap against each other when producing a voiced phoneme. Added to the fundamental frequency are other frequencies that contribute to the quality or timbre of the sound. Those additional frequencies make it possible to distinguish voices of specific individuals.

Some bands of secondary frequencies play a central role in distinguishing one phoneme from another. They are called formants and are produced by resonance. Resonance it the ability of a vibrating source of sound to cause another object to vibrate. Sound boxes of a musical instrument make use of resonance and are designed to respond to and amplify sound vibrations of specific frequencies. Such a sound box is called a resonating chamber. The throat, the mouth and nose are also resonating chambers and amplify bands of formant frequencies contained in the sound wave generated by the vocal cords. Because humans have control over the size and shape of the mouth and the flow of air to the nose (by moving the soft palate), the amplified formant is different for every change of size, shape etc. Formant patterns are strongest for vowels and weakest for voiceless consonants.

Vowels can be distinguished by their characteristic formant patterns. When viewed as a waveform vowels are cyclic in nature. Sound waveforms which are acyclic are called noise. Examples of noise are hisses, crashes, and voiceless phonemes such as /s/ and /p/. All consonants and semi-vowels have noise components. For example, a prolonged pattern of noise is indicative of a fricative consonant. Silence followed by a burst is characteristic of a

stop consonant. Consonants are also revealed by a shift in formants when the articulators move from the vowel to a consonant and from the consonant to the next phoneme.

The frequencies and shifts in formants can be easily seen in a spectrogram. The horizontal axis represents time, the vertical axis represents the frequency. The darker areas represent the intense presence of a frequency at that moment. Bands of dark areas represent formants. Noise areas often lack clear formants and it is hard to identify which consonant it is. But the transition from the previous or to the next clear formant is indicative of what the consonant was. Other information which can be used are the duration, the onset and amplitude of the sound.

Examples of how formants, amplitude and other features can be visualized can be found in in Chapter 4 when the sound analysis program Praat is described in detail.

Perception

Humans can quite easily understand speech under difficult circumstances. Understanding the human auditory system will lead to improvements in speech recognition, but a lot of research is yet to be done. Research until now suggests that the auditory system is strongly adapted to speech. Humans perceive sounds from 20 Hz to 20,000 Hz, but are most sensitive to sounds from 1,000 Hz to 6,000 Hz. The human ear is also more sensitive to small changes in certain frequency ranges than others. Studying human perception has already led to improvements such as the mel scale, which represents the pitch perception patterns of the human ear.

Speech preprocessing

Speech data is analog, and when recorded digitally over a microphone produces a lot of bits and bytes. Speech recognition is not applied to this stream of data directly, instead it is preprocessed. The main reason is computational efficiency - reducing the amount of data which needs to be analyzed because it is simply redundant or not relevant - and is necessary in order to real time speech recognition. Another reason is that some information will interfere in the recognition process, such as distortion and non speech noises.

Preprocessing the speech signal should:

- Include all critical data. All data which is necessary to discern one word from another must be preserved.
- Remove redundancies. All data which is not necessary to discern one word from another must be discarded. This can be silences at beginning and end, or frequencies which the human ear cannot hear.
- Remove noise and distortion. Most recordings will have at least some minor noise. Using advanced signal processing techniques these noises can be reduced to improve recognition.
- Avoid introducing new distortions. Any transformation of the speech signal (e.g. to remove noise) should not make it more difficult to recognize what is said.

The most critical formants and much of the characteristic noise of fricatives are in the range of 100 Hz to 3,100 Hz. Any filtering done should keep at least this frequency range intact. Usually one should record at a rate of at least two times the frequency of interest. To capture sounds of 3,100 Hz, one would record it digitally at a rate of at least 6,200 samples per second.

After this is done, the audio is split into frames or analysis windows of 10 to 50 milliseconds each. The preprocessor extracts frequency information of that frame using a spectral analysis and also records changes that occur from one frame to the next. The most commonly used spectral analysis approaches are based on linear predictive coding (LPC), which estimates acoustic features or parameters from the current sample using parameter values from the previous. LPC is accurate and is faster than most other approaches. Each frame can then be represented as a feature vector consisting of a series of numbers.

2.2.2 Hidden Markov Models

The most successful approach to date for speech recognition is based on Hidden Markov Models. Hidden Markov Models were around for some time, but it wasn't until people like Lawrence Rabiner [Rabiner89] published some articles about its uses in speech recognition before its use became widespread. Hidden Markov Models and the associated algorithms are not explained in detail here. This section merely serves to give a general idea how Hidden Markov Models are used in speech recognition.

To understand what a Markov Model is, consider a system that at any time as being in a set of N distinct states, S_1, S_2, ..., S_N, as illustrated in Figure 2.3 ($N = 5$). At regularly spaced discrete times the state undergoes a change, possibly back to the same state. The times associated with state changes are denoted by $t = 1, 2, ...$, and the actual state at time t is denoted by q_t.

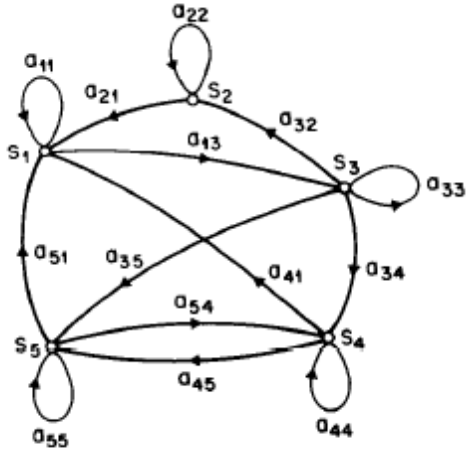


Figure 2.3: An example of a Markov chain

For a discrete first order Markov chain the current state is only dependent on the predecessor state, and the state transitions each have certain probabilities. Let a_{ij} be the probability that the state changes from state i to state j and the probability distribution $A = \{a_{ij}\}$:

$$\begin{aligned} a_{ij} = P[q_t = S_j | q_{t-1} = S_i], \quad 1 \leq i, j \leq N \\ a_{ij} \geq 0 \\ \sum_{j=1}^{N} a_{ij} = 1 \end{aligned} \tag{2.1}$$

In the simplest case, the state is observable. If the system for example produces an observation sequence $O = \{S_3, S_3, S_3, S_1, S_3, S_2, S_3\}$ then the probability that the model generated this observation sequence is:

$$\begin{aligned} P[O|Model] &= P[S_3, S_3, S_3, S_1, S_3, S_2, S_3 | Model] \\ &= P[S_3] \cdot P[S_3|S_3] \cdot P[S_3|S_3] \cdot P[S_1|S_3] \cdot P[S_3|S_1] \cdot P[S_2|S_3] \cdot P[S_3|S_2] \\ &= \pi_3 \cdot a_{33} \cdot a_{33} \cdot a_{31} \cdot a_{13} \cdot a_{32} \cdot a_{23} \end{aligned} \tag{2.2}$$

Where π is the initial state distribution. $\pi = \{\pi_i\}$

$$\pi_i = P[q_1 = S_i], \quad 1 \geq i \geq N \tag{2.3}$$

Things are more difficult when the states are not seen directly. States can produce different observations and the same observation can be produced by different states. A simple example is a coin toss. Different types of coins (fair, biased to heads, biased to tails) are each represented by a state. An unknown coin is tossed, and this produces an observation sequence of heads and tails. Based on this sequence we can hopefully determine in what state the model is, and thus what type of coin was thrown up.

The earlier Markov model is now extended with some extra parameters to get a Hidden Markov Model $\lambda = (A, B, \pi)$. The symbol probability distribution $B = b_j(k)$ is the distribution of observations given that the system is in a certain state. M denotes the total number of distinct observations the state can produce, and the types of observations are marked by $V = \{V_1, V_2, ..., V_M\}$.

$$b_j(k) = P[V_k \text{ at time } t | q_t = S_j], \quad \begin{matrix} 1 \geq j \geq N \\ 1 \geq k \geq M. \end{matrix} \tag{2.4}$$

To apply this model to speech recognition, in its simplest form, imagine that a state is the intended phoneme being produced by a speaker, determined by the position and shape of the vocal apparatus, and an observation is the realization of that phoneme (as sound). There is usually no one to one correspondence between observation and states. This is because different speakers pronounce a phoneme differently, co-articulation in transitions between phonemes etc. So one state can have more than one realizations or observations. The Hidden Markov Model records among others, things like how often does a phoneme follow another given past phonemes, and how often a certain realization can be heard given a phoneme. In theory all observations can be produced by every state, even if a lot of times this probability is near zero. A speech recognizer can only hear the realization of the sound and from that it must compute sequence of states (or phonemes) which most likely produced it.

There are three algorithms which can solve the following problems when using Hidden Markov Models:

- The Forward-Backward Procedure:
  Given the observation sequence $O = O_1 O_2 ... O_T$ and model $\lambda = (A, B, \pi)$ how do we efficiently compute $P(O|\lambda)$, the probability of the observation sequence, given the model? This is an evaluation problem. Given a model and observation sequence, how do we compute the probability that the sequence was produced by the model. In speech terms this could be translated as, given what we know about which phonemes tend to follow another, and what the different realizations of phonemes are, how likely is it that a sound is produced such as it is heard?

- The Viterbi algorithm:
  Given the observation sequence $O = O_1 O_2 ... O_T$ and model $\lambda = (A, B, \pi)$ how do we choose a corresponding state sequence $Q = q_1 q_2 ... q_T$ which best explains the observation? In speech terms this could be translated as, given a series of phoneme realizations, what is the most likely sequence of phonemes the speaker was trying to pronounce?

- The Baum Welch algorithm:
  How do we adjust model parameters $\lambda = (A, B, \pi)$ to maximize $P(O|\lambda)$? How do we maximize the probability of the observation sequence given the model? In other words, how do we train a speech recognizer such that it recognizes as many observation sequences as possible?

Because observations in speech recognizer (represented as a feature vector) are not discrete but continuous over time, the picture painted in this section is slightly skewed. In order to use Hidden Markov Models in speech recognition more sophisticated versions are used such as Continuous Density Hidden Markov Models. Also phonemes are often represented by more than one state. One common configuration is to split a phoneme in 3-states, namely the on-set, sustainment and off-set of the phoneme, because the length of a phoneme can be variable, for example when a vowel is elongated.

2.2.3 Training a speech recognizer

The training of a speech recognizer based on Hidden Markov Model generally happens in several stages. A common procedure is as follows.

- Data collection:
  In this stage speech data is gathered. To train a good speech recognizer it is desirable that recordings are made from a lot of speakers, male and female, of all ages, and from all dialect areas of the country. This way the speech recognizer will become more robust and able to cope with speaker variability. Furthermore it is important that speech data gathered in a consistent manner using the same recording conditions. Different things can be recorded depending on the goal of the speech recognizer. For general speech recognizers it is important to record sentences and words which cover all the sounds (phones, phonemes) of the language within a variety of words, to cover as many transitions between sounds as possible.

- Labeling:
  After speech data is collected, the data must be transcribed. This is time consuming, and requires that people listen to each recording, and write down exactly what is heard, so basically everything which is said, including mouth noises and other sounds. At this stage recordings which are unsuitable for training, due to for example having too much noise, can also be filtered out. When only words are written down this is called an orthographic transcriptions. Sometimes recordings are manually transcribed phonetically as well, where each sound is identified by expert listener as the right phone, thus making it theoretically possible to transcribe recordings in languages which are unknown to even the listener. Usually however a crude phonetic transcription is made by using the orthographic transcription and a lexicon, so an expert listener is not always needed.

- Alignment:
  The transcriptions made in the labeling phase must now also get the times when a sound starts and ends within a recording. The phonetic alignment will make it possible to split the recording into phoneme segments, which will in turn be used for training. Alignments can be made by the person doing the transcription, but it is now common practice to use automatic alignments, for example using forced Viterbi alignment. Viterbi alignments are less accurate and require a lot of iterations for the alignment quality to improve when starting from scratch. However since speech recognition has been around for a while, alignments can be made faster by bootstrapping from existing models, even if they were initially meant for another purpose.

- Training:
  Each segment is represented as a sequence of features which are obtained by analyzing the speech signal using special techniques such as PMVDR. All the segments belonging to the the same phoneme are used to train the HMM model for that phoneme. It might require several alignment and training rounds before the recognition performance is acceptable for use.

2.3 Multilingual recognition and techniques

There is a lot of transcribed speech data available for languages whose speakers are of significant number or whose speakers have sufficient economical interest, for example English, French, German, Japanese, Chinese. Successful state of the art speech recognition applications have been built for these languages, but for many languages such data is not available, and for some research areas is near impossible, such as non-native speech. State of the art speech recognition applications have been built for these “in-demand” languages, but it can happen that at some point, for example due to political or economic reasons, there can be immediate demand for the same application but in a language which has almost no data available.

Recording and transcribing speech for a new language is expensive and takes a long time, thus therefore methods are sought to reduce the amount of data needed from the new language using existing recorded data from languages which do have large amounts. There is a huge amount of overlap in sounds from different languages, and ways are

being sought to exploit that. Some researchers believe that it is possible to make a “universal recognizer” which can recognize all the sounds of all spoken languages of the world using only training data from a limited number of languages which together have a broad coverage of phones.

2.3.1 Multilingual recognition

When making speech applications suitable for new languages, there are 3 methods according to [Uebler01]: porting, cross-lingual recognition and simultaneous speech recognition. These approaches differ according to the available training data for the development of the recognizer and the application the recognition system is designed for.

Porting

A speech recognition system designed for a language is ported to another language in order to be used in that language. The dialog system is the same for the new language, but the training data are from the new language. This is often done for dictation of systems which will be used also for recognition of another language. Difficulties to cope with are the characteristics of the new language. For example, if a system is ported to German, it must deal with the compound words, or, for French, with homophones.

A system that has been developed for one language must be optimized to be used in another language, and some of the algorithms have to be adapted to work well in that new language. There should be enough training data in the new language to completely establish a system in the new language. The acoustic models, grammars and language models of the new language are trained with data of the new language, and these replace the ones of the original dialog system.

(Multilingual) Cross-lingual recognition

This approach for the most part follows the porting approach. The difference is that insufficient training material is available to train the recognizer in the new language. Thus, in cross-lingual recognition, methods must be found to use training material of one or more (using multilingual acoustic models) source languages for modeling the acoustic parameters of the target language. Thus a recognizer for one language is bootstrapped from training data from other languages. Optionally, an adaptation with a small amount of data from the target language takes place.

First the languages used for training of the recognizer must be determined. The language(s) leading to the best recognition performance on the new language must be found. A relation between the languages used for training and the language to be recognized has to be selected. As in the porting scenario, the algorithms of the recognizer are adapted to the new language. Furthermore, training material of the source language(s) is used for the parameter estimation of the recognition system for the target language.

For this approach, a relation between the languages and the language to be recognized must be found in order to model the parameters of the language to be recognized using the parameters of the training language. One focus will be put on the acoustic units if the training and recognition languages.

One main problem is to determine identical acoustic units or to model existing acoustic units in a way that good recognition can be provided. One can think of a couple of different measures to determine the similarity of phones across languages. If some adaptation material of the new language is available, the best adaptation algorithm, that is able to optimally use such data, must be found.

Simultaneous multilingual recognition

Applications with this approach allow for recognition of utterances of different languages at the same time. The system does not know in which language an utterance is spoken. Training data are available for each language. The result is a single recognizer for all involved languages together.

There are two main strategies for simultaneous multilingual speech recognition, with explicit language identification and with an implicit identification of the spoken language. The first strategy performs language identification on the speech signal. After identification of the language, the speech recognition system of the identified language is activated and the utterance is recognized. The advantage of this strategy is a performance that is identical to monolingual recognition as long as the language identification step is performed without errors.

The other strategy performs an implicit language identification. The words of all involved languages can be recognized equally or by language model distribution. A language model which allows a transition between several monolingual language models is also possible. The spoken language is determined according to the recognized words. For this strategy the same acoustic model is used for several languages, instead of a cluster of different monolingual models.

The strategy performing best within this approach may vary depending on the given languages and data. For example, if there is only a little data for one language, acoustic units may be shared across languages. If the languages are similar or if there are non-natives among the speakers, multilingual units may lead to a better performance. On the other hand, if the languages are similar, it may be more useful to separate the languages as much as possible in order to avoid confusions among the languages.

2.3.2 Phone substitutions

Each language has its own characteristic set of phonetic units. In order to perform cross-language or simultaneous multilingual speech recognition, a relation among the acoustic inventory of the involved languages must be found.

In cross-language recognition there are trained phones of one languages, and untrained (or not well trained) phones of the language to be recognized. The sounds of each of the languages to be recognized must be replaced by the most similar trained sound of the other language. In simultaneous multilingual speech recognition, monolingual phones are trained for each of the languages to be recognized. Some of the sounds of different languages may be similar enough to be represented as the same sound in order to reduce the number of parameters of the recognition system.

In speech recognition usually phonemes are used for representation of the pronunciation of the words. Phonemes are defined as the smallest unit in speech leading to a difference in the meaning of a word, whereas phones are characterized according to their acoustic properties. Thus a phoneme may be realized by different phones. The relation between phones and the phones of a language differs across languages. In one language 2 phones might be the same phoneme, but in another they might be 2 different phonemes.

In order to establish relations between the sounds of different languages, for each sound of the recognition language the most similar sound of the training language must be determined. This can be a 1:1 mapping, but also an n:1 mapping where a 2 or more sounds in the training language are used to estimate the values of phone. There are 3 approaches to find these relations phonemes and will be explained in the following sections.

Na(t)ive approach

This approach of phone substitution follows the principle a non-native follows when speaking a second language. A non-native might do his best to use the phonetic inventory of the spoken foreign language. The accent of a non-native speaker is, among others, determined by the phonetic inventory of his mother tongue he uses when speaking the foreign language. A non-native tries to use the phonetic inventory of the new language. Still, he may fall back to his native phone inventory in stress conditions or with difficult words.

For some sounds there is a similar one in the native language of the speaker, and the speaker may not learn the small difference for the new sound and will only use the sound of his native language. This may happen for vowels with a small difference in the mean values in their formant frequencies between languages. Thus a non-native might pronounce the foreign language using the correct phonemes, using the closest phoneme in his own phonetic inventory, or a combination of both.

Phonetic approach

This strategy follows principles in the production of sounds in the human vocal tract. When describing the production of sounds, usually the place and manner of the production are determined. The place of production describes where obstacles are put in the air flow and which organs are involved in the production of the sound. The manner of production describes how the obstacles act, for example, if a complete or partial closure of the air flow is caused.

Consonants can be distinguished with regard to manner (stop, fricative, approximant, lateral, rhotics, etc.) and place (labial, dental, alveolar, palatal, velar, alveolar etc.). Another criterion is the voicing of consonants which can be either voiced or unvoiced. For vowels, different tongue positions are distinguished such as front, central, back, and for the opening of the mouth (close, close-mid, open-mid, open) as well as between rounded and unrounded to describe the shape of the lips.

Finding differences between consonants in different languages is easier than finding differences in vowels in different languages, because for example the position of the tongue can gradually change. To find a good mapping, the trained sound that matches the largest number of phonetic features is substituted for the untrained sound.

Data driven approach

This approach determines the similarity among phones with the data given by a trained speech recognizer. This approach is only possible if there is at least some data available for the new language to be recognized in cross-language recognition. For simultaneous recognition it can be used to determine the similarity of sounds and to set a threshold such that any degree of joining sounds into a multilingual acoustic set can be realized.

Measures for the similarity can be estimated from the Gaussian densities or the codebook parameters of a trained recognizer. Therefore a recognizer must be trained with all languages, and for all observations of a language-dependent sound the similarity of the parameters must be estimated. According to a distance measure the most similar units may be joined. This merging of units can happen in one ore more steps and it may also be allowed to split units. The advantage of this approach is that there is no human knowledge or manual work necessary to estimate similarities.

2.3.3 Multilingual acoustic models

Methods to find similarities between phones from different languages were discussed in the previous section. By finding similarities in sounds in languages a global phoneme set can be created. By relabeling training data from various languages to this global set, compact acoustic models can be made which can simultaneously recognize languages it is trained on, or which can be used to do cross-lingual recognition because of its broader phoneme set. This compactness is also beneficial to performance, because less resources are needed to hold the models for the global set in memory than phoneme models for all languages separately.

Training

The porting scenario is of limited interest because the amount of data is limited. The cross-lingual and simultaneous recognition scenarios are interesting since they make use of existing Dutch and English data. Cross-lingual recognition can be done with mono or multilingual acoustic models. Making monolingual acoustic models is a

known practice. Therefore the process of creating multilingual acoustic models is looked at here in more detail in this section.

[Schultz98a] introduce 3 acoustic model combination methods to share data. [2]

- In the ML-sep combination method each language specific phoneme is trained solely with data from its own language. No data are shared across languages to train the acoustic models. Thus the number of phonemes in the resulting model is the sum of phonemes of the source languages.
- In the ML-mix combination method data across different languages is shared to train acoustic models of polyphonemes. Phonemes of different languages but with the same IPA-symbol share the same Gaussian mixtures and weights. Context dependent models are made using linguistic questions. No information about the language the training data is from is preserved.
- The ML-tag combination is almost the same as ML-mix, in the sense that they both share all the training data and use the same clustering procedure. But for ML-mix the training data are only labeled by phoneme identity, whereas for ML-tag the training data is labeled by both phoneme and language identity. The clustering procedure is extended by introducing questions about the language and language groups to which a phoneme belongs. The Gaussian components are shared across languages as in the ML-mix method but the mixture weights are kept separately. Therefore, the relative importance of phonetic context and language membership is resolved during the clustering procedure by a data-driven method.

[Köhler01] also compares three methods to share data:

- The most simple way, IPA-MAP, where all language dependent phones are mapped to IPA phones, and those with the same IPA symbol share data. Downsides mentioned are that IPA symbols do not model language dependent properties of the sound, such as speaking rate, different phonetic context dependency and prosodic features. A Viterbi training is done on all data using the mapped phones. This method is very similar to Schultz and Waibel's ML-mix.
- MUL-CLUS, or multilingual phone clustering, use a bottom-up cluster algorithm to map the language-dependent phone models to the multilingual set. First language-dependent phone models must be trained, then a similarity between two phone models has to be defined, for example the log-likelihood LL-based distance measure. Initially each phone is a cluster. Clustering is done in a number of steps and at each cluster step the most similar pair of clusters is merged to a new cluster. Because joined clusters are not real phones it is hard to compute the distance between them, thus clusters are compared by comparing the distance of the the phones they are composed and the maximum distance between them is taken. Clustering is done while the distance is below a threshold setting. The clustering results in a phone mapping, and this mapping determines which phones will share data when the Viterbi training is run.
- The previous two approaches try to create complete multilingual phone models. This means that all parameters of one model are shared across different languages. On the other hand there are several language-specific properties of the sounds which exist due to speaking style and rate, prosodic features, allophonic variations, but these are not modeled in the previous methods. In IPA-OVL, or IPA-based density clustering, the assumptions is that there are some language-independent phone realizations. First language-dependent models are trained, where each language-dependent phone consists of three segments, each modeled by a mixture density. In the second step the mixtures of the language-dependent segments which belong to the same IPA-based phone are collected in one common pool of densities. Then a hierarchical agglomerative clustering algorithm is applied to find and merge similar densities. After each clustering step the overall number of densities is reduced by 1, and stops after a predefined number of clusters is reached. After clustering for each IPA-based phone there is a multilingual mixture density. Whereas the mixture density has multilingual regions, the mixture weights are still language dependent. Finally the parameters of the multilingual mixture densities are re-estimated by running a Viterbi training over it. This method achieves

[2] In this paper called Lang-sep, Lang-mix, Lang-tag, but in later papers ML-sep, ML-mix, ML-tag respectively.

a huge reduction of parameters in the multilingual system, while still retaining language-specific properties and automatic detection of multilingual realizations exploit the acoustic-phonetic similarities in an optimal way.

[Walker03] emphasize consistent labeling across all corpora used so that phonemes can be shared optimally. They introduce a procedure they call "bottom-up, two-level forced alignment", which consists of choosing a reference corpus and repeatedly adding new languages and corpora while making the labeling of the new corpus consistent with the previously added corpora. To make a new corpus consistent with the reference set, they used the hand marked labels or first pass forced alignment, followed by a second alignment. In the second alignment, phone models trained on the reference corpus are used, while some phones are allowed to be renamed based on allophonic variation that exists in the language.

Performance

Various papers have been written on multilingual acoustic models, and for each purpose the best method to use differs. As for phoneme mapping, the most common method used are the phonetic and native approaches, since these are simple and require no speech data to perform. Data-driven methods to find phoneme mappings are only possible when sufficient data is available, but are not really proved to be better than the other approaches. The best performing acoustic modeling technique differs from task to task and will be discussed next.

For simultaneous recognition of languages the multilingual acoustic model is trained on, it is best that information about the language is preserved. [Schultz98a]'s ML-tag and [Köhler01]'s IPA-OVL are good examples, but also in [Cohen97] the language information preserving approach is slightly better than not preserving the information. However these techniques are not up to par with the performance of traditional monolingual recognizers.

For multilingual cross-lingual recognition and bootstrapping new languages [Schultz98a] shows that simply sharing training data without preserving language information works best, and this works better than performing cross-lingual recognition from only one language. [Uebler01] found similar results regarding cross-language recognition from a mono or multilingual model. [Kunzmann04] shows that by adding more and more training data from other languages step by step gradually improved recognition of Spanish, which was not included in the training.

For non-native recognition the performance is as follows. In [Kunzmann04] a name dialing application was built with prospective users from 6 languages which pronounce names both from their native language and from the other languages. They show that adding more and more training data from other languages step by step gradually improved recognition of both native and non-native speech. [Fischer03] show that performing pronunciation modeling for phonemes not in the native set in combination with multilingual acoustic outperforms adaptation techniques on an English digit recognition task with speakers from various countries. [Stemmer01] implemented a movie name recognition service for a German audience with many of the titles having English names, thus non-native pronunciations. They compared several methods: knowledge based mapping of English phonemes to German with a German recognizer (baseline), language dependent models which are only trained on either German or English sounds spoken by Germans, knowledge based merging using IPA correspondences to share training data, and data driven merging of phoneme sets. The baseline performed the worst, and they got the most improvement using knowledge-based merging of phonemes, followed by data-driven merging of phonemes. [Uebler99] found that it is useful to add speech data of the language which the speaker is native from to the training in order to recognize non-native speakers of that origin better.

2.4 Machine Learning and Call classification

Machine learning is an area of artificial intelligence concerned with the development of techniques which allow computers to "learn" by analyzing huge data sets. Machine learning has a wide spectrum of applications including

search engines, medical diagnosis, detecting credit card fraud, stock market analysis, classifying DNA sequences, speech and handwriting recognition, object recognition in computer vision etc.

Some machine learning systems attempt to eliminate the need for human intuition in the analysis of the data, while others adopt a collaborative approach between human and machine. Human intuition cannot be entirely eliminated since the designer of the system must specify how the data are to be represented and what mechanisms will be used to search for a characterization of the data. Machine learning can be viewed as an attempt to automate parts of the scientific method.

There are different types of Machine learning algorithms depending on the desired outcome of the algorithm. Common algorithm types include:

- supervised learning, where the algorithm generates a function that maps inputs to desired outputs. One standard formulation of the supervised learning task is the classification problem: the learner is required to learn (to approximate the behavior of) a function which maps a vector into one of several classes by looking at several input-output examples of the function.
- unsupervised learning, which models a set of inputs: labeled examples are not available.
- semi-supervised learning, which combines both labeled and unlabeled examples to generate an appropriate function or classifier.
- reinforcement learning, where the algorithm learns a policy of how to act given an observation of the world. Every action has some impact in the environment, and the environment provides feedback that guides the learning algorithm.
- transduction, similar to supervised learning, but does not explicitly construct a function: instead, tries to predict new outputs based on training inputs, training outputs, and new inputs.
- learning to learn , where the algorithm learns its own inductive bias based on previous experience.

The performance and computational analysis of machine learning algorithms is a branch of statistics known as computational learning theory.

In a call classification context, the machine learner must find a function which maps the spoken utterance to an appropriate class so that a call-center agent specialized in the class can take care of the caller's needs. The classifier does not deal with the audio data directly, but uses the output of speech recognizer. In literature there are several machine learning algorithms used for call classification. Here are just some:

- Vector based call routing ([ChuCarroll99, Carpenter98]): Words are converted to more basic forms using morphological analysis (verbs to stem forms, plurals to singular etc), and stopwords (words to make the sentence grammatically correct but with no or little meaning, such as "the") are filtered out. The remaining words are called terms and are used for classification. For training, a matrix A is constructed with size $m * n$ with m being the number of terms and n being the number of destinations and $A_{t,d}$ represents the frequency of term t in calls to destination d. The rows are then normalized to unit length, and IDF (inverse document frequency) scaling is applied. For classifying, a sentence is transformed into a vector similarly and a distance measure is calculated for example using the cosine distance (dot product). The result is a score (confidence) of each destination, which can be compared to a threshold for making a decision, or which can be sent to the disambiguation module if more categories have high scores.
- AT&T's "How may I help you", "Help Desk" and BoosTexter. The BoosTexter algorithm developed by Schapire and Singer ([Schapire00]) was originally written to classify text, but has been tested successfully for classifying speech as well ([Schapire00, DiFabbrizio02, Rochery02]). A general outline of the algorithm follows.

  Boosting makes a strong classifier out of several weak ones by "boosting" it with weights. A weak classifier is a classifier which perform slightly better than random. In the case of text, BoostTexter checks for the presence or absence of a word (or n-gram). The algorithm starts out with a weighing for each class.

At each training round a decision stub (e.g. is the word present or not?) is added. Each stub gives a different weighting for each class depending on the result of the question. For all classes, all the weights of all the decision stubs are added to the base class weights, and the the class with the highest score 'wins'. The boosting training algorithm recalculates the weights by focusing on the examples which are hardest to classify, giving certain questions an extra "boost".

In this thesis classification will be attempted with BoosTexter, as well as with RIPPER. RIPPER (Repeated Incremental Pruning to Produce Error Reduction) was developed by W. Cohen [Cohen95] and is a rule learning algorithm. The resulting classifier is a series of if-then-else rules and is easy to understand by humans. RIPPER is a modification of IREP (Incremental Reduced Error Pruning). In general lines RIPPER works by randomly splitting the data set in a grow set (2/3) and prune set (1/3). Rules are grown, greedily, adding conditions with the largest information gain in the grow set compared to the rule without that condition, making the rule more specialized. After the rule is grown, it is pruned (simplified) using the prune set, making the rule more general. The resulting ruleset is then optimized, this time using a different growing and pruning process with the existing rules as base. A rule learning system is attractive because the resulting classification algorithm is a series of if-then-else rules which test for the presence or absence of a word and are easy to understand and modify by humans. JRIP is WEKA's implementation of the RIPPER algorithm. The exact algorithms for BoosTexter and RIPPER are given in Chapter 6.

It is hard to determine which is the best classifier in any situation based on the literature alone since there are no direct comparisons, and often they are tested on different kinds of tasks, different amounts of training material. Thus it is not straightforward which algorithm will perform best and new testing must be done.

Chapter 3

Corpora

3.1 Introduction

The various speech corpora used in this thesis are described in this chapter. The process of creating a speech corpus is laborious, expensive and time-consuming. Corpora can be made for various purposes. There are corpora to cover different recording conditions, microphone and telephone, different languages and dialects, general language corpora and task specific, for use in speaker verification, adult or children's speech, speaker recognition or speech recognition etc. Furthermore there are some corpora collected to study anomalies, such as non-native speech or noisy speech.

For telephony speech recognition generally the goal is to recognize as many people as possible. Therefore generally great care is taken in selecting speakers so that it is representative of the population. One would for example choose speakers in proportion to all major geographic or dialect areas. In each area there must be roughly the same amount of male and female speakers of people in different age groups, although usually there is more interest in adults than in children. Finding the right people in sufficient amounts willing to donate their time and voice to science (or commercial corpus collectors) is another problem. Usually they must be approached with mailings or by calling them up. If recordings of several thousands of people must be made it is an enormous task of organizing recording sessions, and registering their age, dialect, gender etc. Luckily for telephony speech corpora the recordings can be made over the telephone. Respondents can be instructed to dial a telephone number so that their properties can be registered and recordings can be made.

After recordings are made, they must be transcribed and verified to see if the speakers followed their instructions properly. Also some recordings will not be suitable, for example due to having too much noise and being unrecognizable, so these must be eliminated. Sentence transcribers might need a little training first so that their transcriptions and labels are consistent. Phonetic transcriptions on the other hand require expert listeners that can classify each sound as a series of phonemes, even if they don't know the words being said, for example due to it being in an unknown language.

The material that is recorded must also be selected carefully. For general speech corpora they usually consist of phonetically rich sentences taken from newspapers or articles. This in the hope that all phonemes and phoneme sequences will be covered sufficiently. For task specific corpora only recordings are made for that task. It can be very narrow for example only digits for digit recognition, only zip-codes for zip-code recognition etc., but also broader where people answer open questions in different ways.

Corpora of 3 languages are used in this thesis, some for training, some for testing. In some cases the full corpus was not available, or only certain parts were interesting enough. This is why information is given only about the parts of the corpora which were actually used in Chapter 5. This is summarized in tables 3.1 and 3.2.

The following sections provide more detailed information about each corpus such as information about the people

Table 3.1: Overview of training corpora

Corpus	Language	Utterances	Hours	Type
Dutch Polyphone	Dutch	42101	48.92	Short utterances
TIMIT	American English	3696	3.27	Sentences
Swiss French Polyphone	Swiss French	7961	10.60	Sentences

Table 3.2: Overview of testing corpora

Corpus	Language	Utterances	Type
VoiceConnect97	Dutch	5300	Short utterances
TIMIT	American English	2604	Sentences
DDAC2000	Dutch mixed with English words	1018	Short utterances

who recorded, what types of utterances it contains etc.

3.2 Dutch Polyphone

The Dutch Polyphone corpus [denOs95] contains the recordings of 5,050 Dutch speakers recorded over the fixed telephone network (ISDN line). The corpus comprises 222,075 speech files (44 or 43 items per speaker), which all have been orthographically transcribed. The data was collected in 8-bit A-law digital form.

The corpus contains both read and extemporaneous items.

The recorded items consist of isolated digits, numbers (one telephone number, two bank accounts or credit card numbers, and the participation number), a postal code, guilder amounts, time, date, amounts, application words, sentences with application words, phonetically rich sentences, spelled words, city names.

Several questions were asked to get some spontaneous speech (e.g. Is Dutch your native language?, Did you ever live in another country than the Netherlands? In which cities did you grow up? Are you a man or a woman? Are you calling from your home phone?, etc.). Tables 3.3 and 3.4 show example phonetically rich sentences and application words respectively.

Table 3.3: Example Dutch Polyphone sentences

Natuurliefhebbers willen Lelystad omzetten in een oerbos.
Grindwinner Machiels lijkt aan zijn saneringsplicht te ontkomen.
Wij willen invloed uitoefenen bij toekomstige verkiezingen.
Tussen de auteur en zijn personages ontstaat een gevoelsmatige band.
Zij heeft de hulp van haar ouders nodig om zich te identificeren.

3.3 TIMIT

The TIMIT speech corpus [Garofolo93] was collected by Texas Instruments (TI) and the Massachusetts Institute of Technology (MIT). It contains a total of 6300 phonetically rich utterances, 10 spoken by each of 630 speakers from 8 major dialect regions of the United States. A speaker's dialect region is the geographical area of the U.S. where they lived during their childhood years. There is a separate "dialect region" for people who moved a lot during childhood. The average male to female ratio is around 7 to 3. TIMIT is divided into a test and train set. The test set consists of the recordings of 168 speakers. Except for two sentences which were recorded for every person, there is no overlap in sentences between test and train set, nor is there an overlap in speakers. Some

Table 3.4: Example Dutch Polyphone application words

Nieuwjaar	Ontvangstbevestiging
Afstemmen	Negen
Keuze	Schiphol
Overzicht	's Gravenhage

example sentences spoken in the TIMIT corpus are given in Table 3.5. The recordings were made over microphone and the utterances are read, not spontaneous.

Table 3.5: Example TIMIT sentences

Grandmother outgrew her upbringing in petticoats.
At twilight on the twelfth day we'll have Chablis.
Catastrophic economic cutbacks neglect the poor.
Ambidextrous pickpockets accomplish more.
Her classical performance gained critical acclaim.
Even a simple vocabulary contains symbols.

3.4 Swiss French Polyphone

Like the Dutch and German polyphone corpora, Swiss French Polyphone [Chollet96] is a Polyphone-like database recorded in Switzerland to cover the French language as spoken in the Roman area.

Recording has been carried out by IDIAP in co-operation with Swiss TELECOMM-PTT. They collected 5,000 speakers who answered several questions (around 10) over the telephone, leading to spontaneous speech, and reading about 28 items from a form supplied by IDIAP.

This form contains several speech sequences, including sentences from different sources (local newspapers, existing corpora, law articles, etc.) to ensure a good phonetic coverage, application words from a defined list of command words, currency amounts, quantities, credit card numbers, spelled words (mainly names), etc. The database is divided into two subsets: the first one comprises 1,000 speakers and the second one 4,000 speakers (1,000 speakers are not available). Each subset is divided into two subsets: the phonetically rich sentences and the application-oriented data.

Some time after Swiss French Polyphone was recorded, some speakers were asked to call back to make new recordings. This is the PolyVar database whose aim it is to capture intra speaker variability.

Tables 3.6 and 3.7 show example phonetically rich sentences and application words respectively.

Table 3.6: Example Swiss French Polyphone sentences

Des bandits de grand chemin ne se gênent plus pour barrer les routes en plein jour.
Le devoir d'ingérence s'impose aussi pour ce pays qu'on veut hermétique.
Cela nous stimulera sûrement à faire le meilleur boulot possible, mais pas dans un esprit de revanche.
Certes, depuis la crise conomique de la fin des années septante, l'émigration s'est ralentie.
La présence de monsieur Gorbatchev est l'élément central du décor.

Table 3.7: Example Swiss French Polyphone application words

Message	Quitter
Muse	Non
Cinma	Concert
Abonnement	Quatre

3.5 Voiceconnect97

The Voiceconnect97 database was recorded internally at KPN research in 1997. It consists of 5300 telephone recordings made of first and last names during the test trials of the Voiceconnect name-dialing service. The service makes it possible to connect to a coworker by simply stating his name (first and last). Everyone in the company was asked to volunteer and use the service during the test trials and fill in questionnaires about its use afterward. [1]

The lexicon size of this database is 535, but with pronunciation variants added the lexicon size is 1335. Each name has at least 2 pronunciation variant: one with silence between first and last name, and one without. Of the 535 names, the major part of the names are from Dutch origin; the number of names of foreign origin is negligible. As the speech in this corpus is spontaneous many utterances have hesitations and mouth noises. Some example names in the database are shown in Table 3.8.

Table 3.8: Example VoiceConnect97 utterances

Aad de Ronde	Victor Tijssen
Jan Koster	Petra Held
Agaath Sluijter	Tineke boogaart
Marcel Vos	Kees van den Bergh

3.6 DDAC2000

The DDAC2000 corpus was recorded in 2000 in the Netherlands during experiments with an automated Directory Assistance task. People all over the country who called were greeted by a welcome message and asked the city of the person or company they would like to reach. Next they were asked the name of the person or company. The conversation is finalized by a confirmation prompt. The complete corpus consists of about 50,000 dialogs. The corpus was transcribed by the Speech Processing Expertise Centre (SPEX). Transcribers were instructed to tag words with an English pronunciation with an “<e>”. Some examples of English utterances pronounced by Dutch people are given in Table 3.9.

Table 3.9: Example DDAC utterances

British Airways	Honeywell
Delta Airlines	Hewlett Packard
Free record shop	World Online
XS4ALL (access for all)	Toys “R“ Us (toys are us)

[1] The current version of Voiceconnect can be reached by telephone by dialing (+31)152857575

Chapter 4

Tools

The various software tools that were used during the course of this work are described here. The features that were used will be dealt with in more detail.

In Table 4 a short summary of the tools used is given.

Tool	Type	Stability	Docs	License	Functions	GUI	Ease
Praat	Speech Analysis	+++	+++	OS	+++	Y	+++
SONIC	Speech Recognizer	++	++	Research	++	N	+
SpeechPearl 2000	Speech Recognizer	+++	+++	Commercial	+++	Y	++
WEKA	Classifier Toolkit	+	+	OS	+++	Y	+++
BoosTexter	Classifier	+	+	Research	+++	N	+

In chapter 5 Praat and SONIC are used. Praat because it is a good free all around tool which provides all analysis functionalities needed. SONIC because it provides the functionality to train our own acoustic models.

In chapter 6 SpeechPearl 2000, WEKA and BoosTexter are used for various reasons. SpeechPearl 2000 is used over SONIC because it has well tested commercial grade acoustic models, is very stable, and in-house software provides good integration with telephony systems for real life data gathering and testing. BoosTexter is used because it is a very well known algorithm based on AdaBoost, reasonably easy to use, and needed for a comparison.

Some other tools were also considered. For example the BOW classification toolkit [McCallum96], and the now open source and actively developed CMU Sphinx speech recognizer [Lee89], but were not selected, because BOW was surpassed in features by the WEKA toolkit and had a very unfriendly format for data files and due to the presence of in house knowledge for Sonic and SpeechPearl 2000.

4.1 The SONIC Large Vocabulary Speech Recognizer

SONIC is a toolkit for enabling research and development of new algorithms for continuous speech recognition. Since March of 2001, SONIC has been used as a test bed for integrating new ideas and for supporting research activities that include speech recognition as core components at the Center for Spoken Language Research. While not a general HMM modeling toolkit, SONIC is specifically designed for speech recognition research with careful attention applied for speed and efficiency needed for real-time use in live applications.

SONIC utilizes state-of-the-art statistical acoustic and language modeling methods. The system acoustic models are decision-tree state-clustered Hidden Markov Models (HMMs) with associated gamma probability density functions to model state-durations. The recognizer implements a two-pass search strategy. The first pass consists of a time-synchronous, beam-pruned Viterbi token-passing search through a lexical prefix tree. Cross-word

acoustic models and trigram or four-gram language models are applied in the first pass of search. During the second pass, the resulting word-lattice is converted into a word-graph. One can dump lattices in Finite State Machine (FSM) format compatible with AT&T tools. Also one can generate n-best lists using the A* algorithm. Finally, word-posterior probabilities can be calculated to provide a measure of word confidence.

The recognizer also incorporates both model-based and feature-based speaker adaptation methods. Model-based adaptation methods include: Maximum Likelihood Linear Regression (MLLR), Structured MAP Linear Regressions (SMAPLR). In addition, SONIC also includes implementation of feature-based adaptation methods such as Vocal Tract Length Normalization (VTLN), cepstral mean & variance normalization, and Constrained MLLR (CMLLR). Finally, advanced language-modeling strategies such as concept language models can be used to improve performance for dialog system recognition tasks.

The SONIC Large Vocabulary Speech Recognizer [Pellom01, Pellom03] is used to perform the experiments in the multilingual acoustic model part of this work. SONIC was initially written Bryan Pellom of the University of Colorado. It is a research recognizer and thus constantly in development. SONIC is a collection of tools which are run from the command line, thus requiring the documentation to be read extensively in order to master it. The documentation is somewhat outdated and more of a survival guide, thus requiring a large amount of "hands-on" experience to master. Another downside is that it has many bugs, and sometimes the programs can crash rather inexplicably. Source code and support can be purchased separately. Elements of the source were modified and binaries recompiled to alleviate bugs and certain limitations in the program (e.g. limitation of 66 phonemes maximum, Windows file name issues, Windows socket code).

The following sections discusses the elements of SONIC that were actually used.

4.1.1 Acoustic model training

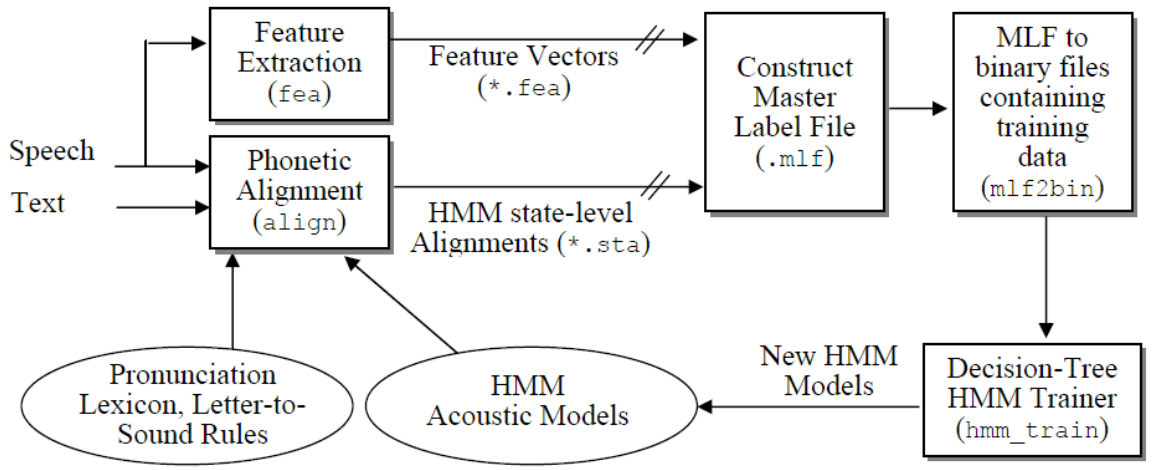


Figure 4.1: Sonic training process

Data Preparation

The acoustic modelling process for SONIC is outlined in Figure 4.1. A lot of audio data is needed to do acoustic model training. The audio files need to be in 16 bit linear PCM audio format in the machine's native byte format. To convert from various formats to the format SONIC requires the "sox" program is used. Acoustic models for the recognition of telephone speech are desired, so for some corpora the audio is bandpass filtered and downsampled to 8 kHz to match the acoustic conditions of a telephone line more closely.

Word-level text transcriptions for each audio file are required. It is also possible to force alignment of phonemes by inserting a '!' in front of it: `!SIL`. Each word in the transcription (separated by white space) needs an entry in the pronunciation lexicon, so it's necessary to remove all punctuation marks from the transcriptions. In the lexicon each word has a phonetic transcription, which tells the speech recognizer how the word is pronounced and what pattern it should look out for during recognition.

If no lexicon is available, letter-to-sound rules can be trained. Using an existing lexicon as training material, it will automatically generate letter-to-sound rules using the ID3 decision tree algorithm. When confronted with words not in the lexicon, a phonetic transcription will be generated using these rules. The textttt2p_fea and textttt2p_train programs can be used for this.

For each audio file the PVMDR feature vectors are extracted with the `fea` command. For every frame of audio, the 39 PMVDR features are extracted.

Alignment

For each audio file and its corresponding text transcription a forced Viterbi alignment is performed. The alignment process attempts to match sequences of PMVDR features to phonemes. Later on during training these aligned phonemes will serve as training input. SONIC is distributed with English monophone, context independent models, which can be used to bootstrap an English recognizer, as well as other languages when a phoneme mapping file is given. The phoneme mapping file specifies for each phoneme of the target language which phoneme of the source language phoneme it is most similar to. This mapping does not need to be a 100% accurate as it only serves to guide the initial alignments. More training iterations can be run to make the final alignment more accurate, although the more accurate the mapping, the less iterations there are necessary. Alignment is done with the `align` tool.

HMM Training

The alignments of all the audio files are combined into a single file. Binary files are then made based on these alignment and extracted audio file features for each phoneme. The binary files are the input for the training process. 3 files are created per phoneme representing each state (0, 1, 2). To create the binary file out of the master label file, the `mlf2bin` program is used.

Training is done with the `hmm_train` program. The training process uses the binary files as input. For training usually full context is selected which will create 12 acoustic models for each phoneme, consisting of 4 sets with each having 3 states. The 4 sets are no-context, left-context, right-context and left-and-right context. All the files for all phonemes are combined to create the final acoustic model using the unix `cat` command. To create context dependent acoustic models, all files are combined into one. To create context independent models, only the no-context files are combined.

Model Refinement

The resulting acoustic model can be refined in several ways. The first method is to adapt the model, for example to new speakers or noise, but this is not discussed here, nor applied in this work. The second method is to redo the alignment process, this time using the new acoustic model as bootstrap, which will hopefully lead to more accurate alignments. The rest of the training steps are the same. The refinement iterations can be repeated many times.

4.1.2 Recognizer configuration

As Figure 4.2 shows, there are two programs in the SONIC package which can perform speech recognition: `sonic_batch` and `sonic_server`. As the name says, `sonic_batch` does recognition in batches, and is capable of recognizing thousands of audio files one by one, and is most useful when one wants to perform tests. `sonic_server` is used to perform realtime recognition or dictation. It communicates with `sonicclient` over a TCP/IP port which in turn takes microphone input.

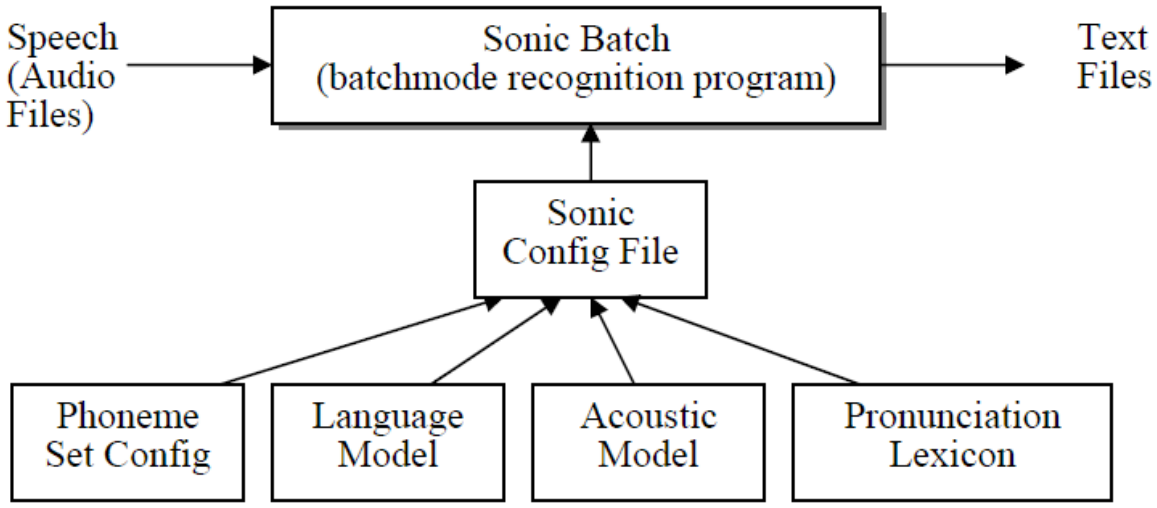


Figure 4.2: Sonic recognition process

Both recognition programs require a language model or a grammar specification in order to do recognition. The CMU Statistical Language Modeling Toolkit can be used to create a language model from training transcriptions. For evaluating recognition result the NIST `sclite` package can be used to calculate recognition rate statistics such as sentence error rate and word error rate.

4.2 Praat

Praat is a multifunctional tool for doing phonetics written by Paul Boersma and David Weenink of the Institute of Phonetic Science of the University of Amsterdam [Boersma01, Boersma05]. It provides features for the speech analysis, synthesis and manipulation, listening experiments, labeling and segmentation, as well as the ability to collect statistics and output in high quality graphics. Furthermore it is open source, compatible with many file formats and has good documentation and tutorials for using most functions. The most used feature during the course of this work was the spectrogram analysis function. The software appears stable and bug free as far as the limited amount of features that were used.

A screenshot of the interface used when doing analysis of wave files is shown in Figure 4.3, giving two graphical representations of a sound sample. The top representation shows the amplitude (Y-Axis) of the sound over time (X-Axis). The bottom representation shows a spectrogram of the sound. Again, the X-axis represents time, and the Y-axis represents increasing frequencies. Darker areas represent areas where a frequency is more prevalent. These dark areas can form lines, which are called formants. These dark lines are numbered from bottom to top F_0, F_1 etc. starting from the lowest frequency to highest. Most vowels can be characterized by the first 2 formants.

Praat has a feature to automatically detect formants, and these are graphically depicted in the spectrogram view with red dots. It can also determine the pitch and intensity at a certain time. Furthermore it can be used to annotate

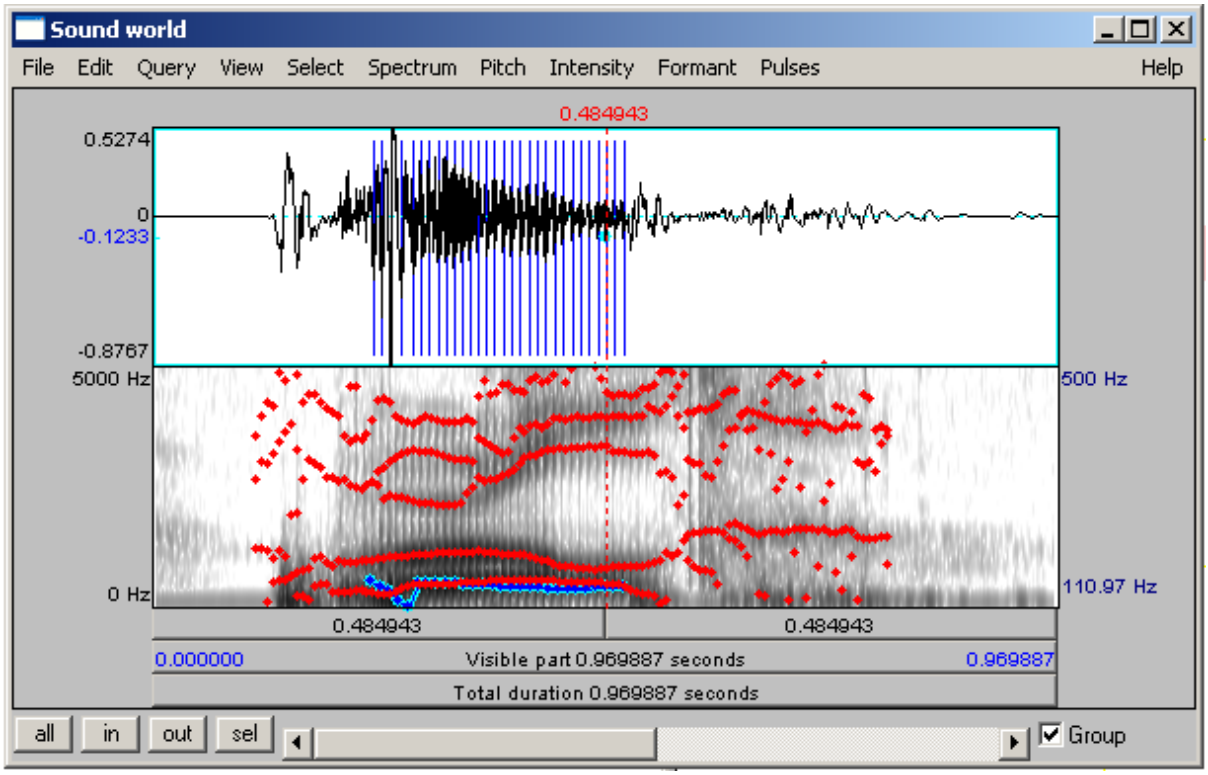


Figure 4.3: Screenshot of Praat's sound editor

and label portions of the file with the corresponding word or phoneme using mouse selection and listen to to each sub-selection over and over.

4.3 Philips SpeechPearl 2000

Philips Speechpearl 2000 is a commercial package which makes it possible to program speech recognition in custom applications. It is certainly not a new product having its roots in an older product called SpeechMania, which was also developed by Philips Research, but even though the product is not actively marketed anymore, recent updates have still been released by Scansoft, the current intellectual property rights owner of the SpeechPearl product line. Speechpearl 2000 provides access to functions in the C programing language, to provide high performance low level integration into applications such as telephony. Several acoustic models are provided with the package, offering some trade-off between recognition performance and speed, but a training tool is not commercially available.

It features several applications to aid in the development of "speechblocks". Each speechblock consists of a lexicon, grammar, acoustic model and language model. A speechblock generally only recognizes one part of a dialog and can be made by constructing a grammar. Speechpearl2000 supports both open and closed grammars. Closed grammars allow developers to specify which utterances should be recognized in a notation similar to BNF. This is most suited for short utterances or instances where the utterances are very predictable. In all other cases open grammars are more suitable to detect key phrases. After the grammar is specified, the word list or lexicon can be extracted from it, and a basic language model is also compiled automatically.

This basic language model can be improved by training it on real spoken utterances. This is done as follows. Whenever SpeechPearl is instructed to recognize an utterance, it can log what it has recognized to disk, as well as the actual audio. When a number of these utterances is recorded, they form a corpus. By using the included

annotation tool, these automatic annotations can be corrected manually. A large number of utterances will provide enough statistics to improve the language model, and will lead to extra entries in the lexicon, as there are usually words which people use which the developers did not think of at first. The retraining of the speechblock is done with the included language model training tool. This process can be repeated until the performance is satisfying.

Although somewhat old, it has proven to be a reliable and stable platform to build applications which are not too demanding. However due to it being commercial it provides limited insight in the inner workings of the recognizer and provides limited options for tweaking.

4.4 WEKA - JRIP

An exciting and potentially far-reaching development in computer science is the invention and application of methods of machine learning. These enable a computer program to automatically analyze a large body of data and decide what information is most relevant. This crystallized information can then be used to automatically make predictions or to help people make decisions faster and more accurately.

WEKA [Witten05] is a collection of machine learning algorithms for data mining tasks. The overall goal of the WEKA project is to build a state-of-the-art facility for developing machine learning (ML) techniques and to apply them to real-world data mining problems. The development team has incorporated several standard ML techniques into a software ”workbench” called WEKA, for Waikato Environment for Knowledge Analysis. With it, a specialist in a particular field is able to use ML to derive useful knowledge from databases that are far too large to be analyzed by hand. WEKA’s users are ML researchers and industrial scientists, but it is also widely used for teaching.

The algorithms can either be applied directly to a dataset or called from your own Java code. WEKA contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes. WEKA is open source software issued under the GNU General Public License.

In general using WEKA to train a classifier is as follows. In the “preprocess” tab (Fig 4.4), open the file containing the training data. WEKA uses its own format, and has a few import filters, but generally its best to convert the data to the WEKA format. After it is loaded, it is possible to perform some preprocessing operations. One of the most useful is to convert a string to a word vector, so one field containing a string will turn into many fields representing a word and the value of 1 or 0 depending on whether the word was present in the string or not. After preprocessing the field containing the class must be selected. After this is done some statistics are made, like the number of items per class.

In the “classify” tab (Fig 4.5 and 4.6) the desired algorithm must be chosen to train a classifier. There are many to choose from, and each has its own set of options and parameters. Classifiers are roughly sorted by type, such as numerical (neural), Bayesian, decision-tree based etc. After an algorithm is chosen one can train a classifier. This can take a while, and sometimes WEKA will stop saying not enough memory. After the classifier finishes training, it can be tested on a separate test set, on the training set, or cross validation can be done. After each test a detailed report is shown with statistics. The classifier can be saved and reloaded for future use.

WEKA is a very useful tool for testing a large amount of classifiers on the same data, and custom classifiers can be plugged in easily. The standard package comes with a lot of classifiers already, as well as references to the papers in which their workings are described. Because WEKA is written in JAVA there are problems handling large datasets (or huge amounts of attributes), and it will often fail in the middle of training of classifier, even when increasing the maximum amount of memory it is allowed to use. Nevertheless it remains a useful tool for rapidly testing many classifiers using the same data set. The WEKA classifier used most in this thesis, JRIP, is an implementation of the RIPPER algorithm which is described in Section 6.3.1.

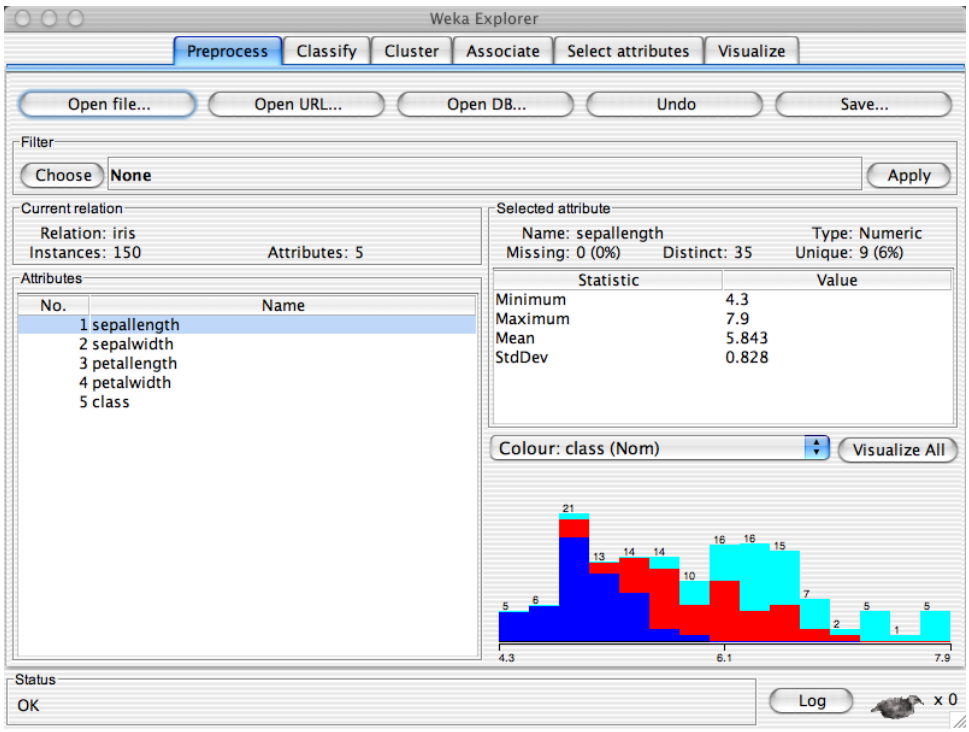


Figure 4.4: Weka Preprocess Panel screenshot (Mac)

4.5 BoosTexter

AT&T has done extensive research developing and using BoosTexter for call classification tasks, and BoosTexter can still be regarded as state of the art. BoosTexter [Schapire00] is a general purpose machine-learning program based on boosting for building a classifier from text and/or attribute-value data. BoosTexter can handle:

- multiple attributes which may be textual, discrete or continuous;
- data with missing attributes;
- multiclass problems, including problems in which some instances belong to more than one class;
- fairly large datasets (up to around 100,000 examples or more, depending on the computer and the form of the data).

BoosTexter uses boosting on top of very simple decision rules (sometimes called “decision stumps”). Although this allows BoosTexter to run very fast while often giving highly accurate results, this approach may not be appropriate for all learning tasks. For instance, boosting on top of decision trees (such as C4.5 or CART) may be more effective for some applications. BoosTexter is based on the AdaBoost [Freund97] algorithm and is described in more detail in Sectino 6.3.2

BoosTexter is a command line utility and has a very short text file explaining the usage. Not all options are explained thoroughly. The source code is not available. BoosTexter uses its own file format, and has a few (limited) options, which make it possible to create decision stubs on single words, bigrams or trigrams. Training is done by giving the data file as input. Testing is done in a similar fashion. Not many statistics are output by the program.

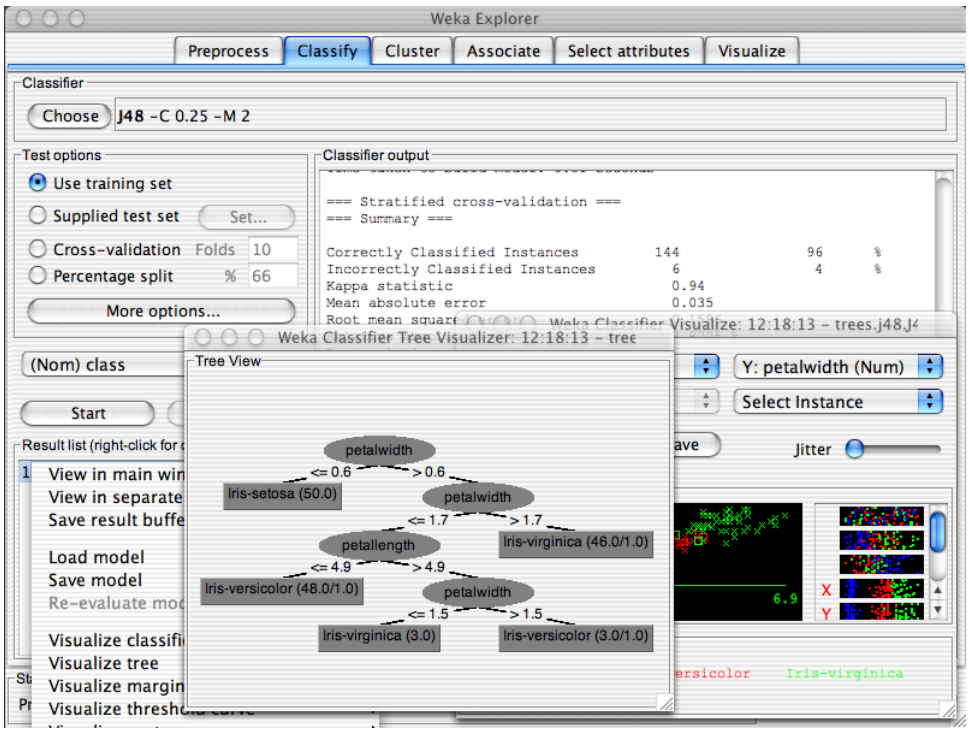


Figure 4.5: Weka Classify Panel screenshot (Mac)

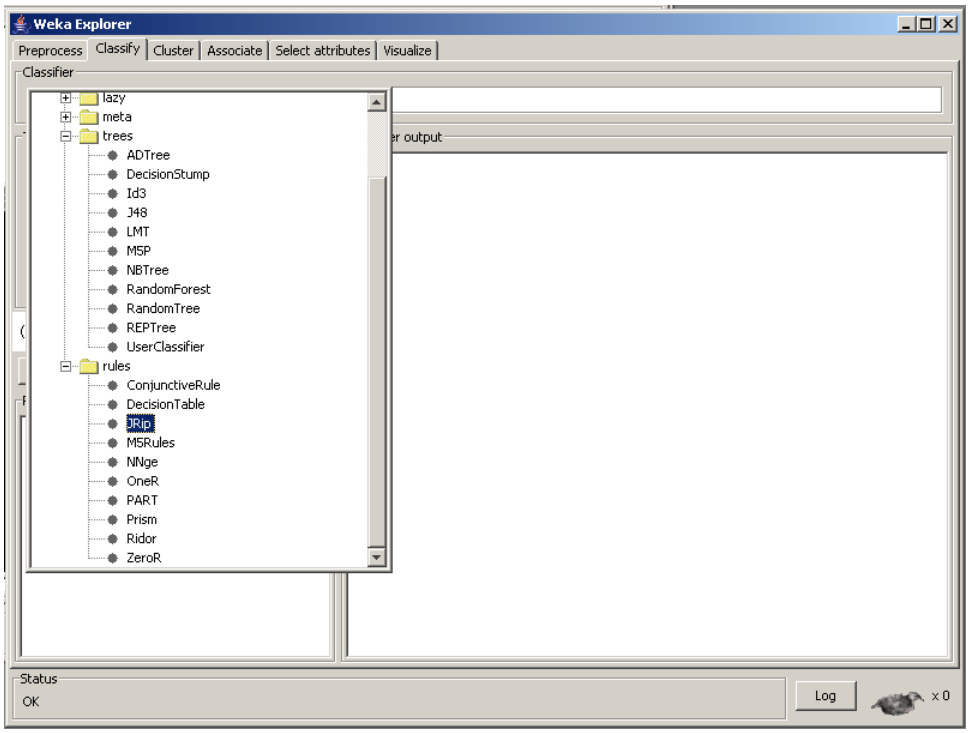


Figure 4.6: Weka Classify Panel screenshot (Windows)

Chapter 5

English foreign word recognition

5.1 Introduction

When people of a foreign country try to speak the local language at least some will have problems with the pronunciation. The reverse is also true. When people try to speak a foreign language, some will have the correct pronunciation but some will have trouble with the pronunciation. For humans, even though pronunciation errors are made, these kinds of utterances can still be understandable, sometimes with the help of the context of the situation where such an utterance is made.

Of particular interest is the pronunciation of English by Dutch in the Netherlands. This is because over the years a lot of popular English words have made it into the vocabulary of the average inhabitant of the Netherlands and are sometimes preferred over Dutch equivalents (if they exist at all), driven by the globalization of organizations, and media such as internet, television and radio.

The typical pronunciation of a Dutch person speaking English will not necessarily be bad. Foreign phonemes originally not belonging to Dutch are sometimes taken over flawlessly. For example the “g” in “goal”. But take for example the sentence “I will put the cup on the table”. It is not uncommon to hear the English word “put” pronounced as the Dutch word “poet” and “cup” be pronounced as “kop” or “kup” (using Dutch pronunciation rules). This phenomenon occurs because they are unable to pronounce the phoneme correctly and instead they will use the closest phoneme that sounds like it or use their own pronunciation rules.

The typical speech recognizer is heavily trained on phonemes typical for one language, and is only lightly trained on foreign ones, if at all. For example “goal” with the hard “g” is now a common word in the Dutch language, but the “g” is of such low frequency in the language that is hard to justify creating a separate model for it, and instead the phoneme will be merged with another one. For recognition tasks in Dutch only sentences they perform well because they are trained on it. The occasional foreign word can be recognized by mapping the foreign phoneme to the closest native one.

However in some speech recognition applications the use of foreign words (and thus foreign phonemes) can be expected with very high probability and in those applications the recognition of such a foreign word can be essential in addition to the recognition of Dutch.

Examples of such applications:

- Applications which accepts unrestricted speech: a telephone ordering service: “Ik wil graag de nieuwste single van Britney Spears bestellen die ik gisteren op MTV gezien heb”, or an internet helpdesk: “Ik heb problemen met Outlook Express en kan geen e-mail ontvangen. Ik kreeg deze error message ...”.

- A name dialing application in an international office or directory service which accepts both Dutch and

English names: “Bill Gates”, “Free record shop”.

For such applications a monolingually trained Dutch or English speech recognizer (even though common practice) may not be adequate enough. A Dutch speech recognizer will have a hard time detecting some English words, because it is not trained on some English phonemes. An English speech recognizer can also not be used because it will fail to recognize some Dutch phonemes.

In summary recognition of English foreign words spoken by Dutch has a few problems:

- The generally accepted English pronunciation requires the use of phonemes which do not occur in generally accepted Dutch.
- As a result Dutch might speak English with an accent and will use pronunciations different from the generally accepted ones: they might try to reproduce English phonemes by using the closest Dutch phoneme, or they might be attempting to pronounce English using Dutch pronunciation rules, and many more possibilities.
- Dutch language corpora are not collected often and over the years the importance and presence of foreign words has increased significantly. The corpora remain a good source of training material for the typical Dutch phonemes but are inadequate to train a speech recognizer which can detect English foreign words.

A solution would be to have a Dutch recognizer additionally trained on English data, so that it hopefully will recognize Dutch and Dutch speaking English. One technique to do this is multilingual acoustic modeling which takes training data from multiple languages and combines the phoneme inventory of each language into one multilingual set. Rather than mapping foreign phonemes to the closest one and use a monolingual recognizer, the foreign phonemes are added to the native creating an expanded multilingual model.

Multilingual acoustic modeling was chosen for the following reasons:

- Instead of mapping English phonemes to Dutch, the Dutch phoneme set is expanded with the missing English phonemes. The word lexicons can therefore be more accurate [Gustafson95], hopefully resulting in better recognition.
- They are known to be able to recognize multiple languages with one recognizer instance and recognition of Dutch and the occasional English is desired. The use of one recognizer instance is also desirable for reduction of costs.
- Multilingual models are known to achieve better cross-lingual recognition performance compared to using a monolingual recognizer when there is little data available to bootstrap the recognizer. There is no suitable training data available of Dutch people speaking English, so building a recognizer with that as basis is out of the question and neither is adaptation of a Dutch recognizer. However there is a lot of Dutch speech data available, and also some English and French data. This can analogous to a bootstrapping a new language if “English spoken by Dutch” is regarded as a new language.

This chapter investigates the possibility of using multilingual modeling for recognition and the problem definition is as follows: Does using multilingual models improve recognition performance of English utterances spoken by Dutch, and why or why doesn’t recognition improve?

This will be answered by answering the following subquestions:

1. How is the performance of multilingual models on Dutch?
   The performance for Dutch must not degrade too much compared to a Dutch only recognizer. The VoiceConnect97 corpus will be used for testing this.
2. How is the performance of multilingual models on native English?
   Not strictly required, but is useful to know if native English can also be recognized. The TIMIT corpus will be used for this.

3. How is the performance of multilingual models on English utterances spoken by Dutch?
   The most interesting test. Does performance in fact improve compared to a Dutch only recognizer, and if so how much, under what circumstances and why? The DDAC2000 corpus will be used for this.

5.2 Training multilingual acoustic models

This section describes how the acoustic models were trained. Recognition of both Dutch and English phonemes is required, so it is reasonable to use Dutch and English training data. Swiss French data is also added to the mix, because [Kunzmann04] shows that adding data even not belonging to the languages to be tested might improve performance, as it simply provides more training samples and allophonic variations for the phonemes the languages have in common. So in fact 3 languages were used for training.

As the corpora came from different sources different transcription guidelines were used. To unify them phoneme sets were converted to IPA, sometimes simplified to make the granularity the same, and converted back into a machine readable form, X-SAMPA [Wells97b]. Statistics regarding the used training corpora can be found in Table 3.1 and Table B.1. In this text phonemes are generally described using X-SAMPA symbols, not IPA ones, and are surrounded by slashes, for example /p/. This section gives an overview of all acoustic models trained (Table 5.1) and describes how they were trained.

Table 5.1: Acoustic model overview

Code	Trained on	Phonemes	Alignments	Comments
AMO_NL1	Dutch	45	1	
AMO_NL2	Dutch	45	2	baseline acoustic model
AMO_NLEN1a	Dutch, English	66	1	Using Eng. word trans.
AMO_NLEN1b	Dutch, English	66	1	Using Eng. phoneme trans.
AMO_NLEN2	Dutch, English	66	2	
AMO_NLEN2q	Dutch, English	66	2	language question
AMO_NLENFR1	Dutch, English, French	78	1	
AMO_NLENFR2	Dutch, English, French	78	2	
AMO_NLFR2	Dutch, French	78	2	Eng. phonemes not trained

5.2.1 Iterative training procedure

Acoustic models were trained initially according to the Lang-mix acoustic modeling strategy, meaning data is shared for phonemes with the same IPA symbol. Lang-sep was not considered because the desire is to expand the phoneme set, not recognize two languages separately. Some tests with the Lang-tag strategy were also done in a later stage.

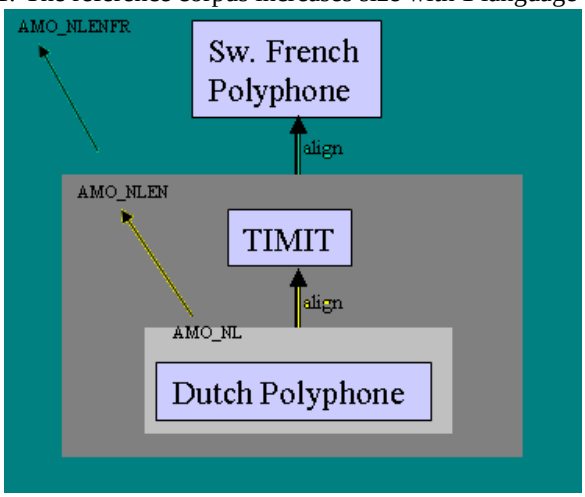
Since much of the training data did not come from the same sources, did not have time aligned phonetic transcriptions but only word-level transcriptions, and a relatively large amount of data was available for Dutch and relatively little for the other training languages, an iterative approach to aligning the training data was chosen, similar to [Walker03].

The iterative procedure ensures that alignments are as consistent as possible across corpora and is as follows (also see Figure 5.1:

1. Initialize the multilingual set with a reference corpus C_0 and train a recognizer R_0. Initialize $x = 0$.
2. C_{new} is the new corpus not yet in the reference corpus.
3. Use recognizer R_x to align the data C_{new}. If C_{new} has phonemes which are not in the reference corpus, the aligner is configured to initialize this phoneme with the closest phoneme in C_x.

4. Expand the reference corpus C_x with C_{new}, creating C_{x+1}.
5. Train a new recognizer R_{x+1} on the new reference corpus C_{x+1}. If phonemes from the existing set and new corpus have the same phonetic symbol, they will share the same acoustic model.
6. Increase x by 1.
7. If there are still corpora not in the reference set go to step 2.
8. Else the procedure is finished.

Figure 5.1: The reference corpus increases size with 1 language each time



5.2.2 Training Dutch

The Dutch training data was chosen as the reference corpus, since the performance of the model trained with this data would be used for baseline testing.[1]

The Dutch training data is taken from Dutch Polyphone [denOs95], but is different from the standard training set. The used training set consists of short utterances only and this training set has been shown to perform better on short utterances, but worse on longer utterances [Sturm00]. The Dutch lexicon was transcribed with the PHICOS phoneme set, an extension of SAMPA [Wells97a] with postvocalic /L/ and /R/. PHICOS to IPA mappings were taken from [SpeechPearlManual] and [Wester02]. The complete Dutch phoneme set can be found in Table 5.2.

The Dutch training data was aligned with text transcriptions by bootstrapping from the very basic English monophone models provided with SONIC. Training from this data results in the first context dependent Dutch acoustic model, AMO_NL1. The data was then realigned and retrained using this model, resulting in AMO_NL2. The alignments for Dutch were not changed anymore after this point. The phonetic set at this point consisted of 45 symbols, including 3 non-speech (silence, background noise, mouth noises).

5.2.3 Training and adding English

The English training data consists of the TIMIT [Garofolo93] training set. The TIMIT data consists of phonetically rich sentences. The audio was filtered with a low-pass of 3.4 kHz and a high-pass of 0.3 kHz and down-sampled

[1] Also for the practical reason that it was the only corpus available at the time. The other corpora became available in the weeks following the decision and were still unknown at the time.

Table 5.2: Phonemes in the Dutch Polyphone corpus

IPA	X-SAMPA	Example
p	p	pak
b	b	bak
t	t	tak
d	d	dak
k	k	kap
g	g	goal
f	f	fel
v	v	vel
s	s	sein
z	z	zijn
x	x	toch
h	h	hand
ʃ	S	show
ɣ	G	goed

IPA	X-SAMPA	Example
ʒ	Z	bagage
m	m	met
n	n	net
ŋ	N	bang
l	l	land
ɬ	5	hal
r	r	rand
ɹ	r\	tor
w	w	wit
j	j	ja
ɪ	I	pit
ɛ	E	pet
ɑ	A	pat
ɔ	O	pot

IPA	X-SAMPA	Example
ʏ	Y	put
ə	@	gemak
i	i	vier
y	y	vuur
u	u	voer
aː	a:	naam
eː	e:	veer
øː	2:	deur
oː	o:	voor
ɛi	Ei	fijn
œy	9y	huis
ɑu	Au	goud
ɛː	E:	créme
ɔː	O:	roze

to 8 kHz to simulate telephone speech as much as possible, as TIMIT was originally recorded with high quality microphones in a quiet environment. The lexicon provided with TIMIT was converted to IPA using the table found in [Keating98]; some symbols additionally had lengthening marks added.

After adding English phonemes to the Dutch set, the multilingual phoneme set was expanded 66 phonemes including 3 non speech symbols. The new phonemes were mainly vowels and diphthongs, but some affricates and fricatives as well. A full listing of the English phonemes merged in the multilingual set can be found in Table 5.3.

Table 5.3: Phonemes in the English TIMIT corpus

IPA	X-SAMPA	Example
p	p	pin
b	b	bin
t	t	tin
d	d	din
k	k	kin
g	g	give
tʃ	tS	chin
dʒ	dZ	gin
f	f	fin
v	v	vim
θ	T	thin
ð	D	this
s	s	sin
z	z	zing
ʃ	S	shin

IPA	X-SAMPA	Example
ʒ	Z	measure
h	h	hit
m	m	mock
n	n	knock
ŋ	N	thing
ɹ	r\	wrong
l	l	long
w	w	wasp
j	j	yacht
m̩	m=	bottle
n̩	l=	button
l̩	l=	bottom
ɪ	I	pit
ɛ	E	pet
æ	{	pat

IPA	X-SAMPA	Example
ɑː	A:	pot
ʌ	V	cut
ʊ	U	put
ɨ	1	debit
iː	i:	ease
eɪ	eI	raise
uː	u:	luse
oʊ	oU	nose
ɔ	O	cause
aɪ	aI	rise
ɔɪ	OI	noise
aʊ	aU	rouse
ɝ	3`	furs
ə	@	allow
ɚ	@`	corner

The TIMIT corpus was the only corpus with detailed phonetic transcriptions, which gave time precise phonetic realization of the whole sentence, in addition to the word-level transcriptions (converted to phoneme level transcriptions using a lexicon). The two transcriptions did not always correspond, therefore two context dependent models were trained to see which one was better: AMO_NLEN1a using word level text transcriptions and AMO_NLEN1b using exact timed phonetic transcriptions. The Dutch model (AMO_NL2) was used to align the English data in

both cases.

Initial testing showed that AMO_NLEN1a performed better (see Section 5.3) than AMO_NLEN1b, probably because the phoneme models were not trained on for example allophonic variation. Because of this AMO_NLEN1a was used to realign and retrain the data, resulting in the context dependent model AMO_NLEN2. English alignments were then fixed.

Finally, a language question was added to the decision tree question list to train a different model for Dutch and English according to the Lang-tag strategy, resulting in AMO_NLEN2q. Also a model with the TIMIT data only was trained, AMO_EN, using the alignments from AMO_NLEN2.

5.2.4 Training and adding Swiss French

The Swiss French data consists of a subset of the phonetically rich sentences from Swiss French Polyphone [Chollet96]. No lexicon was readily available for this corpus, so letter to sound rules were trained using the SONIC text to phone tool. The BRULEX lexicon [Content90] was used as training material for the tool and has over 35000 lexicon entries. It was not determined how accurate the generated transcriptions are, but viewing the transcriptions empirically did not reveal any errors of significance. All the French phonemes added to the multilingual set are listed in Table 5.4.

A same approach was was taken for adding French to the reference corpus as for adding English. The Dutch-English model, AMO_NLEN2, was used to initially align the French data. The resulting model after training was used to realign the data again and train a new acoustic model. This resulted in the models AMO_NLENFR1 and AMO_NLENFR2.

An acoustic model with only Dutch and French speech data was trained as well (AMO_NLFR2). No realignments were done with the Dutch only model, instead the alignments used to train AMO_NLENFR2 were taken.

Table 5.4: Phonemes in the Swiss French Polyphone corpus

IPA	X-SAMPA	Example
p	p	pont
b	b	bon
t	t	temps
d	d	dans
k	k	quand
g	g	gant
f	f	femme
v	v	vent
s	s	sans
z	z	zone
ʃ	S	champ
ʒ	Z	gens
j	j	pierre

IPA	X-SAMPA	Example
m	m	mont
n	n	nom
ŋ	N	camping
ɲ	J	oignon
l	l	long
ʁ	R	rond
w	w	coin
ɥ	H	juin
i	i	si
e	e	ses
ɛ	E	seize
a	a	patte
ɑ	A	pâte

IPA	X-SAMPA	Example
ɔ	O	comme
o	o	gros
u	u	doux
y	y	du
ø	2	deux
œ	9	neuf
ə	@	justement
ɛ̃	E~	vin
ɑ̃	A~	vent
ɔ̃	O~	bon
œ̃	9~	brun

5.3 Tests

5.3.1 VoiceConnec97 test

Purpose

This test measures the performance of the various models on Dutch only. As it is known in literature, the monolingual recognition performance tends to drop compared to the multilingual model. Since the recognizer must recognize Dutch in addition to the English words, it is essential to measure the model's performance on Dutch.

Setup

The VoiceConnect97 test set consists of 5300 short utterances of Dutch first and last names and a lexicon of 535 names, each with at least one pronunciation variant (silence between words and not). In each utterance the speaker says the name of the person he or she wants to reach by telephone. In some utterances it is normal to have instances of people hesitating ("uh"), making mouth noises etc. These are handled by adding entries in the "filler" lexicon which will prevent these from cluttering the recognition results. Allowed utterances were sets of first and last name. Basically single word recognition was done, as first and last name were entered as a single word in the lexicon. All names have an equal chance of occurring in the unigram language model.

Results

The results of the performance of the acoustic models are in Table 5.5. The performance is measured by the ratio of correctly recognized utterances when processing all the 5300 test utterances. Because single word recognition was done, the sentence error and word error rate are the same. The percentage of correctly recognized utterances can be found in the table under "Correct".

Table 5.5: Voiceconnect97 results

Model	Correct
AMO_NL1	93.26%
AMO_NL2	93.36%
AMO_NLEN1a	92.98%
AMO_NLEN1b	92.70%
AMO_NLEN2	92.66%
AMO_NLEN2q	92.72%
AMO_NLENFR1	91.19%
AMO_NLENFR2	91.34%
AMO_NLFR2	91.02%

Discussion

- As expected the second models, resulting after realignment (with a "2" in the acoustic model name), usually score better, but not significantly, than the corresponding first models. This is because realignment usually results in more accurate alignments which in turn will yield more accurate acoustic models giving better scores. The second models also scored better in the TIMIT test (section 5.3.2).
- The performance degrades when more languages are added to the mix, thus making the best model for this test AMO_NL2 which is only trained on Dutch data, followed by the combination of English and Dutch (AMO_NLEN2). This is also not strange as this is similar to the results in literature.

- Although adding more languages degrades performance, the performance of the best model with English phonemes (AMO_NL2) remains acceptable compared to the baseline, as the degradation is fairly small. The 3 language combination (AMO_NLENFR2) performs worse but not unacceptable.
- Adding French seems to drop performance the most. This is indicated by the fact that the Dutch-French (AMO_NLFR2) model performs worse than all other models, including the Dutch- English-French combination. This is strange because the first has only 2 languages in the mix, and the latter 3.
- Adding the language question (AMO_NLEN2q) improved the recognition performance of the combined Dutch English acoustic model (AMO_NLEN2) only marginally. Lang-tag in literature improves performance of monolingual recognition so this is also expected.

5.3.2 TIMIT test

Purpose

The TIMIT test set was used to evaluate the native English performance of the acoustic model. If every Dutchman happened to have a perfect (American) English pronunciation, this test would be suitable, but there are some flaws. Because the pronunciation of a non-native speaker can be very off or very good, the results of this experiment are not directly relevant to foreign English word recognition, but it might provide some useful information, since the extra English phonemes are heavily tested. Also a comparison is made to the baseline English recognizer, trained on English data only, to see how much performance drops with Dutch data added.

Setup

The TIMIT test set, like the TIMIT training set, consists of phonetically rich sentences instead of short utterances. There is no overlap of training and test set with regard to utterances and speakers. Because whole sentences had to be recognized, a trigram language model was trained using the list of all the utterances in both the test and train set. For testing, the original TIMIT test set is used, which has no overlap in speakers or utterances with the original TIMIT training set. The original TIMIT training set was also used for training the multilingual models.

Results

The results for the different acoustic models are in Table 5.6. Because long sentences are recognized, there are also additional measures to determine the performance. “Word Correct” is the word accuracy ratio, meaning the amount of words recognized correctly, independent of the sentence. “Sub”, “Del” and “Ins” are more specific and represent respectively how many times a word was recognized wrongly, not at all, or an extra word was inserted in the recognition results. Finally “SER” or Sentence error ratio determines how many sentences were recognized wrong, thus having at least one word error.

Table 5.6: TIMIT test results

Model	Word Correct	Sub	Del	Ins	Err	SER
AMO_EN	91.8%	6.3%	1.9%	2.4%	10.6%	41.0%
AMO_NLEN1a	91.0%	6.9%	2.1%	3.4%	12.4%	46.4%
AMO_NLEN1b	88.9%	8.7%	2.4%	3.9%	15.0%	51.7%
AMO_NLEN2	91.1%	7.0%	2.0%	3.2%	12.1%	45.8%
AMO_NLEN2q	90.8%	7.2%	2.0%	3.4%	12.6%	47.3%
AMO_NLENFR1	90.8%	6.9%	2.2%	3.0%	12.1%	44.8%
AMO_NLENFR2	91.2%	7.0%	1.8%	3.5%	12.3%	46.6%

Discussion

- The best model in this test is the model trained on English data only, the monolingually trained AMO_EN. This is expected because adding additional languages decreases performance.
- Again the mixed English-Dutch models, but especially AMO_NLEN2, does not perform significantly less than the English only recognizer. This indicates that the English phonemes are recognized well.
- The Dutch-English model trained with the phonetic transcriptions performed worse than the model trained with word level transcriptions (AMO_NLEN1b versus AMO_NLEN1a). AMO_NLEN1b was trained using the phonetic transcriptions transcribed from the audio, while AMO_NLEN1a was trained using word transcriptions which were transformed into phonetic transcriptions using the lexicon. It is a known fact that words do not always sound the same, for example if spoken fast, a vowel might be omitted or co-articulation might occur. In the case of AMO_NLEN1b a phoneme was trained if and only if it was observed by the transcriber. In the case of AMO_NLEN1a a phoneme is always trained if the lexicon suggests a word is pronounced in a certain way, therefore making AMO_NLEN1a more robust to variations in pronunciations. AMO_NLEN1a was used to realign the training data for AMO_NLEN2 and this model performs better than both.
- The Lang-tag strategy appears not to increase performance of the monolingual English recognition, and performance is in fact less. This might be explained by the fact that there was a lot more Dutch training data used compared to the amount of English data, therefore resulting in a higher bias toward recognition of Dutch phonemes.
- Adding French to the multilingual model did not seem to have a conclusive effect, compared to the best English Dutch model. The word error rate slightly decreases, but the sentence error slightly increases.

5.3.3 DDAC2000 test

Purpose

This test set is the most important one, since it most closely resembles the goals of the experiment: to see if recognition of English foreign words improves with a multilingual model.

Setup

A subset of the DDAC2000 [Sturm00] corpus was used for this test. The transcription protocol of this corpus specifies that words pronounced as English are tagged with an “<e>”. Only these tagged words were considered for inclusion in the test set, resulting in a test set of around 3600 utterances. This set was further reduced to utterances which occurred two or more times, in order to limit the time to write and verify the lexicon. The result was a final test set of 1018 audio files, and a vocabulary of 272 items of which a complete utterance is regarded as an item.

The items in this lexicon were often a combination of Dutch and English words, but always at least one English word, therefore the phonetic transcriptions were allowed to have a mix of both Dutch and English phonemes, with English phonemes used whenever a word was English. The initial lexicon was generated using letter to sound rules trained on the American English TIMIT lexicon. This obviously produced a lot of junk when generating transcriptions for Dutch words, so all entries were double checked and fixed manually.

When initially testing this lexicon on AMO_NLEN the result was a correct ratio of only around 77%. By listening to what was actually said when errors occurred, pronunciation variants were added to the lexicon. Corrections were only made when what was actually said was significantly different from what was expected. Several types of errors and fixes are listed here:

- Left-over transcription errors: mostly in the manual phonetic transcriptions. Additional scans for errors were made to correct it.
- Words having both a native English and Dutch pronunciation such as “Holland” in “World Travel Holland” or prefixes such as “Euro-” and Dutch words from English origin: Although the words tagged with were “<e>” reserved for words pronounced as English, they were often also mistakenly used for Dutch words of English origin. In all these cases, the Dutch pronunciation variant was added.
- Different British and American pronunciations, for example “master”: The British pronunciation was added to the lexicon, in addition to the American.
- Common phoneme substitutions, for example /Y/ instead of /U/ in “jungle”: This error occurs because the /U/ is not a Dutch phoneme, but people pronounce it as a Dutch /Y/ based on how the word is written.
- Long silences between words: an additional entry in the lexicon was made with the “SIL” phoneme between words.

After applying fixes, the resulting lexicon is called LEX_EN. This lexicon represents the phoneme expansion approach. The LEX_EN lexicon was then converted to one with Dutch phonemes only using the mapping in Table 5.7, resulting in LEX_NL and represents the phoneme mapping approach. The mapping table was constructed such that each English phoneme not present in the Dutch set should have a representation using the Dutch set which matches as close as possible. One way to come up with such a mapping, and which was applied here was by asking the following question: What combination of Dutch phonemes is necessary to produce an acceptable pronunciation for an English word using Dutch text to speech?

Table 5.7: English to Dutch Phoneme mapping table

English	Dutch	English	Dutch	English	Dutch	English	Dutch
1	I	3`	Y r	@`	@ r\	A:	A
D	d	OI	O j	T	t	U	u
V	O	aI	A j	aU	Au	dZ	d Z
eI	e:	i:	i	l=	@ l	m=	@ m
n=	@ n	oU	o:	tS	t S	u:	u
{	E						

More preliminary tests on AMO_NLEN showed that LEX_NL outperformed LEX_EN. This means that the recognition of Dutch phonemes only (LEX_NL) works better than a Dutch phoneme set expanded with English ones (LEX_NL). However, an error analysis showed that different errors were made in both cases. This suggests that people often use a Dutch-like pronunciation which is recognized well by the LEX_NL lexicon, and some an English-like pronunciation which is recognized well by LEX_EN. To get the best of both worlds, the two lexicons were merged together resulting in LEX_NLEN, which indeed resulted in another performance improvement. This was repeated for all acoustic models which also show this behavior. An overview of all the variations of the lexicon is shown in Table 5.8

Table 5.8: Lexicon names

Lexicon Name	Description
LEX_EN	Lexicon using English phonemes where applicable
LEX_NL	Lexicon using Dutch phonemes only by converting LEX_EN with mapping table
LEX_NLEN	LEX_EN and LEX_NL merged

Results

The complete results are shown in Table 5.9 and 5.10 which show the same results, but in a different ordering to make the results easier to interpret. One has the results ordered by acoustic model with lexicon constant, the other

has the acoustic model constant with different lexicons.

Table 5.9: DDAC2000 results ordered by acoustic model

Acoustic model	Lexicon	Correct
AMO_NL2	LEX_NL	91.45%
AMO_NLEN2	LEX_EN	90.96%
AMO_NLEN2	LEX_NL	92.24%
AMO_NLEN2	LEX_NLEN	93.81%
AMO_NLEN2q	LEX_EN	84.87%
AMO_NLEN2q	LEX_NL	92.24%
AMO_NLEN2q	LEX_NLEN	93.71%
AMO_NLENFR2	LEX_EN	87.52%
AMO_NLENFR2	LEX_NL	92.17%
AMO_NLENFR2	LEX_NLEN	93.03%
AMO_NLFR2	LEX_NL	89.88%

Table 5.10: DDAC2000 results ordered by lexicon

Acoustic model	Lexicon	Correct
AMO_NLEN2	LEX_EN	90.96%
AMO_NLEN2q	LEX_EN	84.87%
AMO_NLENFR2	LEX_EN	87.52%
AMO_NL2	LEX_NL	91.45%
AMO_NLEN2	LEX_NL	92.24%
AMO_NLEN2q	LEX_NL	92.24%
AMO_NLENFR2	LEX_NL	92.17%
AMO_NLFR2	LEX_NL	89.88%
AMO_NLEN2	LEX_NLEN	93.81%
AMO_NLEN2q	LEX_NLEN	93.71%
AMO_NLENFR2	LEX_NLEN	93.03%

Discussion

- As noted earlier, because LEX_NL outperforms LEX_EN on all acoustic models it is reasonable to assume that Dutch people mainly use a “Dutch” way of pronouncing English rather than a more English like way. It also so means that the training of English phonemes is of relatively less importance than the training of Dutch phonemes.

- However some utterances recognized well using LEX_NL are recognized poorly when using LEX_EN and vice versa, meaning that extra pronunciation variants with English phonemes might increase recognition rate.

- When comparing the performance of LEX_NL on different acoustic models AMO_NLEN2, AMO_NLEN2q and AMO_NLENFR2 outperform AMO_NL2, suggesting that training Dutch phonemes with native English data is beneficiary when recognizing English spoken by Dutch, since only Dutch phonemes were used for recognition. This might be explained by the fact that although some IPA symbols are the same in English and Dutch, the exact realizations and allophones within a language are still different. Training with English data causes the phoneme to be “broadened” with these allophones and also causes increases in the number of triphones. This can be seen more clearly in Table 5.10, which is just a reordering of 5.9.

- Adding French to the training set however decreases performance. This can be seen by comparing the performance of the AMO_NLENFR to the other acoustic models. This can be explained by the fact that

Dutch phonemes are important for recognition for in test, and that this acoustic model is not so good at it (see VoiceConnect97 test results).

- The Lang-tag strategy with language question has minimal impact and sometimes decreases recognition performance. This can be explained by the fact that the Lang-tag question is generally used for recognizing a single language the multilingual model is trained on, but in this case it is a mix of phonemes from different languages which need to be recognized. The Lang-mix strategy is better.
- Finally, the best performing combination, LEX_NLEN on AMO_NLEN performs 2.54% better than the baseline performance of LEX_NL on AMO_NL. There are 2 explanations for this increase:
  - The acoustic model: Even if using only Dutch phonemes for recognition (LEX_NL), the performance is better with AMO_NLEN2, which is trained additionally on English data than, with AMO_NL2.
  - Pronunciation variants made possible by extra phonemes: When using a lexicon with additional entries using English phonemes (LEX_NLEN) the recognition rate improves over using a lexicon with only Dutch phonemes (LEX_NL)

5.4 Differences between English and Dutch phonemes

Because of the results in the previous section it is interesting to see which phonemes the speech recognizer recognizes better using Dutch phonemes alone, and for which English phonemes are required. When phonemes are recognized better using the mapped LEX_NL it can mean that Dutch people tend to use a more Dutch pronunciation to produce these sounds, while when phonemes are recognized better using LEX_EN it can mean that a Dutch recognizer would have some trouble with these phonemes, thus making use of additional English training data the better choice for these phonemes.

Also because a lot of phonemes are recognized well when using either the mapped LEX_NL or LEX_EN, it means that in the acoustic model they must be very similar, so it would mean that for these phonemes additional English training material is less useful. It would be possible to merge these phonemes during training in future versions of the acoustic model, resulting in a acoustic model which in theory would be have less phonemes, and faster. Such an acoustic model was however not trained, because it is unwise to generalize the phoneme similarities and differences based on this small test case alone.

5.4.1 What the speech recognizer could distinguish

Comparing the results of LEX_NL and LEX_EN on AMO_NLEN, 89 differences were found. This means that for 91,26% in both tests the same recognition or misrecognition occurs. It also means that for 91,26% it does not matter which lexicon is used, since the result is the same.

The 89 differences are split as follows: 27 occurrences where using LEX_EN recognized an utterances correctly when using LEX_NL did not, 38 occurrences where using LEX_NL recognized an utterances correctly while using LEX_EN did not, and 24 occurrences where using both lexicons resulted in a misrecognition.

For several words a phonemes which perform better can be identified, because they are the only phoneme within the word transcription which are changed by the mapping. They are described here.

For words the words “team”, “i office”, “fiat dealer” and “easyjet” the only difference in phonemes using the mapping is /i/ versus /i:/. /E/ performs better than /{/ in the words “palet com” and “Swiss air”, /A j/ performing better than /aI/ in the word “primeline”. /A j/ performs better than /aI/ in the word “pioneer”. In the word “pearl opticien” /Y r\/ is recognized better than /3`/.

For the the other words, more than one phoneme was changed, and it is not certain which of these contributes most in the improvement of recognition.

Table 5.11: Words for which instances are being recognized correctly when using LEX_NL but not when using LEX_EN:

Word	LEX_EN transcription	LEX_NL transcription
I office	i:Of@s	iOf@s
KPN research	ka:pe:Enr\i:s3‘tS	ka:pe:Enr\isYrtS
	ka:pe:Enr\1s3‘tS	ka:pe:Enr\IsYrtS
UPS universal parcel service	ju:pi:Esju:n1v3‘s@lpA:r\s@ls3‘v@s	jupiEsjunIvYrs@lpAr\s@lsYrv@s
British airways	br\It1S{r\weIs	br\ItISEr\we:s
	br\It1Se:r\weIs	br\ItISe:r\we:s
easyjet	i:zi:dZEt	izidZEt
	i:zi:jEt	izijEt
eurolines	jUr\oUlaInz	jur\o:lAjnz
	Yro:laInz	Yro:lAjnz
fiat dealers	fiAtdi:l@r	fiAtdil@r
free record shop	fr\i:r\Ek@‘dSA:p	fr\ir\Ek@rdSAp
	fr\i:r\Ek@‘dSOp	fr\ir\Ek@rdSOp
hotel golden tulip	hoUtElgoUld@ntu:lIp	ho:tElgo:ld@ntulIp
intertoys	Int@‘tOIz	Int@rtOjz
kids factory	kIdsf{kt@‘i:	kIdsfEkt@ri
lady line	leIdi:laIn	le:dilAjn
lucky leder	lVki:le:d@r	lOkile:d@r
nijenrode university	nEi@nro:d@ju:n1v3‘s@ti:	nEi@nro:d@junIvYrs@ti
palet com	p{l@tkOm	pEl@tkOm
pearle opticien	p3‘lOptiSEns	pYrlOptiSEns
pioneer	paI@nIr\	pAj@nIr\
playskool	pleIsku:l	ple:skul
price waterhouse coopers	pr\aIswA:t@‘haUsku:p@‘s	pr\AjswAt@rhAuskup@rs
primeline	pr\aImlaIn	pr\AjmlAjn
readers digest	r\i:d@‘zdaIdZEst	r\id@rzdAjdZEst
scoop	sku:p	skup
scoot vindservice	sku:tvInds@‘v@s	skutvInds@rv@s
summertime	sVm@‘taIm	sOm@rtAjm
	sYm@‘taIm	sYm@rtAjm
swiss air	swIs{r\	swIsEr\
team	ti:m	tim
world access	w3‘ld{ksEs	wYrldEksEs
world online	w3‘ldOnlaIn	wYrldOnlAjn

Phonemes which can be singled out are: the English /aI/ performing better than /A j/ in the word “ticketline” and “design”. No other single phoneme can be extracted from this list.

5.4.2 Spectrograms

For some of the phoneme mappings the differences will be looked into in more detail using spectrograms. Generally phonemes pronounced as an English native are compared to the closes sounding phonemes in the Dutch language. For some phonemes it is pretty straightforward as to what the difference is. For example /i/ and /i:/ are basically the same sound with /i:/ having the tone sustained longer, while /i/ is shorter, so only the less straightforward instances will be looked at. The formants can be recognized by the horizontal dark patterns forming a line. Formants are counted from bottom (low frequency) to top (high frequency), with the first two (F_0 and F_1) being the most important.

Table 5.12: Words for which instances are being recognized correctly using LEX_EN but not when using LEX_NL

Word	LEX_EN transcription	LEX_NL transcription
airmiles	{r\maIlz	Er\mAjlz
	e:r\maIlz	e:r\mAjlz
British airways	br\It1S{r\weIs	br\ItISEr\we:s
	br\It1Se:r\weIs	br\ItISe:r\we:s
design	d@zaIn	d@zAjn
eurolines	jUr\oUlaInz	jur\o:lAjnz
	Yro:laInz	Yro:lAjnz
free record shop	fr\i:r\Ek@‘dSA:p	fr\ir\Ek@rdSAp
	fr\i:r\Ek@‘dSOp	fr\ir\Ek@rdSOp
honda dealer	hOndAdi:l@‘	hOndAdil@r\
hotel new york	hoUtElnu:jOr\k	ho:tElnujOr\k
lucky leder	lVki:le:d@r	lOkile:d@r
powerline	paU@‘laIn	pAu@rlAjn
road air	r\oUd{r\	r\o:dEr\
russia travel	r\VS@tr\{v@l	r\OS@tr\Ev@l
	r\YS@tr\{v@l	r\YS@tr\Ev@l
showbizz city	SoUbIzsIti:	So:bIzsIti
ticketline	tIk@tlaIn	tIk@tlajn
world access	w3‘ld{ksEs	wYrldEksEs
world online	w3‘ldOnlaIn	wYrldOnlAjn

/e:/, /E/ and /{/

In Figure 5.2 the spectrograms of the phones /e:/, /E/ and /{/ are shown. The three phones in the figure can all be realized in for example the word “airport”. When /e:/ is used the pronunciation is like a Dutch adaptation of the word, when /E/ is used it is a close approximant to how a native English speaker would pronounce it, and /{/ is how a native English speaker would pronounce it.

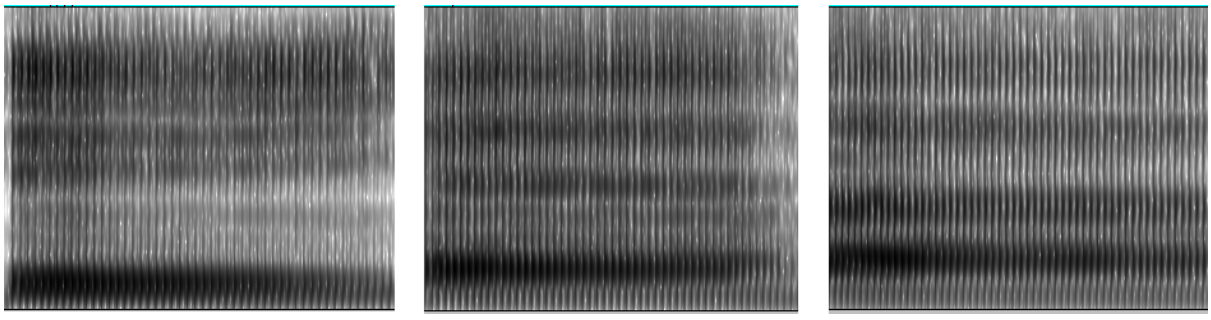


Figure 5.2: Spectrograms of, from left to right, /e:/, /E/ and /{/

The formants from /e:/ are clearly different form those of the other two phones. /E/ and /{/ are more alike. Speech production wise the difference between the three is that from /e:/ to /E/ and /{/ the mouth becomes progressively wider and the tongue is positioned lower and lower in the mouth. In the spectrogram one can see that the first and second formants (the two lowest dark lines) are pretty much apart in /e:/, and come closer in /E/ and are closest to each other in //. The first formant shifts up to a higher frequency while the second shifts down to a lower frequency.

/aI/ and /A j/

In Figure 5.3 the difference is shown between /aI/ and /A j/ which occur in the word “pioneer”. The major difference is in the right half part, the part where the mouth moves from /a/ into /I/ in the left picture, and into /j/ in the right picture. The first formant is sustained in the left picture, while it becomes lower and less intense in the right picture. The second formant is also sustained longer in the left picture.

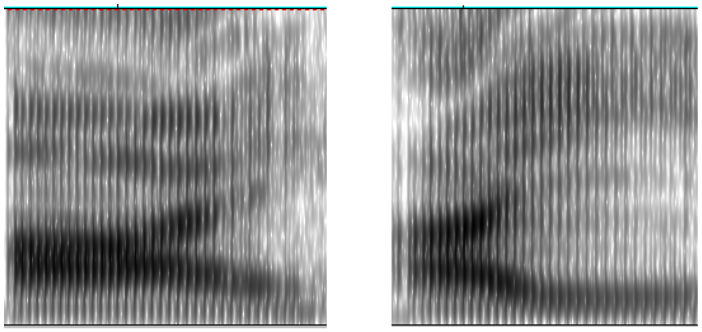


Figure 5.3: Spectrogram of the phonemes /aI/ and /A j/

/3`/ and /Y r\/

In Figure 5.4 the phoneme /3`/ and it’s mapping /Y r\/ are shown. The phoneme /3`/ occurs in words like “world” and “Pearle”. The picture on the left shows the first two formants more or less parallel and the third one rising near the retroflexed ending. The picture one the right shows the second formant rising and falling with its peak at a moment where all the other formants are silent. This silent part is the transition point between /Y/ and /r\/. In the case of /3`/, there is little emphasis on the retroflex part, and the transition sounds smooth. In the other case the transition is much more abrupt with heavier emphasis at the /r\/.

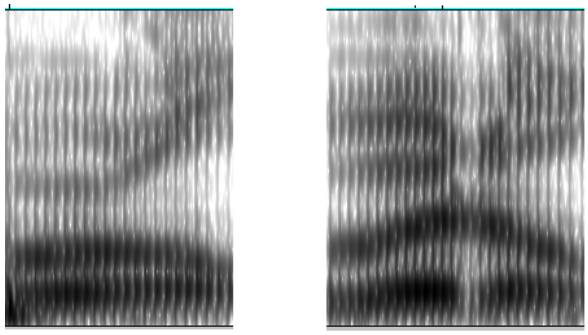


Figure 5.4: Spectrogram of the phonemes /3`/ and /Y r\/

5.5 Conclusion

To answer the questions asked in the beginning of the chapter:
Does using multilingual models improve recognition performance of English utterances spoken by Dutch, and why or why doesn’t recognition improve?

1. How is the performance of multilingual models for Dutch? The VoiceConnect97 tests showed that the

multilingual model could be used to recognize Dutch, at the cost of a small performance degradation, but it could not be described as major, as it is only 0.60% when comparing AMO_NLEN2 to AMO_NL2.

2. How is the performance of multilingual models for native English? When testing on TIMIT the recognition of native English decreased when compared to an English data only trained model, but this test proved that English phonemes could be recognized well by the multilingual model.

3. How is the performance of multilingual models for English spoken by Dutch? Compared to the mapping approach using a pure Dutch acoustic model and a Dutch phoneme only lexicon, performance for English foreign word recognition improved by 0.79% by simply using a multilingual model in stead of the monolingual model. The increase can mount up to 2.54% when also using a more extensive lexicon, made possible by the extra phonemes of the multilingual set.

So in summary from these tests we can derive that a lot of English spoken by Dutch can be recognized using Dutch phonemes and a Dutch recognizer only by using the mapping approach. However some typical English phonemes are still difficult to recognize and are better recognized by using phoneme models trained on English data. An overall benefit is therefore only possible when a sizable amount of utterances consists of English words, and therefore it is still viable to use a Dutch only recognizer to recognize English foreign words, instead of a multilingual model, since the performance increase is not very much. The analysis of the differences in recognition results when using Dutch and English phonemes to recognize the same words prove that using English data in the training of the recognizer is indeed useful for some specific English phonemes in the test case. More extensive tests were not possible due to the lack of availability of suitable material on short notice.

Chapter 6

Call classification

6.1 Introduction

People often call the customer service department of a company of which they bought a product or service from, and often these calls are handled in call centers. Call centers employ many people to answer phone calls (agents), and they are usually specialized in handling only certain types of calls. When someone calls the help line, it is not known beforehand what kind of inquiry he has. Someone has to answer the phone call, listen to his inquiry, and redirect the phone call to the agent specialized in that department. This could easily be remedied by providing a separate phone number for each kind of inquiry, but this is costly and phone numbers are hard to remember, so this approach is less interesting.

For some time now, the above scenario been been automated by IVR (Interactive Voice Response) systems, which greet the callers with a welcome message and a list of menu of options (e.g.: “For billing information press 4”). The problem with IVR systems is that:

1. Some companies have huge menus, or a lot of sub-menus, so callers often get lost. Also there might be a lot of ambiguity in the menu items.
2. Callers have a different mental model of things, so they might be confused because the menu options are not what they expect.
3. Callers become frustrated because they have to listen to a lot of menu options which are irrelevant to them.
4. Such systems require the callers to know clearly where his problem can be handled best. Sometimes callers don’t know themselves what exactly their problem is and can only describe it in a lot of words.
5. Many people just choose the “other” category, or simply a wrong one, hoping that the person that answers can redirect them to the right agent, negating the advantages of an IVR system, because callers simply want to explain in their own language what their problem is instead of having to find out in which menu option their inquiry fits best.

The best solution would be an automated system which can understand callers in their own natural language, and based on the first sentence spoken can redirect the call the person who can help him best. This principle is called an automated call classifier.

There are different types of classification thinkable in call center dialogs. For example based on the tone of the speaker the system should be able to detect whether he is anxious, angry, neutral, happy, sad etc. In the case of angriness it would for example be better to route the call to a more experienced agent. Important factors to classify speech in this way can be rate of speaking, loudness, tremors in the voice etc. Another way to classify speech is

to distinguish between a request or statement. This is especially important for systems which can employ user-initiated dialogs. In the case of a statement, the user is most likely following the system's instructions, but when the user makes a request, the system should halt the current dialog and start a new sub-dialog to try and answer the request. Requests can be detected by certain sentence patterns ("How much...", "Can you...", "I need to know...") etc., but also in the case of a question the utterance usually does not end with a lowering of the voice.

Finally, there is topic classification, where the system should try to discover what topic the user was speaking about. For example if a caller mentions the word "insurance" it is most likely he wants to know something about that topic. In this case "insurance" is a keyword, and keywords typically imply a high correlation between that keyword and a certain topic.

To understand the caller, one needs to recognize what has been spoken first, so a speech recognizer is a must-have. In this section the focus however is not on the recognition but on the understanding and classification of the recognized sentence. Such system have been investigated and deployed in real systems in the United States for some time, most notably the systems used by AT&T. They allow the users to express their request vocally in spontaneous free natural language, and are (ideally) directly routed to the appropriate agent, without callers going through the hassle of listening to the long IVR prompts etc.

Traditionally these speech recognition systems have been based on keyword spotting techniques in combination with a garbage detector. Only keywords are detected and the rest (garbage) is ignored. Certain combinations of keywords can be associated to categories. The list of keywords and their corresponding categories are extracted from domain experts, who are usually experienced agents. Table 6.1 shows some example utterances and keywords. If a human for example hears the word "hypotheek" (en=mortgage) it would be reasonable to classify the call as being from category "hypotheken". Using this kind of domain knowledge, a system can be built using the information from the expert.

Table 6.1: Example of utterances with associated category and possible keywords (chosen by a human)

Sentence	Category	Keywords
Ik heb een vraag over mijn hypotheek	Hypotheken	hypotheek
Ik wil 100 aandelen Heineken	Aandelen order	aandelen, Heineken
Ik wil graag [eeeeh] 1500 euro lenen op korte termijn is dat mogelijk	Nieuwe Lening	lenen
Via internet had ik een reisje geboekt met de creditcard en die deed het niet	Creditcard	internet, reisje, creditcard

More recently large vocabulary recognition has been applied in combination with machine learning techniques originating from the text classification and information retrieval domain. The advantage of such an approach is that less domain specific knowledge is required, as keywords and phrases will be automatically extracted by the training algorithms. Also it can happen that an algorithm will recognize a recurring pattern which a human expert would never have thought of. This idea is attractive, especially when such systems become larger and harder to maintain with manual rules. In this chapter this idea will be further explored. The process is illustrated in Figure 6.1.

The test case which is used is for a financial institution with 45 destinations to route calls to. Since this is a domain for which there is no pre-existing data, recordings were collected from scratch. The 45 classes make this a non-trivial problem, especially since there is quite a lot of overlap in the terminology. The 45 classes or destinations are listed in Table 6.2. Some example utterances were shown before in Table 6.1.

There are classes where the general subject is the same. For example 9 classes are related to different kinds of insurance, 5 are related to stock exchange, 5 related to payments and money transfers, 6 related to remote banking etc. Some classes are not disjunct enough, making classification confusing even for humans. For example there is a general insurance class, but also specific classes for car insurance, life insurance etc.

There are also classes with the same subject but with a difference in the action to be performed. This difference can be quite small (textually), sometimes even one word can make the difference. For example the "overboeking"

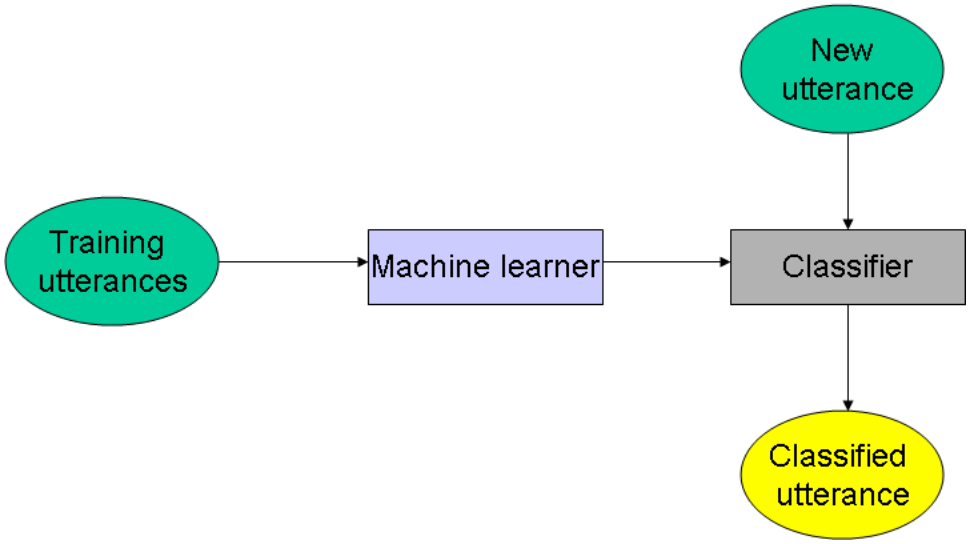


Figure 6.1: Machine learning and call classification process

Table 6.2: 45 Classes

saldo	wijzigen_adres	aansprakelijkheid
andere_categorie	opheffen_rekening	levensverzekering
rekeningnummer	aandelenopinies	autoverzekering
aanvragen_internetbankieren	beurstrends	inboedelverzekering
contract_internetbankieren	opgeven_effectenorder	schade
internetbankieren	opgeven_optieorder	reisverzekering
inloggen	aandelen_order	verzekeringen
wijzigen_toegangscode	pas_aanvragen	rechtsbijstand
hypotheken	pas_blokkeren	suggesties
woonhuis	rekeningnummer	klachten
betaling_geweigerd	automatisch_sparen	gouden_handdruk
incasso_storneren	pensioenvraag	krediet_limiet
bij_afschrijvingen	pensioenverzekering	creditcard
overboeken	openingstijden	nieuwe_lening
spoedoverboeking	afspraak	maximaal_leenbedrag

(=money transfer) class versus the “spoedoverboeking” (=speed money transfer) class. If a distinguishing word such as “spoed” happens to be recognized poorly then the classification is already doomed to fail.

As can be noted from the text before, a call classifier that works with machine learning as a basis is desired. However it is expected that initial classification in the situation described will not be very high, and not at all close to documented text classification problems, for the following reasons:

1. When using proven text classifiers in the call classification domain one must be aware that text classifiers often operate on large bodies of text, whereas a call classifier would have to do with at most 1 or 2 sentences of recognition results.
2. High sentence and word error rates for spontaneous free speech will be expected, because of the lack of training data is is also difficult to get reliable and accurate language models.
3. In literature training and tests are performed with several thousands of examples for training and test set. In our situation there are less than a thousand recordings available for training and testing each.

Thus a list of requirements for the desired classification system:

1. The system must be able to recognize long free speech utterances:
   For recognition the Philips SpeechPearl 2000 recognition engine will be used. It is capable of doing keyword spotting, open grammar recognition, as well as full sentences. SpeachPearl 2000 is of commercial grade, and it's relatively easy to collect speech data from telephone lines using existing software and systems. Also this recognizer has sensible confidence scores per word in the recognition results. As the focus is on the classification, only limited time is spent tuning the recognizer, even though good recognition results are a prerequisite for good classification.

2. The system must work using Machine Learning Techniques:
   One algorithm which hopefully has all the characteristics mentioned above is RIPPER [Cohen95], which is machine learning algorithm well known for its ease of understandability, because it produces a list if-then-else rules to classify items (e.g. Table 6.3). Specifically the WEKA toolkit's RIPPER implementation called JRIP will be used. This chapter describes the set up and results of this experiment.

   Table 6.3: Example of rules produced by JRIP

(inboedel >= 1) => categorie=inboedelverzekering
(idee >= 1) => categorie=suggesties
(geblokkeerd >= 1) and (nieuwe <= 0) => categorie=inloggen

3. The system must perform comparably with standard techniques, such as used by AT&T (BoosTexter):
   The BoosTexter [Schapire00, DiFabbrizio02, Rochery02]) classification algorithm, can be considered state of the art for call classification. It is based on the well known Adaboost [Freund97] algorithm. Because of this some comparative experiments with this algorithm will be done.

4. It is desirable that the system be easy to understand, and modifiable by an expert so that early shortcomings in the system early can perhaps be overcome:
   JRIP's output is easy to understand by humans and can be edited manually.

5. When using speech recognition there is uncertainty in the recognition results:
   Some recognizers give a "score" to each recognized word representing the confidence at which this word was recognized. It is desirable that the machine learning algorithm also uses these confidence scores in a way, so that it will become more robust against noisy data. Experiments will be done with and without taking the confidence scores into account.

6.2 Data collection

A dummy system built with the Philips Speechpearl 2000 package was used to make recordings. In the first two days of data collection 558 calls were made over all categories and transcribed. The next few days an additional 916 calls were made and transcribed, also over all categories. These two sets are the train and test sets respectively. The distribution of calls in the test and train set are not representative of the real-world situation, but are meant to provide each category with at least some training data.

A lexicon and language model were extracted from the training set, and all the data was passed through the speech recognizer. Some time was spent trying to get optimal performance from the speech recognizer, by changing parameters like language model penalty, word insertion penalty etc., but the effort did not lead to results which would be considered "human readable". On the training set the speech recognizer had a word-error-rate of 25.60% and a sentence-error-rate of 64.34%. On the test set these values were 51.49% and 81.55% respectively. For speech recognition these scores are considered bad, but for classification it may not matter, as long as certain key words are recognized.

To prepare this data as input for JRIP, for each word which occurred in the training a real valued attribute was made. For most of the experiments this attribute gets the value of 1 when the word is present in the sentence, and 0 when it is not. In some experiments the attribute's value is the confidence given by the speech recognizer that the word was detected in the recording. The final input of a the machine learner is a vector of ordered values where each value represents the confidence that the word was recognized in the sentence. In the case of BoosTexter, a similar input is used, but the algorithm allows bigrams or trigrams as well as single words.

The recognition results are passed to the classification algorithm as-is. No stop word filtering or stemming is applied, partly because the machine learning algorithm should be able to decide itself what to consider important, and partly because an earlier research with a Dutch stemmer did not seem to produce better classification results [Gaustad02]. Furthermore the training and test sets are relatively small so the vector lengths are still acceptably small.

6.3 Machine learning algorithms

The two machine learning algorithms which can generate classifiers are discussed in detail in this section. JRIP is based on RIPPER (RIPPER)[Cohen95] and generates rules in if-then-else form by growing and pruning, where each rule has an associated class. Items are classified by the first rule that matches.

BoosTexter is based on Adaboost [Freund97] and works by boosting a number of weak classifiers. In this case the weak classifier is a decision stub which determines if a word is present or not in a sentence. Each weak classifier generates a hypothesis. This hypothesis is a number of weights for each class to indicate how important the presence or absence of the word is to belong to that class. The hypotheses scores of the weak classifiers are added together, and the class with the highest score is the class the sentence belongs to.

How RIPPER generates the if-then-else rules and how BoosTexter chooses the weak learners and calculates their weights is described next.

6.3.1 RIPPER

Repeated Incremental Pruning to Produce Error Reduction (RIPPER)[Cohen95], was proposed by William W. Cohen as an optimized version of IREP (incremental reduced error pruning). A two-class version of the IREP is algorithm is given in Figure 6.2. Multi-class problems are handled by sorting the classes in increasing order of prevalence, $C_1, ..., C_k$, from smallest class to biggest. IREP is used to find rules to separate C_1 from $C_2, ..., C_k$. All instances covered by the learned rules so far are removed from the training set. Then IREP is used to separate C_2 from $C_3, ..., C_k$. This is repeated until only the largest class C_k remains, which will be the default class.

RIPPER first obtains a rule set using IREP. IREP starts with an empty rule-set and rules are generated one by one until there are no more positive examples. For each round the training examples are randomly split into a grow and prune-set. The grow-set contains roughly two-thirds of the positive and two-thirds of the negative examples. The rest goes into the prune-set.

The grow-set is used to construct rules. A rule is a conjunction of conditions. Starting with an empty conjunction of conditions, any condition of the form $A_n = v, A_c \leq \theta, A_c \geq \theta$ is considered for the rule, where A_n is a nominal attribute, v is a legal value of A_n, or A_c is a continuous value where θ occurs in the training data. When there are only a few conditions in the rule, the rule is in most cases fairly general, covering some positive and negative examples. As conditions are added the rule becomes more and more specific covering less and less positive and negative examples. Each time the condition which maximizes the information gain is added until the rule is perfect and covers no negative examples from the grow set.

After a rule is created it is immediately pruned. Pruning simplifies the rule generated in the grow phase to prevent overtraining. The rule is evaluated using the prune set and a metric $v = (p - n)/(p + n)$, where p is the number of positive examples covered by the rule, and n the number of negative examples covered by the rule. Each time the

```
procedure IREP(Pos,Neg)
begin
    Ruleset := ∅
    while Pos ≠ ∅ do
        split (Pos, Neg) into (GrowPos, GrowNeg) and (PrunePos, PruneNeg)
        Rule := GrowRule(GrowPos,GrowNeg)
        Rule := PruneRule(Rule,PrunePos,PruneNeg)
        if the error rate of Rule on (PrunePos,PruneNeg) exceeds 50% then
            return Ruleset
        else
            add Rule to Ruleset
            remove examples covered by Rule from (Pos,Neg)
        endif
    endwhile
    return Ruleset
end
```

Figure 6.2: The IREP algorithm

pruning algorithm considers deleting a condition of the rule which increases the value of v until it is maximized.

After growing and pruning the rule is evaluated. In the original IREP if the error rate exceeds 50% on the prune set, the generated rule is thrown away and the algorithm stops. In the RIPPER implementation of IREP, rules are generated until the description length (DL) of the ruleset and examples is 64 bits greater than the smallest DL met so far. After a rule is added all examples covered by the rule, both positive and negative, are removed from the training set.

After an initial rule set is obtained with IREP, RIPPER does a couple of optimization rounds on this rule set. For each rule R_i in the rule set two variants are constructed, a "replacement" and a "revision". The "replacement" is formed by growing and pruning where pruning is guided to minimize the error of the entire rule set on the pruning data. The "revision" is formed analogously, except the revision is grown by greedily adding conditions to R_i rather an empty rule. Finally the decision is made which of the three rules, the original, replacement or revision is the best using the MDL heuristic.

When RIPPER is applied to classifying calls, first the IREP algorithm is called which tries to make a conjunction of (key)words which should be present or not present in the recognition result in order for it to be classified. To find a new rule (conjunction), the algorithm tries all possible combinations of words which should be present or not present. The condition which leads to to biggest information gain is selected. This is in the grow phase. After the grow phase comes the prune phase, and in this phase the rule found in the grow phase is simplified to make it more general. After the initial rule set is made by IREP, RIPPER will post process this set and will construct alternate rules, again using growing and pruning. The best rules are selected from all the alternatives and this is the final RIPPER rule set with which classification will be done.

6.3.2 BoosTexter

BoosTexter is a software program which implements 4 variants of the original Adaboost algorithm, but which can handle multi-class and multi-label problems. In particular "real Adaboost.MH" is used. An overview of the algorithm is shown in Figure 6.3.

Boosting makes a "strong" classification rule out of several "weak" hypotheses. Each "weak" hypothesis will have a different weight, and the weights are computed by concentrating on examples which are hardest to classify. The final "strong" classification is made by computing the sum of all "weak" classifiers. Training the classifier thus involves computing the ideal weights for each "weak" classifier.

Given: $(x_1, Y_1), ..., (x_m, Y_m)$ where $x_i \in \chi$, $Y_i \subseteq \gamma$
Initialize $D_1(i, l) = 1/(mk)$
For $t = 1, ..., T$:
Pass distribution D_t to weak learner.
Get weak hypothesis $h_t : \chi \times \gamma \to \mathbb{R}$
Choose $\alpha_t \in \mathbb{R}$
Update:
$D_{t+1}(i, l) = \frac{D_t(i,l)exp(-\alpha_t Y_i[l]h_t(x_i,l))}{Z_t}$
where Z_t is a normalization factor chosen so that $D_(t + 1)$ will be a distribution
Output the final hypothesis:
$f(x, l) = \sum_{t=1}^{T} \alpha_t h_t(x, l)$.

Figure 6.3: The Adaboost.MH algorithm

In a multi-label problem with k labels, let S be a sequence of training examples $(x_1, Y_1), ..., (x_m, Y_m)$. Each example $x_i \in \chi$ has an associated label $Y_i \subseteq \gamma$ which is +1 if the example belongs to the class or -1 if not. Adaboost.MH maintains a distribution D_t over all examples and labels which is initially uniform.

On each training round t the distribution D_t and training sequence S are passed to a weak learner which computes a weak hypotheses $h_t : \chi \times \gamma \to \mathbb{R}$. The sign of $h(x, l)$ determines whether the label l is assigned to x or not. The magnitude of the prediction is interpreted as the confidence the algorithm has in assigning the label. The parameter α_t is then chosen and the distribution D_t updated in such a way so that example-label pairs which are misclassified get bigger weights.

In BoosTexter the presence or absence of a term or word w is chosen as the "weak" hypothesis. Different weights are assigned depending on whether the word is present or not in x. $h(x, l) = c_{0l}$ if $w \notin x$ and $h(x, l) = c_{1l}$ if $w \in x$. For each term c_{jl}, values are chosen, and a score is defined for the resulting weak hypothesis. Once all terms have been searched, the weak hypothesis with the lowest score is selected and returned by the weak learner.

Let $X_0 = \{x : w \notin x\}$ and $X_1 = \{x : w \in x\}$.
Given the current distribution D_t calculate for each possible label l, for $j \in \{0, 1\}$, and $b \in \{-1, +1\}$:

$$W_b^{jl} = \sum_{i=1}^{m} D_t(i, l)[x_i \in X_j \wedge Y_i[l] = b] \tag{6.1}$$

Now the weights of the decision stub can be computed:

$$c_{jl} = \frac{1}{2} \ln(\frac{W_{+1}^{jl} + \epsilon}{W_{-1}^{jl} + \epsilon}) \tag{6.2}$$

where $\epsilon = 1/mk$.
Finally with $\alpha_t = 1$ the normalization factor can be computed with

$$Z_t = 2 \sum_{j \in \{0,1\}} \sum_{l \in \gamma} \sqrt{W_{+1}^{jl} W_{-1}^{jl}} \tag{6.3}$$

When BoosTexter is applied to classifying calls, it will invoke the weak learner a number of times. The weak learner will go through all the words in the training set and for each word will try to make a weak hypothesis. When a weak hypothesis is selected, the weights are computed as described above, giving each class a score based on whether the word is present or not, and the next call to the weak learner is made. In each round another (key)word is chosen as the weak hypothesis, and the weights recomputed. When an utterance is to be classified,

the algorithm will check the presence or absence of the word, and add for each class the weights put by the weak hypothesis. This is done for all (key)words and the class with the highest score is the "winning" one.

6.4 Tests

Performance of the two machine learning algorithms are compared to each other and the majority baseline (the class with the largest number of samples). The algorithms generate classifiers in 3 ways:

- one on the transcribed data ("perfect recognition"),
- one on recognizer output,
- one on scored recognizer output where the score is generated by the word level confidence of the speech recognizer. The reasoning behind this is that some words or patterns deemed important by a classifier might not be recognized correctly by the speech recognizer.

Classifiers are tested on the training and test sets with both transcriptions and recognizer output. The transcriptions are also used for testing as an indication of what the performance is when there is perfect recognition. This can generally be regarded as an upper bound for performance, and the majority baseline as a lower bound.

JRIP was configured with the default settings, and BoosTexter was configured to use n-grams of up to width 3 and boosting rounds of up to 200, as performance did not improve beyond that point. The rest of BoosTexter's settings were also set to default.

6.4.1 Majority baseline

Purpose

The majority baseline is the most basic classifier and simply classifies all utterances as the largest class in the training set, in this case "saldo". If any "intelligent" classifiers score higher than the results of this test, that means the classifiers are actually working, and not just doing something random. Any score higher than this is a measure of how "intelligent" the classifier is.

Setup

One of the most simple classifiers is used. All utterances are classified as the largest class in the training set, namely "saldo".

Results

Table 6.4 shows the results of the majority baseline. The results are not particularly high, and are equal to the ratio of utterances of category "saldo" in the tested sets.

Table 6.4: Baseline performance

	Training set	Test set
Majority baseline	8.42	12.12

Discussion

- At this point no real conclusions can be drawn. The baseline performance is pretty low, which is logical given the nature of the majority classifier. Future results will be compared to the results in this test. If results are a lot higher, than we can be sure that something intelligent is happening.

6.4.2 Classifiers trained on transcriptions

Purpose

The purpose of this test is to determine the performance when the classifiers are trained on the transcriptions of the training data. Another test will be done later to measure to performance when trained on what is recognized by the speech recognizer instead.

Setup

The classifiers are trained on transcriptions of the training data. Both JRIP and BoosTexter are tested on "perfect" recognition results which are obtained by using transcriptions of the training data as well on real recognizer output.

Results

Table 6.5 and 6.6 show the performance of JRIP and BoosTexter when trained on the training set transcriptions.

Table 6.5: JRIP trained on training-set transcriptions

Tested on	Training set	Test set
Transcriptions	87.28	44.65
Recognizer output	74.37	41.26

Table 6.6: BoosTexter trained on training-set transcriptions

Tested on	Training set	Test set
Transcriptions	98.03	42.47
Recognizer output	79.74	35.37

The results when tested on transcriptions are higher than those when tested on real recognizer output. On the training set BoosTexter performs better than JRIP, but on the test set JRIP performs better.

Discussion

- The results are far better than the baseline (Table 6.4), so it can be said with certainty that the algorithms don't do anything random, but are intelligent indeed.
- The tests performed on the transcriptions score better than those performed on the recognizer output. This is expected, since the recognition is not optimal. However the relative decrease in performance is higher for BoosTexter than for JRIP, so BoosTexter is probably more sensitive to noise.
- JRIP performs better than BoosTexter on the test set, while BoosTexter performs very well when tested again on the train set. This might indicate overtraining, but it could also simply be lack of sufficient training data.

6.4.3 Classifiers trained on recognized output

Purpose

The purpose of this test is to determine the performance when the classifiers are trained on the recognized text after the training data is processed by the speech recognizer. The logic behind this is that the classifier will already be adapted to certain quirks in the speech recognizer, and in doing so will be more robust to some recognition errors. The results here will be compared to the classifier which was trained earlier on on the transcriptions of the data.

Setup

The classifiers are trained on the recognition results of the speech recognizer on the training data. Both JRIP and BoosTexter are tested on "perfect" recognition results or transcriptions of the training data and real recognizer output of the test set.

Results

Table 6.7 and 6.8 show the performance of JRIP and BoosTexter when trained on the training set transcriptions.

Table 6.7: JRIP trained on training-set recognizer output

Tested on	Training set	Test set
Recognizer output	83.87	42.47

Table 6.8: BoosTexter trained on training-set recognizer output

Tested on	Training set	Test set
Recognizer output	99.28	36.03

Discussion

- Again JRIP scores better on the test set than BoosTexter, and again BoosTexter gets a near perfect score when tested on the training set.
- The scores compared to the classifiers trained on transcriptions are higher (Table 6.6 and 6.5), meaning it is probably useful to train on recognizer output instead of transcriptions.

6.4.4 Classifiers trained on scored recognizer output

Purpose

This test determines the performance of the classifiers when they are allowed to make use of the word confidence information in the recognition result. The idea is that words with low confidence might have been wrongly recognized, thus they should be excluded in the evaluation to determine the correct class.

Setup

BoosTexter is now configured to use scored texts, and the training data for JRIP now includes the confidence scored data (real values from 0 to 1), instead of just 1 for word present and 0 for not.

Results

Table 6.9 and 6.10 show the performance of JRIP and BoosTexter when trained on the scored training set transcriptions.

Table 6.9: JRIP trained on scored training-set recognizer output

Tested on	Training set	Test set
Transcriptions	N/A	N/A
Recognizer output	85.30	37.77

Table 6.10: BoosTexter trained on scored training-set recognizer output

Tested on	Training set	Test set
Transcriptions	N/A	N/A
Recognizer output	99.82	29.15

Discussion

- The results from this test are significantly worse than the ones in Sections 6.4.2 and 6.4.3, meaning that at this point it is not a good idea to train on scored output. This is most likely due to the fact that suitable thresholds need to be calculated as well, and there is already not a lot of data.
- JRIP again does better on the test set, while BoosTexter is better on the training set.

6.5 Preliminary conclusions

Taking the results from all tests done so far into account the following can be concluded:

- In the situation which looks most like real-world, testing on recognizer output of the unseen test-set, RIPPER outperforms BoosTexter with 42.27% compared to 39.52%. A full listing of this model (and explanation) is given in the next section in Table 6.11. A lot of keywords seem obvious, but there are also many which make no sense at all, indicating that there is perhaps not enough data to distinguish keywords from a coincidental occurences of that word. So in short, more data is needed.
- When testing on the training data itself, which is in any case not a good idea for a representative indication of performance, BoosTexter performs near perfect with scores of higher than 98%, while JRIP performs much worse. This indicates that the BoosTexter model is well adapted to training data while JRIP is less adapted. Tests were done with BoosTexter with less rounds to see if it helps against overtraining, but the performance did not increase, rather it decreased.
- On the test data, JRIP outperforms BoosTexter in all tests, so JRIP was able to generalize better. For our categorization problem it seems JRIP is the better choice, as BoosTexter might be overly adapted to the training data.

- When testing the classifiers trained on transcriptions, the performance is less with noisy recognizer output than with a classifier trained on recognized output. This is expected, as analogous to a human, it's easier to understand something heard clearly than to understand something with a lot of noise.
- Training on the recognizer output seems to helps improve performance a bit by 1.27% for JRIP and 0,11% for BoosTexter.
- Training on scored recognized data did not have the expected result, as shown by the decrease in performance. A possible reason for this is that the machine learner now not only needs to learn text rules, but needs to learn thresholds for each word as well, creating demand for even more data of which there is already a lack of.

6.6 Improvements

So far the results are far from good, and in practice this would mean that many calls would be directed to the wrong agent. All calls are classified regardless if the classification is of low confidence or not. It would be more meaningful to classify only on the calls which have a fairly high accuracy, and reject the others. The people whose calls are rejected can try again, possibly after being asked by the system to reformulate their request in another way. Also performance might improve when the number of classes is reduced. The JRIP output of the "best" classifier so far is shown in Table 6.11 and a detailed report of its classification ability in Table 6.12.

Table 6.11: The JRIP classifier model trained on the recognized output without confidences

(inboedel >= 1) => categorie=inboedelverzekering (2.0/0.0)
(idee >= 1) => categorie=suggesties (3.0/1.0)
(geblokkeerd >= 1) and (nieuwe <= 0) => categorie=inloggen (3.0/1.0)
(kosten >= 1) => categorie=pensioenverzekering (5.0/1.0)
(verzekeren >= 1) and (het <= 0) => categorie=woonhuis (8.0/2.0)
(vraag >= 1) => categorie=levensverzekering (7.0/3.0)
(creditcard >= 1) => categorie=creditcard (7.0/1.0)
(aandelen >= 1) and (effectenorder >= 1) => categorie=opgeven_effectenorder (4.0/0.0)
(reisverzekering >= 1) => categorie=reisverzekering (7.0/0.0)
(aankoop >= 1) => categorie=nieuwe_lening (3.0/0.0)
(afsluiten >= 1) and (rechtsbijstandverzekering <= 0) => categorie=nieuwe_lening (7.0/3.0)
(afsluiten >= 1) => categorie=aansprakelijkheid (3.0/0.0)
(aansprakelijk >= 1) => categorie=aansprakelijkheid (4.0/0.0)
(gebruiker >= 1) => categorie=contract_internetbankieren (5.0/1.0)
(internetbankieren >= 1) and (wil >= 1) => categorie=contract_internetbankieren (4.0/1.0)
(sparen >= 1) => categorie=automatisch_sparen (6.0/1.0)
(pensioengat >= 1) => categorie=pensioenvraag (4.0/0.0)
(pensioentekort >= 1) => categorie=pensioenvraag (2.0/0.0)
(anders >= 1) => categorie=spoedoverboeking (8.0/3.0)
(spoedoverboeking >= 1) => categorie=spoedoverboeking (2.0/0.0)
(afkoop >= 1) => categorie=gouden_handdruk (2.0/0.0)
(gouden >= 1) => categorie=gouden_handdruk (3.0/0.0)
(ontslagvergoeding >= 1) => categorie=gouden_handdruk (3.0/0.0)
(zie >= 1) => categorie=internetbankieren (5.0/1.0)
(internetbankieren >= 1) => categorie=internetbankieren (3.0/1.0)
(graag >= 1) and (opvragen >= 1) => categorie=beurstrends (6.0/0.0)
(koers >= 1) => categorie=beurstrends (3.0/1.0)
(maximaal >= 1) and (wat <= 0) => categorie=maximaal_leenbedrag (10.0/3.0)
Continued on Next Page...

(laten >= 1) => categorie=maximaal_leenbedrag (3.0/0.0)
(boos >= 1) => categorie=klachten (3.0/0.0)
(klacht >= 1) => categorie=klachten (5.0/0.0)
(nogal >= 1) => categorie=klachten (2.0/0.0)
(schade >= 1) => categorie=schade (8.0/0.0)
(jullie >= 1) => categorie=schade (3.0/0.0)
(autoverzekering >= 1) => categorie=autoverzekering (5.0/0.0)
(verzekeren >= 1) => categorie=autoverzekering (4.0/0.0)
(rood >= 1) => categorie=krediet_limiet (6.0/0.0)
(limiet >= 1) => categorie=krediet_limiet (5.0/0.0)
(opheffen >= 1) => categorie=opheffen_rekening (6.0/0.0)
(opzeggen >= 1) => categorie=opheffen_rekening (4.0/1.0)
(ga >= 1) => categorie=opheffen_rekening (2.0/0.0)
(maar >= 1) and (via >= 1) => categorie=betaling_geweigerd (11.0/0.0)
(bankzaken >= 1) => categorie=aanvragen_internetbankieren (10.0/0.0)
(advies >= 1) => categorie=rechtsbijstand (5.0/0.0)
(mij >= 1) => categorie=rechtsbijstand (2.0/0.0)
(advocaat >= 1) => categorie=rechtsbijstand (3.0/0.0)
(bijstand >= 1) => categorie=rechtsbijstand (2.0/0.0)
(bedrag >= 1) and (overboeken <= 0) => categorie=incasso_storneren (6.0/1.0)
(storneren >= 1) => categorie=incasso_storneren (4.0/1.0)
(terugboeking >= 1) => categorie=incasso_storneren (7.0/1.0)
(hypotheekadvies >= 1) => categorie=hypotheken (4.0/1.0)
(huis >= 1) => categorie=hypotheken (3.0/1.0)
(hypotheek >= 1) => categorie=hypotheken (2.0/0.0)
(adresgegevens >= 1) => categorie=wijzigen_adres (7.0/0.0)
(adres >= 1) => categorie=wijzigen_adres (7.0/0.0)
(verwachtingen >= 1) => categorie=aandelenopinies (11.0/1.0)
(ontwikkeld >= 1) => categorie=aandelenopinies (3.0/0.0)
(kantoor >= 1) => categorie=openingstijden (11.0/0.0)
(open >= 1) => categorie=openingstijden (6.0/0.0)
(tin >= 1) => categorie=wijzigen_toegangscode (11.0/0.0)
(toegangscode >= 1) => categorie=wijzigen_toegangscode (7.0/0.0)
(spreken >= 1) => categorie=afspraak (15.0/1.0)
(opgeven >= 1) => categorie=opgeven_optieorder (6.0/0.0)
(een >= 1) and (order >= 1) => categorie=opgeven_optieorder (6.0/2.0)
(call >= 1) => categorie=opgeven_optieorder (2.0/0.0)
(kopen >= 1) and (aandelen <= 0) => categorie=opgeven_optieorder (7.0/2.0)
(wat >= 1) and (is >= 1) => categorie=bij_afschrijvingen (11.0/1.0)
(er >= 1) and (rekening >= 1) => categorie=bij_afschrijvingen (4.0/0.0)
(afschrijving >= 1) => categorie=bij_afschrijvingen (4.0/1.0)
(bijgeboekt >= 1) => categorie=bij_afschrijvingen (2.0/0.0)
(niet >= 1) and (dat <= 0) => categorie=pas_aanvragen (10.0/0.0)
(pas >= 1) => categorie=pas_aanvragen (11.0/5.0)
(kapot >= 1) => categorie=pas_aanvragen (4.0/0.0)
(gestolen >= 1) => categorie=pas_blokkeren (9.0/0.0)
(bankpas >= 1) => categorie=pas_blokkeren (10.0/2.0)
(pasje >= 1) => categorie=pas_blokkeren (5.0/0.0)
(uitgevoerd >= 1) => categorie=aandelen_order (10.0/0.0)
(aandelen >= 1) => categorie=aandelen_order (14.0/4.0)
(betalen >= 1) => categorie=overboeken (9.0/0.0)
(naar >= 1) => categorie=overboeken (10.0/0.0)

Continued on Next Page...

```
(overboeking >= 1) => categorie=overboeken (9.0/2.0)
(overboeken >= 1) => categorie=overboeken (4.0/1.0)
=> categorie=saldo (80.0/39.0)
```

Table 6.12: Detailed report of best classifier

Precision	Recall	F-Measure	Class
88.89%	27.59%	42.11%	pensioenvraag
55.56%	55.56%	55.56%	incasso_storneren
70.00%	26.92%	38.89%	internetbankieren
50.00%	12.50%	20.00%	suggesties
92.31%	66.67%	77.42%	autoverzekering
25.00%	31.25%	27.78%	aandelen_order
27.78%	21.74%	24.39%	bij_afschrijvingen
86.21%	65.79%	74.63%	schade
29.73%	79.28%	43.24%	saldo
34.09%	62.50%	44.12%	nieuwe_lening
80.77%	84.00%	82.35%	gouden_handdruk
100.00%	43.59%	60.71%	wijzigen_adres
7.69%	25.00%	11.76%	maximaal_leenbedrag
26.92%	63.64%	37.84%	opheffen_rekening
17.95%	46.67%	25.93%	pas_aanvragen
14.29%	23.08%	17.65%	woonhuis
75.00%	27.27%	40.00%	inboedelverzekering
100.00%	66.67%	80.00%	wijzigen_toegangscode
82.14%	76.67%	79.31%	klachten
42.86%	25.00%	31.58%	beurstrends
50.00%	52.63%	51.28%	overboeken
75.00%	66.67%	70.59%	automatisch_sparen
12.82%	41.67%	19.61%	opgeven_optieorder
40.00%	33.33%	36.36%	rechtsbijstand
64.52%	37.74%	47.62%	pas_blokkeren
81.82%	42.86%	56.25%	openingstijden
60.00%	16.67%	26.09%	krediet_limiet
40.00%	57.14%	47.06%	creditcard
5.56%	10.00%	7.14%	levensverzekering
33.33%	14.29%	20.00%	afspraak
74.07%	57.14%	64.52%	reisverzekering
75.00%	64.71%	69.47%	hypotheken

Rejected utterances = 0
Totaal: 916, correct = 389, wrong=527, Accuracy = 42.46%

Table 6.11 shows all the generated rules this model consists of, and how keywords are linked to classes. The output should be interpreted as a series of if-then-else rules where ">=1" means that the word should have been recognized in the sentence with a confidence equal or higher than 1 and <=0 means it should not have been recognized or with confidence lower or equal to 0 (Note: the values in the listing are all 1 and 0, because it was not trained with real recognizer confidences). The last line in the rules is the default rule, namely "=> categorie=saldo", meaning any utterance which did not match any of the other rules will be classified as "saldo". The words which occur in the output could be regarded as what the algorithm considered as "keywords".

Analyzing the rules in the classifier, the default JRIP output is a classifier which consists of a series of if-then-else rules with majority class taken as the default rule. This means that any utterance which does not match any of the

rules will automatically be classified as the majority class, in this case “saldo”. But logically if it were an unclear sentence it should have been rejected. The JRIP output of the “best” classifier so far is shown in Table 6.11.

Some rules are very logical, for example “(inboedel >= 1) => categorie=inboedelverzekering” seems pretty straightforward, but some seem to make no sense at all, for example “(mij >= 1) => categorie=rechtsbijstand”. The latter indicates overtraining or lack of enough data to make good general rules.

6.6.1 Refining the model

Some modifications to the JRIP classifier were made in order to add rejection capabilities. To implement a rejection facility a classifier is trained for the majority class only, namely “saldo”. The training data is relabeled so that all except the majority class belong to one class, in essence creating a binary classification problem. The default rule for “saldo” is then replaced by the rules of this second classifier. The end result is that the classifier will now reject any utterance which does not trigger any of the rules, instead of assuming a “saldo” when there is no match.

Next modifications were needed with regard to how to deal with recognizer confidences. Using them to in the training stage did not have the desired effect, but there might be other ways to make use of this information. First an approach was tried where recognition results were filtered before being passed to the classifier based on a threshold. This way only words with sufficient confidence are passed to the classifier. This is illustrated in Table 6.4. This was also considered for the the original model without rejection, but because of the default rule

Recognized	ja(0.258) ik(0.560) heb(0.417) deze(0.270) er(0.964) een(0.576) klacht(1.000)
Filtered	ik heb er een klacht

Figure 6.4: Example: Filter out words with confidence lower than 0.400

this can not work, and items that failed to be classified by the rules were classified as the majority class. Other attempts were made to do fuzzy math with the word confidences, giving each rule a score (each rule leads to a classification), with the best score selected in the end, but this did not lead to an improvement.

6.6.2 Reducing the number of classes

As the classification task is fairly complex and the performance reached up till now has not been spectacular, the problem was simplified. Classes with similar subjects were grouped together reducing the amount of classes by a factor of 3. For example the many classes for insurance were merged into one class etc. So the number of classes decreases and generally the amount of training data per class increases. The classes were reduced according to Table 6.13.

6.7 Tests (2)

The improvements mentioned in the foregoing section are tested now. First the JRIP with rejection facility and configurable threshold is tested, and after that the same problem with reduced classes.

6.7.1 Improved JRIP rules with rejection facility

Purpose

Since words are now filtered before they are passed to the classifier, and the default output from JRIP was changed so that it doesn’t classify what it does not recognize as the majority class, the classifier must be tested again.

Table 6.13: Mapping of old and new classes

New class	Old classes
saldo	saldo
overig	andere_categorie rekeningnummer
telefoon & internet bank.	aanvragen_internetbankieren contract_internetbankieren internetbankieren inloggen wijzigen_toegangscode
betalingen	betaling_geweigerd incasso_storneren bij_afschrijvingen overboeken spoedoverboeking
administratie	wijzigen_adres opheffen_rekening
beurs, effecten en aandelen	aandelenopinies beurstrends opgeven_effectenorder opgeven_optieorder aandelen_order
passen	pas_aanvragen pas_blokkeren

New class	Old classes
hypotheken	hypotheken woonhuis
sparen	rekeningnummer automatisch_sparen
pensioen	pensioenvraag pensioenverzekering
informatie	openingstijden afspraak
verzekeringen	aansprakelijkheid levensverzekering autoverzekering inboedelverzekering schade reisverzekering verzekeringen rechtsbijstand
feedback	suggesties klachten
gouden_handdruk	gouden_handdruk
creditcard	krediet_limiet creditcard
lenen	nieuwe_lening maximaal_leenbedrag

Setup

Two JRIP models were trained. One using the same method as in the first series of tests. The other was trained by first renaming the data so that it becomes a binary classification problem by renaming all tags which are not “saldo” to “not_saldo”. The resulting rules from both classifiers were merged into one classifier.

Results

Tables 6.14 and 6.15 show the corresponding results after the modifications for various values of the threshold. The relative accuracy is the ratio of correctly classified calls from those which were not rejected, and absolute accuracy is the ratio of correctly classified calls from all calls, rejected and non-rejected.

Discussion

Again the classifier trained on recognized output does slightly better than the one trained on transcriptions at comparable rejection levels. The rejection facility works, and it adds another parameter for configuring the classifier. Higher classification accuracy can be gained by setting the recognition threshold higher at the cost of more rejected calls.

Comparing the most similar tests in the previous test series (Table 6.5 and 6.7) which showed the best achievable accuracy result was around 42% (with no rejection allowed), the results are now ranging from 53% to 69% (relative) when the classifier is allowed to reject calls for varying values of the threshold. This improvement has a downside and will be explained further.

Table 6.14: JRIP trained on training-set transcriptions with rejection

Threshold	Rel. Accuracy %	Rejection%	Abs. Accuracy%
0.000	53.32	26.09	39.41
0.100	54.11	26.97	39.52
0.200	55.57	28.49	39.74
0.300	57.05	30.35	39.74
0.400	58.09	32.53	39.19
0.500	59.69	35.81	38.32
0.600	60.78	38.76	37.22
0.700	62.19	42.25	35.91
0.800	63.60	46.62	33.94
0.900	65.13	50.22	32.42
1.000	68.06	55.57	30.24

Table 6.15: JRIP trained on training-set recognizer output with rejection

Threshold	Rel. Accuracy %	Rejection%	Abs. Accuracy%
0.000	52.93	23.69	40.39
0.100	53.31	24.24	40.39
0.200	54.86	25.98	40.61
0.300	56.21	27.95	40.50
0.400	57.75	30.24	40.29
0.500	58.51	32.64	39.41
0.600	60.45	36.24	38.54
0.700	62.13	40.61	36.90
0.800	63.94	45.20	35.04
0.900	65.96	48.69	33.84
1.000	69.45	54.26	31.77

When counting rejections as misclassified, and thus computing the absolute accuracy, the scores in this test are ranging from 30% (highest threshold) to 40% (lowest threshold). The setting with the lowest threshold is the most comparable to the previous test series, and with 40% versus 42% in the previous test series this is a small decrease. The decrease can be explained by the fact there is now an additional classifier which distinguishes between “saldo” and “not_saldo” and the classifier can fail to recognize some calls as “saldo”, whereas in the previous test series calls were given the label “saldo” by default if they did not match any other rule. However most of the calls which are now rejected would be false positives and wrongly classified as “saldo” in the previous test series.

Generally, for the sake of improving the dialog success rate, it is better to have a higher accuracy ratio when the classifier makes a classification, instead of having a lot of false positives where the caller is directed to the wrong agent, as would be the case in the previous test series. In the case of a rejection the caller can be asked to restate his request differently, and if it fails again the caller can ultimately be directed to a human agent. So overall the modifications are successful and a definite improvement, even if the absolute number of correctly classified calls is a bit lower.

6.7.2 Reduced classes

Purpose

Because of overtraining and lack of enough data for each class, the data was relabeled to have fewer, but broader classes, and with more data for each broad class. This test serves to see how the performance of the classifier changes when it has to deal with the simplified problem.

Setup

The data was relabeled according to Table 6.13. The classifier was constructed much the same way as the other with rejection facility and consisting of rules from 2 classifiers.

Results

The corresponding results are shown here in Table 6.16 for varying values of the threshold. The relative accuracy is the ratio of correctly classified calls from those which were not rejected, and absolute accuracy is the ratio of correctly classified calls from all calls, rejected and non-rejected.

Table 6.16: JRIP trained on reduced classes training-set recognizer output with rejection

Threshold	Rel. Accuracy %	Rejection%	Abs. Accuracy%
0.000	72.17	26.53	53.02
0.100	72.86	27.07	53.14
0.200	75.23	28.93	53.47
0.300	76.90	30.90	53.14
0.400	79.01	33.30	52.70
0.500	79.18	35.92	50.74
0.600	80.69	39.41	48.89
0.700	81.34	43.12	46.27
0.800	82.87	48.25	42.89
0.900	84.39	51.64	40.81
1.000	86.18	57.21	36.88

Discussion

As this is now a simplified problem with less confusion between categories and more training data for each category, the accuracy is improved from 72% to 86% for varying values of the threshold. As expected, at comparable rejection levels in Table 6.15, both the relative and absolute accuracy scores gotten in this test are higher.

6.8 Conclusion

With the list of requirements defined in the beginning of this chapter, they were achieved in the following manner:

1. The system must be able to recognize long free speech utterances:
   For recognition the Philips SpeechPearl 2000 recognition engine was used. However the word and sentence error rates were reasonably high, and definitely some improvement is needed here. By using more recent speech recognition technology and by gathering better language models improvements will be gained in this department. Correct recognition is preferred in order to properly classify the utterances. Even though sentence and word error rates were high there was no huge difference in performance when comparing the performance when testing on perfect recognition (transcriptions) and actual recognition output, meaning that the classifiers are somewhat robust against recognition errors.

2. The system must work using Machine Learning Techniques:
   All rules are automatically generated by JRIP, and no human intervention is required. The classification scores of JRIP are satisfying given the circumstances and it's relative performance compared to BoosTexter,

but a long way from perfect. [1] More data is required and how the algorithm scales remains to be seen once data becomes available.

3. The system must perform comparably with standard techniques, such as used by AT&T (BoosTexter):
   In the performed experiments BoosTexter was consistently outperformed by JRIP when testing on new unseen data. BoosTexter seemed sensitive to noise and had a tendency to overtrain.

4. It is desirable that the system be easy to understand, and modifiable by an expert so that early shortcomings in the system early can perhaps be overcome:
   JRIP's output is easy to understand by humans and can in theory be edited manually. However this was not attempted in these experiments.

5. When using speech recognition there is uncertainty in the recognition results:
   By filtering the data sent and modifying the classifier to have a rejection facility it was possible to make useful use of speech recognizer confidences, by only allowing words which had confidences above the specified threshold.

Overall the machine learning classifiers do not perform at an acceptable operational level yet, therefore these must be improved first. The apparent lack of data is a serious hindering factor in the training procedure. Using JRIP does not seem inferior to BoosTexter at this stage, but the performance of both leaves much to be desired. Unfortunately it is only possible to know once more data becomes available.

Yet, still some ways to improve performance can be thought of. Adding rejection and reducing the number of classes improved the performance, but the future several other improvements can also be done. With JRIP there is the possibility of easily modifying the auto-generated rules with some human knowledge, especially in the beginning stages of a project, since it works pretty much like an expert system. Even though overall performance is still low, one can make use of the fact classification performance is not the same for all classes. By only allowing high scoring classes to be classified, thus further reducing the number of classes, the overall performance can go up even further, again at the cost of higher rejection.

[1] Although details of a system consisting of human rules only were not given here, JRIP seems comparable in performance on the same training and test set.

Chapter 7

Summary, Conclusions and Future Work

The results of the research in the two topics discussed in this thesis were limited, but yet insightful and will hopefully contribute to better products and services to be developed in the future. The two topics shared one common problem, namely limited training data, and in both topics there were promising methods to solve this shortcoming. A short summary of the topics English Foreign Word Recognition and Call Classification which were discussed in detail in Chapter 5 and 6 respectively, will be given with the final conclusions and future work, each in their own section.

7.1 English foreign word recognition

Summary and Conclusions

English words are an increasing phenomenon in Dutch speech. It is common that one or several English words or names are used within a Dutch sentence. This is a potential problem because English has phonemes which are not typical for Dutch, and existing Dutch speech recognizers are designed with Dutch phonemes in mind. Non native speakers, such as the Dutch, might adopt the English phonemes successfully, but often they will (unknowingly) to some extent use the phonemes which are closest in their native inventory instead of an English one. Because of this variety in how Dutch can pronounce English words it is important to recognize both Dutch and English phonemes.

The problem definition posed in Section 1.2.1 is whether using a Dutch recognizer is enough to handle this increasing phenomenon, or if adding training data from other languages, particularly English, using multilingual techniques are a better option.

Getting a Dutch speech recognizer to recognize the English words spoken by Dutch people traditionally required gathering recordings of English utterances from the Dutch speaker group. Collecting a new corpus is expensive, time-consuming and laborious. Thus the attention was put on multilingual techniques using existing corpora. During the course of this work, an existing Dutch corpus was used in conjunction with existing English and French corpora.

Using corpora from various languages and sources posed a number of problems (Section 5.2). For the English corpus it was necessary to down-sample the audio to telephone-like speech. The main problem however is choosing an appropriate phoneme set for all languages involved. Different organizations and languages tend to have different definitions of the phoneme set or choose a very fine granularity, so the first problem was to define a unified phoneme set which covers all training languages. The phoneme sets chosen were based for the most part on the sets defined by SAMPA, and unified using IPA correspondences with an X-SAMPA notation. For training, using alignments from pre-trained monolingual recognizers might not be optimal, thus an iterative approach was chosen

to train the multilingual recognizer. A reference corpus was chosen and this was used to align a new language. After alignment the new language was added to the reference corpus, which was in turn used to align the next corpus. Several acoustic models combinations were trained with Dutch, English and French data.

Different tests were done to test the various models. First of all a test with the VoiceConnect97 corpus to verify that the recognition performance on native Dutch does not decrease (too much) with the multilingual model (Section 5.3.1). Second, a test on native English using the TIMIT corpus (Section 5.3.2) which heavily tests the English phonemes, and third a test on English words spoken by Dutch people using the DDAC2000 corpus (Section 5.3.3). A summary of the results of these three tests can be found in Table 7.1, Table 7.2 and Table 7.3 respectively.

The native Dutch test using Voiceconnect97 showed that the expected performance decrease was within acceptable limits, dropping from 93.36% to 92.66% when comparing the monolingually trained Dutch acoustic model (AMO_NL2) to the Dutch-English model (AMO_NLEN2).

Table 7.1: Summary of test results from VoiceConnect97

Model	Correct
AMO_NL2	93.36%
AMO_NLEN2	92.66%
AMO_NLEN2q	92.72%
AMO_NLENFR2	91.34%
AMO_NLFR2	91.02%

In the TIMIT test the monolingually trained English acoustic model performed best, with the other acoustic models performing less and the Dutch-English model trained with language questions (AMO_NLEN2q) the worst.

Table 7.2: Summary of test results from TIMIT

TIMIT	
Model	W. Correct
AMO_EN	91.8%
AMO_NLEN2	91.1%
AMO_NLENFR2	91.2%
AMO_NLEN2q	90.8%

The DDAC2000 test was the most interesting, as it resembled the actual problem the most, with Dutch speakers pronouncing English words, thus a mix of Dutch and English phonemes. The various acoustic models were tested using different lexicons. Initially using the lexicon with Dutch phonemes only (LEX_NL), with English phonemes mapped to Dutch ones, performed better than the lexicon with primarily English and some Dutch phonemes (LEX_EN), but different errors were made in each case. This lead to the conclusion that in general the use of Dutch phonemes is more prevalent, but there are instances where the specific detection of English phonemes lead to better results.

Table 7.3: Summary of test results from DDAC2000

DDAC2000		
Acoustic model	Lexicon	Correct
AMO_NLEN2	LEX_EN	90.96%
AMO_NL2	LEX_NL	91.45%
AMO_NLEN2	LEX_NL	92.24%
AMO_NLEN2	LEX_NLEN	93.81%

Motivated by that, the two lexicons were combined to LEX_NLEN and tested with the Dutch-English acoustic model (AMO_NLEN2). The results were higher than previously achieved in both cases. Comparing this result to the baseline case of a recognizer trained with Dutch data only (AMO_NL2) and English phonemes mapped

(LEX_NL) the performance went from 90.96% to 93.81%. Furthermore testing with the Dutch lexicon on the different acoustic models showed that the Dutch-English trained acoustic model performed better than the Dutch acoustic model, leading to the conclusion that even if English phonemes are not explicitly detected, training Dutch phonemes with English data is beneficial and provides additional robustness. The differences between English and mapped Dutch phonemes were investigated further in Section 5.4.

In the VoiceConnect97 and DDAC2000 test, the acoustic models with French in the training data performed relatively worse than the other ones. Several possible reasons can be thought of: French might be significantly different from Dutch and English, there were problems with the automatic generation of the French lexicon, or not enough French training data was used.

In conclusion, better recognition results for English foreign words spoken by Dutch people are obtainable using the multilingual techniques, but at a slight cost of recognition performance on pure Dutch material. This can be derived from the fact that the performance of the best Dutch-English model dropped slightly when testing on the Dutch only material in VoiceConnect97, and there was a small performance gain when testing on the English-spoken-by-Dutch material in DDAC2000. So to get an overall benefit from multilingual models, there must be a significant amount of expected English utterances, compared to the amount of Dutch utterances.

Future Work

The results achieved so far are promising, but some more experiments can be thought of in order to deepen the understanding of the topic.

- A Bigger and more extensive test set:
  The DDAC2000 test set was relatively small and consisted of short utterances. A bigger test set, as well as experiments with longer utterances would give more accurate and more representative results.

- Different English to Dutch phoneme mappings:
  In the performed experiments the mappings were chosen somewhat intuitively with the simple heuristic that a Dutch text to speech synthesizer should be able to pronounce decent English using those mappings, and it should be the phoneme which a Dutch person would most likely use when pronouncing an English phoneme. Of course opinions on whether the produced English is close enough to real English are subjective, and not all people will borrow the same phoneme, so other mappings are thinkable, perhaps based on other methods, such as data-driven methods.

- Reducing the phoneme set:
  In these experiments the phonemes of the different languages were merged based on IPA correspondences. This results in a huge phoneme set, which is not always practical, for example transcription, and amount of training data for each phoneme. It would be thinkable to further reduce the phoneme set further in cases where the English phoneme does not at all perform better than the mapped Dutch phoneme, thus only keeping English phonemes in the set which contribute to better recognition.

- Adding other languages:
  In these experiments adding French to the training data did not lead to performance improvements. French was considered because non-native recognition is often boosted by more training material of foreign origin. French is perhaps not close enough to either Dutch or English, or simply problems with the French transcriptions, causing a performance decrease. In any case, other languages, for example German, might perform satisfactory, as this language seems closer to both English and Dutch, and in the case of multilingual models and speech recognition there is no better data than more data.

7.2 Call classification

Summary and Conclusions

Call classification is the automated decision making of to which agent a call should go after the caller expresses his wishes and this is recognized by the speech recognizer. The caller is invited to express his desire with a simple opening prompt like “How may I help you?”. Such an approach would hopefully be cheaper and faster to work with, as agents are freed up to handle actual problems instead of routing callers and callers are directly send to the right person.

Algorithms which can automatically construct classifiers are attractive, since they are more flexible, work faster and cheaper than human experts, and can be invoked at any given time. However such algorithms require training data. This data takes time to acquire, and this is opposite to the desire of the call center who would like to have the system working yesterday. Also the output of some algorithms is not always understandable by humans and thus not easy to tweak if anything is not going as hoped for.

In the beginning phases of a new classification system there is no or insufficient training data available. It is likely that a human expert will perform better when there is not enough training data to go around, and this will progressively change as more data comes available, and progressively harder for the human expert to keep up with the amount of data.

In Section 1.2.2 a problem definition was posed of how create a call classifying system based on machine learning techniques which performs well in the beginning stages with limited data available. The classifiers generated by the machine learner should be easily understandable so that a human expert can supervise and modify them as needed. Once more data is available the need for supervision will decrease as the classifiers will become more accurate.

The first problem was to collect data for a 45 class problem. Two days were spent recording the data for a little over 1500 calls. This was not a lot, but realistic given the goal of creating an algorithm which does decently in the beginning phases and hopefully extensible for the future when there is more data.

The second problem to deal with was getting the speech recognizer to recognize utterances reliably. Recognition need not be perfect, as long as specific keywords which are unique for a class are recognized. As recognition was not the main focus of this research topic, only one recognition pass was done and the output of this pass used as input for training and testing the algorithms. The recognition performance was sadly far from perfect and there is room for improvement here.

Two algorithms were tested. These were JRIP, based on RIPPER and known for generating human-readable rules, and BoosTexter which is based on the Adaboost algorithm but adapted for text. BoosTexter was already used in literature for similar tasks of classifying calls. JRIP is particularly attractive because the output is very easy to understand, and in the future could serve as a basis for a human expert to continue with.

The algorithms were tested by training on transcriptions, training on recognition output and training on scored recognition output (Section 6.4). A short summary of the results is shown in Table 7.4. For both algorithms, training on recognition output yielded the best results, with JRIP getting the highest score of 42.27% and BoosTexter 36.03%, followed by training on transcriptions. Training on scored output (with recognizer confidences) gave scores significantly lower. Overally the scores are much higher than the majority baseline of 12.12% and a bit lower than the maximum scores of 44.65% (JRIP) and 42.47% (Boostexter) which can be achieved with “perfect” recognition (using transcriptions).

Table 7.4: Accuracy of classifiers when tested on speech recognition output

Trained on	JRIP	BoosTexter
Transcriptions	41.26	35.37
Recognized output	42.47	36.03
Scored output	37.77	29.15

In general the impression is that JRIP can generalize better than BoosTexter, is more robust to noise, and has the better scores on the tests set. BoosTexter performed a lot better than JRIP when testing on the training set again, which is perhaps a sign of overfitting. Training on scored (with recognizer confidences) texts disappointedly gave scores significantly lower than when training on just the output without scores. This is probably attributed to the fact that the algorithm not only needs to learn the keywords but also needs to find suitable thresholds, meaning that more training data is needed.

Analyzing the errors made by JRIP showed that many of the errors resulted from a default association with the majority class when no rule matched the utterance. JRIP was then modified with a rejection facility, where utterances which did not match any rule were rejected instead of being classified as the majority class. In doing so a word confidence threshold was put in place. This lead to a better relative classification score with accuracies from around 52% to 69%, for different values of the threshold with rejections, but a slightly worse absolute accuracy of about 40% when counting rejections as misclassifications (Section 6.7.1).

Because of the huge number of classes, and relatively few training samples some classes were underrepresented. Also the system could easily get confused by a number of related classes. The 45 classes were reduced in number, and related classes joined together. As expected the classification performance went up to around 72% to 68% relative accuracy for varying values of the threshold (Section 6.7.2).

Overall the scores in all tests were not what one would expect from a system in operational use, but this can be attributed largely to a lack of training data. The results are still promising, because an easy to understand algorithm such as JRIP outperformed a well known.

Future Work

- Bigger training and tests sets which are representative of actual situations:
  The training and test data were collected in a relatively short time, partly to simulate the beginning stages of creating a call classifier or perhaps a brand new call center (which was the idea). That being said, it was possible to make something which performed well given the circumstances, but more data is needed to determine whether performance of the classification algorithms will indeed improve once it becomes available, and whether the performance of the algorithms relative to each other remains the same. More training data will make it possible to extract better language models and a more extensive lexicon, which will in turn lead to better speech recognition, and better classification performance.

- Extending the system with rules from a human expert:
  Only the base performance of classifiers generated by the machine learners was tested. Not all rules always make sense, and perhaps a human expert has some additional rules which can further improve performance.

- Only concentrate on categories with a lot of true positives:
  Because the performance per individual class varies, it's maybe wise to only classify utterances for classes for which it is known that the system has a high probability of getting it right, if the system were to go operational. If the system classifies a call to a class for which it is known that it is often incorrect it should be directed to a human operator. Although this system is now not fully automatic it is probably more friendly to the user.

- Stop word filtering and stemming:
  Although they were not applied in these experiments, in the future it might be necessary from a practical point of view. Currently the test and training sets are small, so memory consumption at the moment is not a problem. However as the data grows the word vector lengths should be reduced by filtering out words which are not expected to contribute to the classification, the so called stop words.

- Experiments with n-grams:
  In the case of JRIP, only single words were represented as a value. There are no real bigrams or trigrams. JRIP might be able to construct a conjunction of several words which must be present in the sentence, but it does not generate rules to specify in what order they must be in or how big the distance (in words) is between them.

- Other machine learning algorithms:
  BoosTexter and JRIP are only some examples of the many machine learning algorithms out there. It was difficult to assess beforehand which algorithm will perform well, so these two were selected because of their attractive properties, namely being widely published, and relatively easy to understand. They might be not the best, as there are more algorithms out there with similar properties, but which were not tested.

Bibliography

[Boersma01] - P. Boersma - Praat, a system for doing phonetics by computer - Glot International Vol 5, Issue 9/10, 2001.

[Boersma05] - P. Boersma, D. Weenink - Praat: doing phonetics by computer - (Version 4.3.33) [Computer program], URL: http://www.praat.org/, 2005.

[Carpenter98] - B. Carpenter, J. Chu-Carrol - Natural language call routing: a robust self-organizing approach - Proceedings of the 5th International Conference of Spoken Language, 1998.

[Chollet96] - G. Chollet, J. Cochard, A. Constantinescu, C. Jaboulet, P. Langlais, Swiss French PolyPhone and PolyVar: telephone speech databases to model inter- and intra-speaker variability - IDIAP-RR 96, 1996.

[ChuCarroll99] - J. Chu-Carrol, B. Carpenter - Vector-based Natural Language Call Routing - Computational Linguistics, 1999.

[Cohen95] - W. Cohen - Fast Effective Rule Induction - International Conference on Machine Learning, 1995.

[Cohen97] - P. Cohen, S. Dharanipragada, J. Gros, M. Monkowski, C. Neti, S. Roukos, T. Ward - Towards a universal speech recognizer for multiple languages - Proceedings of ASRU 1997.

[Content90] - A. Content, P. Mousty, M. Radeau - BRULEX, Une base de donnes lexicales informatise our le franais crit et parl - L'Anne Pschychologique, 1990.

[DiFabbrizio02] - G. Di Fabbrizio, D. Dutton, N. Gupta, B. Hollister, M. Rahim, G. Riccardi, R. Schapire, J. Schroeter - AT&T help desk - 7th International Conference on Spoken Language Processing, 2002.

[Fischer03] - V. Fischer, E. Janke, S.Kunzmann - Recent progress in the decoding of non-native speech with multilingual acoustic models - Eurospeech 2003.

[Freund97] - Y. Freund, R. Schapire - A decision-theoretic generalization of on-line learning and an application to boosting - Journal of Computer and System Sciences, 1997.

[IPA96] - International Phonetic Association - Reproduction of The International Phonetic Alphabet (Revised to 1993, Updated 1996) - URL: http://www2.arts.gla.ac.uk/IPA/ipa.html.

[Garofolo93] - J. Garofolo, L. Lamel, W. Fisher, J. Fiscus, D. Pallett, N. Dahlgren - The DARPA TIMIT Acoustic-Phonetic Continuous Speech Corpus CDROM, 1993.

[Gaustad02] - T. Gaustad, G. Bouma - Accurate Stemming of Dutch for Text Classification Alfa - LANGUAGE AND COMPUTERS, 2002.

[Gustafson95] - J. Gustafson - Transcribing names with foreign origin in the ONOMOSTICA project - Proc. Int. Congress on Phonetics, 1995.

[Keating98] - P. Keating, Word-level phonetic variation in large speech corpora - ZAS Papers in Linguistics 11, 1998.

[Köhler98] - J. Köhler - Language adaptation of multilingual phone models for vocabulary independent speech recognition tasks - ICASSP 1998.

[Köhler01] - J. Köhler - Multilingual phone models for vocabulary-independent speech recognition tasks - Speech Communication 2001.

[Kunzmann04] - S. Kunzmann, V. Fischer, J. Gonzalez, O. Emam, C. Günther, E. Janke - Multilingual Acoustic Models for Speech Recognition and Synthesis - ICCASP 2004

[Lee89] - K. Lee - Automatic Speech Recognition: The Development of the SPHINX SYSTEM - Boston: Kluwer Academic Publishers, 1989.

[Markowitz96] - J. Markowitz - Using Speech Recognition: A Guide for Application Developers, Prentice Hall, ISBN 013186321-5, 1995.

[McCallum96] - A. McCallum - Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering - URL: http://www.cs.cmu.edu/ mccallum/bow, 1996.

[denOs95] - E. den Os, T. Boogaart, L. Boves, E. Klabbers - The Dutch Polyphone corpus - Eurospeech 1995.

[Pellom01] - B. Pellom - SONIC: The University of Colorado Continuous Speech Recognizer - University of Colorado Technical report #TR-CSLR-2001-01, 2001

[Pellom03] - B. Pellom, K. Hacioglu - Recent Improvements in the CU SONIC ASR System for Noisy Speech: The SPINE Task - ICASSP 2003.

[Rabiner89] - L. Rabiner - A tutorial on hidden markov models and selected applications - Proceedings of the IEEE, 1989.

[Rochery02] - M. Rochery, R. Schapire, M. Rahim, N. Gupta, G. Riccardi, S. Bangalore, H. Alshawi, S. Douglas - Combining prior knowledge and boosting for call classification in spoken language dialogue - ICASSP 2002.

[Schapire00] - R. Schapire, Y. Singer - BoosTexter: A boosting-based system for text categorization - Machine Learning Vol 39 Issue 2/3, 2000.

[Schultz01] - T. Schultz, A. Waibel - Language independent and language adaptive acoustic modeling for speech recognition - Speech Communication 2001.

[Schultz98a] - T. Schultz, A. Waibel - Language Independent and Language Adaptive Large Vocabulary Speech Recognition - ICSLP 1998.

[Schultz98b] - T. Schultz, A. Waibel - Multilingual and Crosslingual speech recognition - Proc. DARPA Workshop on Broadcast News Transcription and Understanding, 1998.

[SpeechPearlManual] - Philips Dialog Systems - The Philips SpeechPearl 2000 manual, 2000.

[Stemmer01] - G. Stemmer, E. Nöth, H. Niemann - Acoustic Modeling of Foreign Words in a German Speech Recognition System - Eurospeech 2001.

[Sturm00] - J. Sturm, H. Kamperman, L. Boves, E. den Os - Impact of speaking style and speaking task on acoustic models - Proc. ICSLP, 2000.

[Uebler01] - U. Uebler - Multilingual speech recognition in seven languages - Speech Communication 2001.

[Uebler99] - U. Uebler, M. Boros - Recognition of non-native German speech with multilingual recognizers - Eurospeech 1999.

[Walker03] - B. Walker, B. Lackey, J. Muller, P. Schone - Language-reconfigurable universal phone recognition - Eurospeech 2003.

[Wells97a] - J. Wells - SAMPA computer readable phonetic alphabet - University College London, URL: http://www.phon.ucl.ac.uk/home/sampa/, 1997.

[Wells97b] - J. Wells - Computer Coding the IPA: a proposed extension of SAMPA - University College London, URL: http://www.phon.ucl.ac.uk/home/sampa/x-sampa.htm, 1997.

[Wang03] - Z. Wang, T. Schultz, A. Waibel - Comparison of acoustic model adaptation techniques on non-native speech - ICASSP 2003.

[Weng97] - F. Weng, H. Bratt, L. Neumeyer, A. Stolcke, A study of multilingual speech recognition, Eurospeech 1997.

[Wester02] - M. Wester, Pronunciation Variation Modeling for Dutch Automatic Speech Recognition - PhD thesis, University of Nijmegen, 2002.

[Witten05] - I. Witten, E. Frank - Data Mining: Practical machine learning tools and techniques - 2nd Edition, Morgan Kaufmann, San Francisco, 2005.

Appendix A

Glossary

Acoustic model	A model which has information about how phonemes are realized by in an audio signal
Agent	A person who will handle an incoming call in a call center
ASR	Automated speech recognition
Corpus	A collection of speech recordings designed for a training or testing a speech recognition system
HMM	Hidden Markov Model
IPA	International Phonetic Association
IVR system	Interactive Voice Response system, used to describe a telephone system which allows users to browse spoken menus by using touchtones.
SAMPA	Speech Assessment Methods Phonetic Alphabet
Language model	A model which has information about allowed transitions between words
Lexicon	A dictionary containing phonemic transcriptions of words
Machine Learning	Methods and algorithms whereby a system can automatically learn characteristics from training data which can in turn be used to predict, recognize or classify new data
Phone	A small acoustic unit independent of the language
Phoneme	The smallest unit in speech leading to difference in a word within a language. A phoneme can be realized by different phones

Appendix B

Phoneme counts

Table B.1: Phonemes per corpus

IPA Symbol	X-SAMPA Symbol	Dutch Polyphone	TIMIT	Swiss French Polyphone	Total	Example
ɨ	1		3118		3118	debit(EN)
ø	2			1559	1559	deux(FR)
øː	2:	512			512	deur(NL)
ɜ˞	3`		1069		1069	furs(EN)
ɬ	5	6590			6590	hal(NL)
œ	9			1787	1787	neuf(FR)
œ̃	9~			1100	1100	brun(FR)
œy	9y	1854			1854	huis(NL)
ə	@	31631	6286	15408	53325	gemakkelijk(NL)
ə˞	@`			2111	2111	corner(EN)
ɑ	A	12912		2498	15410	pat(NL)
ɑ̃	A~			11949	11949	vent(FR)
ɑː	A:		2160		2160	pot(EN)
ɑu	Au	1392			1392	goud(NL)
ð	D		2738		2738	this(EN)
ɛ	E	11145	2738	18650	32533	pet(NL)
ɛ̃	E~			3220	3220	vin(FR)
ɛː	E:	61			61	créme(NL)
ɛi	Ei	4775			4775	fijn(NL)
ɣ	G	3284			3284	goed(NL)
ɥ	H			1514	1514	juin(FR)
ɪ	I	8089	6176		14265	pit(NL)
ɲ	J			387	387	oignon(FR)
ŋ	N	4421	1250	11	5682	bang(NL)
ɔ	O	7337	2295	7578	17210	pot(NL)
ɔ̃	O~			7102	7102	bon(FR)
ɔː	O:	14			14	roze(NL)
ɔɪ	OI		301		301	noise(EN)
ʁ	R			25642	25642	rond(FR)
ʃ	S	633	1242	1520	3395	show(NL)
θ	T		599		599	thin(EN)
ʊ	U		781		781	put(EN)
ʌ	V		1905		1905	cut(EN)

Continued on Next Page…

IPA Symbol	X-SAMPA Symbol	Dutch Polyphone	TIMIT	Swiss French Polyphone	Total	Example
ʏ	Y	3584			3584	put(NL)
ʒ	Z	209	83	3618	3910	bagage(NL)
a	a			20108	20108	patte(FR)
aː	a:	12407			12407	naam(NL)
aɪ	aI		2044		2044	rise(EN)
aʊ	aU		799		799	rouse(EN)
b	b	7252	2489	3224	12965	bak(NL)
d	d	12484	5052	15301	32837	dak(NL)
dʒ	dZ		1048		1048	gin(EN)
e	e			14193	14193	ses(FR)
eː	e:	8172			8172	veer(NL)
eɪ	eI		2180		2180	raise(EN)
f	f	3432	2213	4631	10276	fel(NL)
ɡ	g	191	1471	2069	3731	goal(NL)
h	h	5215	1989		7204	hand(NL)
i	i	6276		17017	23293	vier(NL)
iː	i:		4415		4415	cease(EN)
j	j	2534	1133	6181	9848	ja(NL)
k	k	12405	4535	12203	29143	kap(NL)
l	l	10404	4614	20410	35428	land(NL)
l̩	l=		911		911	bottle(EN)
m	m	11269	3552	9116	23937	met(NL)
m̩	m=		52		52	bottom(EN)
n	n	24036	7416	8867	40319	net(NL)
n̩	n=		248		248	button(EN)
o	o			3503	3503	gros(FR)
oː	o:	8087			8087	voor(NL)
oʊ	oU		1736		1736	nose(EN)
p	p	6603	2970	11516	21089	pak(NL)
r	r	13719			13719	rand(NL)
ɹ	r\	16301	5874		22175	tor(NL)
s	s	20587	6144	21590	48321	sein(NL)
t	t	26210	7983	18641	52834	ttak(NL)
tʃ	tS		798		798	chin(EN)
u	u	2604		5070	7674	voer(NL)
uː	u:		2503		2503	lose(EN)
v	v	7656	2074	6160	15890	vel(NL)
w	w	5893	2186	2403	10482	wit(NL)
x	x	8692			8692	toch(NL)
y	y	1447		6884	8331	vuur(NL)
z	z	3485	4210	3142	10837	zijn(NL)
æ	{		3703		3703	pat(EN)
	SIL	90470	7725	18156	116351	silence
	#n#	3822			3822	background noises
	#s#	3015			3015	mouth noises

Appendix C

Source code listing

C.1 English foreign words

C.1.1 mux-transcription-hypothesis.pl

Multiplexes a transcription file and SONIC batch recognition file together.

```
#!/usr/local/bin/perl5


(@ARGV == 3) ||
        die "usage: (perl) mux_transcription_hypothesis.pl transcription hypothesis outfile\n";

open(TRANSFILE,"<@ARGV[0]") ||
                die "error: could not open @ARGV[0]\n";

open(HYPFILE,"<@ARGV[1]") ||
                die "error: could not open @ARGV[1]\n";

open(OUTFILE,">@ARGV[2]") ||
                die "error: could not open @ARGV[2]\n";


$line = 0;
$file_count = 0;
while ($trline = <TRANSFILE> and $hypline = <HYPFILE>) {
        $line++;

        $trline =~ s/\n//;
        $hypline =~ s/\n//;

        $trline =~ /^(.*)\s*\((\d+)\)$/;
        $transcription = $1;
        $transnum = int($2);

        $hypline =~ /^(.*)\s*\((\d+)\s*(\S*)\)$/;
        $hypothesis = $1;
        $hypnum = int($2);


        print "$hypnum $transnum\n";
```

```
        if (hypnum == transnum) {
                print OUTFILE "$hypnum;; $transcription;; $hypothesis ;; $3\n";
        }
        else {
                die("mismatch");
        }

}

close(INFILE);
```

C.1.2 analyze-output.pl

Analyzes the multiplexed file and gives statistics

```
#!/usr/local/bin/perl5

(@ARGV == 1) || die "usage: (perl) analyze_output.pl \n";

open(INFILE,"<@ARGV[0]") ||
                die "error: could not open @ARGV[0]\n";


$line = 0;
$correct = 0;
$wrong = 0;
$rejected = 0;
$correctfirst = 0;
$path_threshold = -99999;
$false_rejected = 0;

while ($inline = <INFILE>) {
        $inline =~ s/\n//;

        if ($inline =~ /(.*);;(.*);;(.*);;(.*)/) {
                $number = $1;
                $transcription = $2;
                $hypothesis = $3;
                $pathscore = $4;

                #clear whitespace left and right
                $hypothesis =~ s/^\s*//;
                $transcription =~ s/^\s*//;
                $hypothesis =~ s/\s*$//;
                $transcription =~ s/\s*$//;

                if ($pathscore < $path_threshold){
                        $rejected++;
                        if ($transcription eq $hypothesis){
                                $false_rejected++;
                        }
                }
                elsif ($transcription eq $hypothesis){
                                $correct++;
                }
                elsif ($hypothesis eq ""){
```

```
                        $rejected++;
                }
                else {
                        $wrong++;
                        @words = split(/\s+/,$hypothesis);
                        if (@words[0] eq $transcription) {
                                $correctfirst++;
                        }

                }

        }
        $line ++;
}

printf "Total $line, correct $correct, wrong $wrong, rejected $rejected
        (threshold=$path_threshold), wrong but first word correct $correctfirst.\n";
$accepted = $line - $rejected;
$acceptedcorrect = 100*($correct/$accepted);
$acceptedcorrectfirst = 100*($correctfirst/$accepted);

$correct *= 100/$line;
$wrong *= 100/$line;
$rejected *= 100/$line;


$correctfirst *= 100/$line;
$postcorrect = $correctfirst + $correct;
printf "Correct %.2f%%, wrong %.2f%%, rejected %.2f%%, false rejected %.2f%,
        wrong but correct with filter %.2f%%.\n", $correct, $wrong, $rejected,
         100*$false_rejected/$line, $correctfirst;

print "\n";

print "Total correct with postprocess (absolute) = $postcorrect%\n";

print "\n";

print "Correct from accepted $accepted = $acceptedcorrect%\n";

$acceptedcorrectfirst += $acceptedcorrect;
print "Correct from accepted with filter $accepted = $acceptedcorrectfirst%\n";


close(INFILE);
```

C.1.3 filter-wrong.pl

Filters the multiplexed file so that instances which were recognized wrongly remain.

```
#!/usr/local/bin/perl5



(@ARGV == 3) || die "usage: (perl) filter_wrong.pl <testoutput> <output file>\n";

open(INFILE,"<@ARGV[0]") ||
```

```
                die "error: could not open @ARGV[0]\n";

open(INFILE2,"<@ARGV[1]") ||
                die "error: could not open @ARGV[1]\n";


open(OUTFILE,">@ARGV[2]") ||
                die "error: could not open @ARGV[2]\n";


$line = 0;
$correct = 0;
$wrong = 0;
$rejected = 0;
$correctfirst = 0;
$file;

while ($inline = <INFILE> and $file = <INFILE2>) {
        $inline =~ s/\n//;

        $file =~ s/\d+\s*$//;
        $file =~ s/\.lin/\.wav/;

        if ($inline =~ /(.*);;(.*);;(.*);;(.*)/) {
                $number = $1;
                $transcription = $2;
                $hypothesis = $3;

                #clear whitespace left and right
                $hypothesis =~ s/^\s*//;
                $transcription =~ s/^\s*//;
                $hypothesis =~ s/\s*$//;
                $transcription =~ s/\s*$//;

                if ($transcription eq $hypothesis){
                        $correct++;
                }
                else {
                        print OUTFILE "$inline ;; $file\n" ;

                }

        }
        $line ++;
}

close(INFILE);
```

C.2 Call classification

C.2.1 dialogs-to-boosttexter.pl

Extracts the transcriptions from a list of SpeechPearl 2000 dialog files and converts them to BoosTexter format.

```
#!/usr/local/bin/perl5
use strict;
```

```
my $inline;

(@ARGV == 2) || die "usage: <list of dialog files> <output file base name>\n";

open(INFILE,"<@ARGV[0]") ||
                die "error: could not open @ARGV[0]\n";

open(OUTFILE,">@ARGV[1]") ||
                die "error: could not open @ARGV[1]\n";



while ($inline = <INFILE>) {
        #strip newline
        $inline =~ s/\n//;



        if ($inline     =~ /(.*)info\.dlg/)     {
                my $category;
                my $transcribed;
                my $file;
                my $language;
                my $path = $1;
                my $dlgline;
                my $exclusion;

                #print "$1\n";
                open(DLGFILE,"<$inline");
                while($dlgline = <DLGFILE> ) {

                        $dlgline =~ s/\n//;


                        if ($dlgline =~ /\$Attribute\$ Exclusion=(.*)/){
                                $exclusion = $1;
                                #print "$1\n";
                        }

                        if ($dlgline =~ /language=(.*)/){
                                $language = $1;
                                #print "$1\n";
                        }
                        if ($dlgline =~ /\$Attribute\$ Category=(.*)/){
                                $category = $1;
                                #print "$1\n";
                        }
                        if ($dlgline =~ /\$Utterance\$ \(transcribed\) (.*)/){
                                $file = $1;
                                #print "$1\n";
                        }
                        if ($dlgline =~ /transcribed=(.*)/){
                                $transcribed = $1;
                                $transcribed =~ s/,/ /g;
                                $transcribed =~ s/\s+/ /g;

                                #print "$1\n";
```

```
                        }
                }

                if ($language eq "xxx_xxxx" and $category and $transcribed and $file
                        and not $category eq "janee")
                {
                        $transcribed =~ tr/A-Z/a-z/;
                        $transcribed =~ s/\\//g;
                        $transcribed =~ s/\[/ \[/g;
                        $transcribed =~ s/\]/\] /g;
                        $transcribed =~ s/\'/ /g;
                        $transcribed =~ s/\./ /g;
                        $transcribed =~ s/,/ /g;
                        $transcribed =~ s/\?/ /g;
                        $transcribed =~ s/\s+/ /g;


                        print OUTFILE "$transcribed, $category.\n";
                }

                close(DLGFILE);
        }
        else{
                die "fout in invoer file";
        }

}

close(INFILE);
```

C.2.2 dialogs-to-weka.pl

Extracts the transcriptions from a list of SpeechPearl 2000 dialog files and converts them to WEKA .arff format.

```
#!/usr/local/bin/perl5
use strict;

my $inline;

(@ARGV == 2) || die "usage: <list of dialog files> <output file>\n";

open(INFILE,"<@ARGV[0]") ||
                die "error: could not open @ARGV[0]\n";

open(OUTFILE,">@ARGV[1]") ||
                die "error: could not open @ARGV[1]\n";


print OUTFILE "\@relation 'bank'\n";
print OUTFILE "\@attribute transcriptie string\n";
print OUTFILE "\@attribute categorie {'rekeningnummer','verzekeringen',
        'saldo','bij_afschrijvingen','overboeken','wijzigen_toegangscode',
        'beurstrends','aandelenopinies','aandelen_order','pas_blokkeren',
        'pas_aanvragen','wijzigen_adres','opheffen_rekening','incasso_storneren',
        'automatisch_sparen','krediet_limiet','nieuwe_lening','spoedoverboeking',
        'creditcard','openingstijden','afspraak','internetbankieren',
        'x_xxxxxxxxx','inloggen','klachten','suggesties','opgeven_effectenorder',
```

```
        'opgeven_optieorder','levensverzekering','pensioenverzekering',
        'pensioenvraag','gouden_handdruk','hypotheken','schade','autoverzekering',
        'inboedelverzekering','woonhuis','rechtsbijstand','aansprakelijkheid',
        'reisverzekering','andere_categorie','maximaal_leenbedrag',
        'aanvragen_internetbankieren','betaling_geweigerd','contract_internetbankieren'}\n";
print OUTFILE "\@data\n";


while ($inline = <INFILE>) {
        #strip newline
        $inline =~ s/\n//;



        if ($inline     =~ /(.*)info\.dlg/)     {
                my $category;
                my $transcribed;
                my $file;
                my $language;
                my $path = $1;
                my $dlgline;
                my $exclusion;

                #print "$1\n";
                open(DLGFILE,"<$inline");
                while($dlgline = <DLGFILE> ) {

                        $dlgline =~ s/\n//;


                        if ($dlgline =~ /\$Attribute\$ Exclusion=(.*)/){
                                $exclusion = $1;
                                #print "$1\n";
                        }

                        if ($dlgline =~ /language=(.*)/){
                                $language = $1;
                                #print "$1\n";
                        }
                        if ($dlgline =~ /\$Attribute\$ Category=(.*)/){
                                $category = $1;
                                #print "$1\n";
                        }
                        if ($dlgline =~ /\$Utterance\$ \(transcribed\) (.*)/){
                                $file = $1;
                                #print "$1\n";
                        }
                        if ($dlgline =~ /transcribed=(.*)/){
                                $transcribed = $1;
                                $transcribed =~ s/,/ /g;
                                $transcribed =~ s/\s+/ /g;

                                #print "$1\n";
                        }
                }

                if ($language eq "xxx_xxxx" and $category and $transcribed and $file
                        and not $category eq "janee")
```

```
                {

                        $transcribed =~ tr/A-Z/a-z/;
                        $transcribed =~ s/\\//g;
                        $transcribed =~ s/\[/ \[/g;
                        $transcribed =~ s/\]/\] /g;
                        $transcribed =~ s/\'/ /g;
                        $transcribed =~ s/\./ /g;
                        $transcribed =~ s/,/ /g;
                        $transcribed =~ s/\?/ /g;
                        $transcribed =~ s/\s+/ /g;

                        print OUTFILE "'$transcribed', '$category'\n";
                }

                close(DLGFILE);
        }
        else{
                die "fout in invoer file";
        }

}

close(INFILE);
```

C.2.3 SP2000-recognized-to-scored-boostexter.pl

Converts SpeechPearl 2000 recognized text to scored BoosTexter text.

```
#!/usr/local/bin/perl5

## author: Hans Jongebloed, modified by Derek Liauw
## TNO Telecom - DUTCHEAR

## Datum: 23 maart 2005

## parses the result output file and generates an item.dat file with the
## 1-best recognition result (incl. confidence per word).

use strict;

(@ARGV == 2) || die "usage: (perl) SP2000-recognized-to-scored-boostexter.pl input output\n";


open(INFILE,"@ARGV[0].txt") ||
                die "error: could not open @ARGV[0].txt\n";


open(OUTFILE,">@ARGV[1]") ||
                die "error: could not open @ARGV[1]\n";




#print OUTFILE "file;soundfile;spoken;reco;\n";
```

```
my $line = 0;
my $file = 0;
my $soundfile;
my $spoken;
my $reco;
my $inline;
my $conf;
my $keyword;

while ($inline = <INFILE>) {
        $line++;

#       if ($file > 50) {last;}

        $inline =~ s/\n//;


        if ($inline =~ /Sentence: (\d+)/) {
                ## verwerken resultaat vorige wav file:
                if ($file > 0) {
                        $reco =~ s/\s*$//;
                        $reco =~ s/\<.+\>//g;
                    #print OUTFILE "$file;$soundfile;$spoken;$reco;\n";
                        print OUTFILE "$reco,$spoken.\n";
                }


                ## init variabelen voor volgende wav file:
            $file = $1;
            $spoken = "";
            $reco = "";
            $soundfile = "";
        }
        elsif ($inline =~ /Spoken: (.*)$/) {
            $spoken = $1;
        }
        elsif ($inline =~ /SoundFile: (.*)$/) {
            $soundfile = $1;
        }
        #elsif ($inline =~ /Recognized: (.*)$/) {
        #    $reco = $1;
        #    $reco =~ s/\#PAUSE\#//g;
        #    $reco =~ s/\<.*?\>//g;
        #    $reco =~ s/\s+$//;
        #    $reco =~ s/^\s+//;
        #}
        elsif ($inline =~ /Detailed errors:/) {next;}
        #elsif ($inline =~ /Ins  filler.*\s*(\d\.\d\d\d)/) {
        #       $conf = $1;
        #       $conf =~ s/0\.//;
        #       $conf =~ s/\.//;
        #}
        elsif ($inline =~ /Ins  (\S*)\s*(\d\.\d\d\d)/) {
                $keyword = $1;
                $conf = $2;
                $conf =~ s/0\.//;
                $conf =~ s/\.//;
```

```
                $reco .= "$keyword $conf ";
        }
        elsif ($inline =~ /Sub  (\S*)\s*(\d\.\d\d\d)/) {
                $keyword = $1;
                $conf = $2;
                $conf =~ s/0\.//;
                $conf =~ s/\.//;
                $reco .= "$keyword $conf ";
        }
        elsif ($inline =~ /Del  .*/) {
        #       $keyword = $1;
        #       $conf = $2;
        #       $conf =~ s/0\.//;
        #       $conf =~ s/\.//;
        #       $reco .= "$keyword($conf) ";
                #print "$inline\n";
        }
        elsif ($inline =~ /Cor  (\S*)\s*(\d\.\d\d\d)/) {
                $keyword = $1;
                $conf = $2;
                $conf =~ s/0\.//;
                $conf =~ s/\.//;
                $reco .= "$keyword $conf ";
        }


}


if ($file > 0) {
        $reco =~ s/\s*$//;
        $reco =~ s/\<.+\>//g;
        print OUTFILE "$reco,$spoken.\n";
}


close(INFILE);

close(OUTFILE);
```

C.2.4 SP2000-recognized-to-scored-weka.pl

Converts SpeechPearl 2000 recognized texst to a word vectors suitable for use with WEKA.

```
#!/usr/local/bin/perl5

## author: Hans Jongebloed, modified by Derek Liauw
## TNO Telecom - DUTCHEAR

## Datum: 23 maart 2005

## parses the result output file and generates an item.dat file with the
## 1-best recognition result (incl. confidence per word).

use strict;
```

```
(@ARGV == 2) || die "usage: (perl) SP2000-recognized-to-scored-boostexter.pl input output\n";


open(INFILE,"@ARGV[0].txt") ||
                die "error: could not open @ARGV[0].txt\n";


open(OUTFILE,">@ARGV[1]") ||
                die "error: could not open @ARGV[1]\n";




#print OUTFILE "file;soundfile;spoken;reco;\n";


my $line = 0;
my $file = 0;
my $soundfile;
my $spoken;
my $reco;
my $inline;
my $conf;
my $keyword;

while ($inline = <INFILE>) {
        $line++;

#       if ($file > 50) {last;}

        $inline =~ s/\n//;


        if ($inline =~ /Sentence: (\d+)/) {
                ## verwerken resultaat vorige wav file:
                if ($file > 0) {
                        $reco =~ s/\s*$//;
                        $reco =~ s/\<.+\>//g;
                    #print OUTFILE "$file;$soundfile;$spoken;$reco;\n";
                        print OUTFILE "$reco,$spoken.\n";
                }


                ## init variabelen voor volgende wav file:
            $file = $1;
            $spoken = "";
            $reco = "";
            $soundfile = "";
        }
        elsif ($inline =~ /Spoken: (.*)$/) {
            $spoken = $1;
        }
        elsif ($inline =~ /SoundFile: (.*)$/) {
            $soundfile = $1;
        }
        #elsif ($inline =~ /Recognized: (.*)$/) {
```

```
        #    $reco = $1;
        #    $reco =~ s/\#PAUSE\#//g;
        #    $reco =~ s/\<.*?\>//g;
        #    $reco =~ s/\s+$//;
        #    $reco =~ s/^\s+//;
        #}
        elsif ($inline =~ /Detailed errors:/) {next;}
        #elsif ($inline =~ /Ins  filler.*\s*(\d\.\d\d\d)/) {
        #       $conf = $1;
        #       $conf =~ s/0\.//;
        #       $conf =~ s/\.//;
        #}
        elsif ($inline =~ /Ins  (\S*)\s*(\d\.\d\d\d)/) {
                $keyword = $1;
                $conf = $2;
                $conf =~ s/0\.//;
                $conf =~ s/\.//;
                $reco .= "$keyword $conf ";
        }
        elsif ($inline =~ /Sub  (\S*)\s*(\d\.\d\d\d)/) {
                $keyword = $1;
                $conf = $2;
                $conf =~ s/0\.//;
                $conf =~ s/\.//;
                $reco .= "$keyword $conf ";
        }
        elsif ($inline =~ /Del  .*/) {
        #       $keyword = $1;
        #       $conf = $2;
        #       $conf =~ s/0\.//;
        #       $conf =~ s/\.//;
        #       $reco .= "$keyword($conf) ";
                #print "$inline\n";
        }
        elsif ($inline =~ /Cor  (\S*)\s*(\d\.\d\d\d)/) {
                $keyword = $1;
                $conf = $2;
                $conf =~ s/0\.//;
                $conf =~ s/\.//;
                $reco .= "$keyword $conf ";
        }


}


if ($file > 0) {
        $reco =~ s/\s*$//;
        $reco =~ s/\<.+\>//g;
        print OUTFILE "$reco,$spoken.\n";
}


close(INFILE);

close(OUTFILE);
```

C.2.5 jrip-rules-to-perl-coverter.pl

Converts WEKA JRIP output to a standalone perl classifier.

```
#!/usr/local/bin/perl5
use strict;

my $inline;
my $firstrule = 1;
my %keywords;

(@ARGV == 3) || die "Generates a standalone perl script from JRIP rules.
                \n usage: <list of rules> <output classifier> <output list of keywords>\n";

open(INFILE,"<@ARGV[0]") ||
                die "error: could not open @ARGV[0]\n";

open(OUTFILE,">@ARGV[1]") ||
                die "error: could not open @ARGV[1]\n";

open(OUTFILE2,">@ARGV[2]") ||
                die "error: could not open @ARGV[2]\n";



print OUTFILE "#!/usr/local/bin/perl5\n";
print OUTFILE "#Autogegenereerde file\n";
print OUTFILE "use strict;\n";
print OUTFILE "\n";
print OUTFILE "my \$class;\n";
print OUTFILE "my \$inputstring;\n";
print OUTFILE "my %wordvector;\n";
print OUTFILE "\$inputstring = join(\" \", \@ARGV);\n";
print OUTFILE "print \"\$inputstring\\n\";\n";
print OUTFILE "\n";
print OUTFILE "foreach my \$word (\@ARGV){\n\t\$wordvector{\$word} = 1;\n};\n";
print OUTFILE "\n";

while ($inline = <INFILE>) {
        #strip newline
        $inline =~ s/\n//;
        if ($inline =~ /(.*)=> .*=\s*(\S+)\s*\(.*\)/){

                my $preconditions = $1;
                my $class = $2;

                my @pres = split(" and ", $preconditions);
                my $newpreconditions;

                $preconditions =~ s/^\s*//;
                $preconditions =~ s/\s*$//;

                if ($preconditions){
                        foreach my $p (@pres) {
                                $p =~ /\((\S*).*\)/;
                                my $word = $1;
```

```
                                $word =~ s/\[/\\\[/;
                                $word =~ s/\]/\\\]/;
                                print "$p\n";
                                $keywords{$word}++;
                                $p =~ s/$word/\$wordvector{\"$word\"}/;
                                if (not $newpreconditions){
                                        $newpreconditions = $p;
                                }
                                else{
                                        $newpreconditions = $newpreconditions." and ".$p;
                                }

                        }
                }

                if ($firstrule) {
                        print OUTFILE "if ($newpreconditions) { \$class = \"$class\" }\n";
                        $firstrule = 0;
                }
                elsif(not $preconditions)
                {
                        print OUTFILE "else { \$class = \"$class\" }\n";
                        $firstrule = 0;

                }
                else
                {
                        print OUTFILE "elsif ($newpreconditions) { \$class = \"$class\" }\n";
                        $firstrule = 0;

                }


        }
        else{
                die "Unknown format for line\n";
        }
}

print OUTFILE "\n";
print OUTFILE "print \"\$class\\n\";\n";

for my $keyword (keys %keywords) {
        print OUTFILE2 "$keyword\n";
}
```

C.2.6 evaluate-classifier-arff.pl

Evaluates a file in WEKA .arff format using a perl classifier.

```
#!/usr/local/bin/perl5
use strict;

my $inline;
my $perlpath='c:\perl\bin\perl';

(@ARGV == 2) || die "Evaluates a classifer.\n usage: <classifier> <arff to evaluate>\n";
```

```
open(INFILE2,"<@ARGV[1]") ||
                die "error: could not open @ARGV[1]\n";

my %tp;           #geclassificeerd als $key en goed
my %fp;           #geclassificeerd als $key maar fout
#my %tn;           #de juiste klasse is niet $key en is ook zo geclassificeerd
                // tn kan afgeleid worden uit andere waarden
my %fn;               #juiste klasse is $key maar fout geclassificeerd


my %distribution; #aantal items van elke categorie in de database
my $tests = 0;
my $reject = 0;

#lees tot de @data tag
while($inline = <INFILE2>){
        $inline =~ s/\n$//;
        if ($inline =~ /\@data/)
        {
                last;
        }
}

while($inline = <INFILE2>){
        $inline =~ s/\n$//;

        if ($inline =~ /\s*\'(.*)\'\s*,\s*\'(.*)\'\s*/){
                my $text = $1;
                my $class = $2;

                print "$text\n";

                my @classoutput = `$perlpath @ARGV[0] $text`;
                print "@classoutput[1]\n";
                my $classified_as = @classoutput[1];

                $class =~ s/^\s*//;
                $class =~ s/\s*$//;
                $classified_as =~ s/^\s*//;
                $classified_as =~ s/\s*$//;

                if ($classified_as eq ""){
                        $reject++;
                        next;
                }

                $distribution{$class}++;

                if ($classified_as eq $class){
                        $tp{'_ALL_'}++;
                        $tp{$class}++;
                }
                else
                {
                        $fn{'_ALL_'}++;
                        $fn{$class}++;
                        $fp{$classified_as}++;
```

```
                }
                $tests++;
        }
        else {
                die("File format error");
        }
}


printf "Precision \t Recall \t F-Measure \t Category\n";
printf "--------- \t ------ \t --------- \t --------\n";
for my $key (keys %tp){
        if ($key eq '_ALL_'){
                next;
        }

        #my $truepositive = 100*$tp{$key}/($tp{$key}+$fp{$key});
        #my $falsepositive = 100*$fp{$key}/($tests-$distribution{$key});

        #my $truenegative = 100*$tp{$key}/($tp{$key}+$fp{$key});
        #my $falsenegative = 100*$fn{$key}/($tests-$distribution{$key});

        my $precision = $tp{$key}/($tp{$key}+$fp{$key});
        my $recall = $tp{$key}/($tp{$key}+$fn{$key});
        my $fmeasure = 2*$precision*$recall / ($recall+$precision);


        printf "%6.2f%% \t %6.2f%% \t %6.2f%% \t $key\n",
                                        $precision*100, $recall*100, $fmeasure*100;
}

my $score = 100*$tp{'_ALL_'}/$tests;
print "Rejected $reject\n";
print "Totaal: $tests, correct = $tp{'_ALL_'}, wrong=$fn{'_ALL_'}, Accuracy = $score%\n";
```

C.2.7 evaluate-classifier-SP2000-recognized.pl

Evaluates recognition output using a perl classifier.

```
#!/usr/local/bin/perl5
use strict;

my $inline;
my $perlpath='c:\perl\bin\perl';

(@ARGV == 3) || die "Evaluates a classifer.\n usage: <classifier>
                                        <reco file to process> <threshold>\n";


open(INFILE2,"<@ARGV[1]") ||
                die "error: could not open @ARGV[1]\n";

my $threshold = @ARGV[2];

my %tp;           #geclassificeerd als $key en goed
my %fp;           #geclassificeerd als $key maar fout
#my %tn;           #de juiste klasse is niet $key en is ook zo geclassificeerd
                 // tn kan afgeleid worden uit andere waarden
```

```
my %fn;               #juiste klasse is $key maar fout geclassificeerd
my $rejected = 0;


my %distribution; #aantal items van elke categorie in de database
my $tests = 0;

$inline = <INFILE2>; #skip eerste regel

while($inline = <INFILE2>){
        $inline =~ s/\n$//;

        if ($inline =~ /(.*);(.*);(.*);(.*);/){
                my $reco = $4;
                my $class = $3;
                my $text;

                $text = "";

                my @keywords = split(/ /, $reco);
                for my $keyword (@keywords)
                {
                        $keyword =~ /(\S+)\((\d+)\)/;
                        my $conf = $2;
                        $keyword = $1;
                        if ($conf >= $threshold){
                                $keyword =~ s/\(\d+\)//g;
                                $text .= "$keyword ";
                        }
                }
                print "$reco\n$text\n";

                $text =~ s/filler//g;
                $text =~ s/#OOV#//g;
                $text =~ s/^\s+//g;
                $text =~ s/\s+$//g;
                if ($text eq "") { print "$class, reject(nietsherkend)\n\n";
                                                        $rejected++; next;}

                $text =~ s/<.+?>//g;

                my @classoutput = `$perlpath @ARGV[0] $text`;
                print "$class, @classoutput[1]\n";
                my $classified_as = @classoutput[1];

                $class =~ s/^\s*//;
                $class =~ s/\s*$//;
                $classified_as =~ s/^\s*//;
                $classified_as =~ s/\s*$//;

                if ($classified_as eq "") { print "$class, reject\n\n"; $rejected++; next;}

                $distribution{$class}++;

                if ($classified_as eq $class){
                        $tp{'_ALL_'}++;
                        $tp{$class}++;
                }
```

```
                else
                {
                        $fn{'_ALL_'}++;
                        $fn{$class}++;
                        $fp{$classified_as}++;
                }
                $tests++;
        }
        else {
                die("File format error");
        }
}


printf "Precision \t Recall \t F-Measure \t Category\n";
printf "--------- \t ------ \t --------- \t --------\n";
for my $key (keys %tp){
        if ($key eq '_ALL_'){
                next;
        }

        #my $truepositive = 100*$tp{$key}/($tp{$key}+$fp{$key});
        #my $falsepositive = 100*$fp{$key}/($tests-$distribution{$key});

        #my $truenegative = 100*$tp{$key}/($tp{$key}+$fp{$key});
        #my $falsenegative = 100*$fn{$key}/($tests-$distribution{$key});

        my $precision = $tp{$key}/($tp{$key}+$fp{$key});
        my $recall = $tp{$key}/($tp{$key}+$fn{$key});
        my $fmeasure = 2*$precision*$recall / ($recall+$precision);


        printf "%6.2f%% \t %6.2f%% \t %6.2f%% \t $key\n",
                        $precision*100, $recall*100, $fmeasure*100;
}

my $score = 100*$tp{'_ALL_'}/$tests;
print "Rejected utterances = $rejected\n";
print "Totaal: $tests, correct = $tp{'_ALL_'}, wrong=$fn{'_ALL_'}, Accuracy = $score%\n";
```

Appendix D

Summary Paper

Page intentionally left blank.
The summary paper starts on the next uneven page!

Page intentionally left blank.
The summary paper starts on the next uneven page!

Topics in Speech Recognition

D. Liauw Kie Fa, BSc.
Delft University of Technology, Dept. of Mediamatics
Dutchear B.V.

June 1, 2006

Abstract

ASR has many open problems. In this work two well-known problems are researched. The first topic deals with the ever growing phenomenon of English words being used in Dutch colloquial speech. This makes it increasingly expectable that speech recognizers in the Netherlands should be able to recognize these English words as well as Dutch. Dutch people speaking English introduces a number of problems, including non-native speech and lack of training data. Thus a multilingual acoustic modeling approach was attempted using training data from widely available Dutch and English corpora. The second topic deals with the realization of a call classifier which is trained using machine learning techniques. Machine learning techniques algorithms in call classification literature such as BoosTexter generally require a large amount of data before satisfying results are obtained, thus making them unsuitable for use in the beginning stages of a project where human expertise is more suitable. By using the RIPPER rule generating algorithm hopefully a system can be made which accepts human knowledge in the early phases of a project, as well as machine generated rules which can be inspected by the expert in later stages of a project as more data becomes available.

1 English Foreign Word Recognition

1.1 Introduction

English words are a common phenomenon in Dutch colloquial speech. Examples of such words are: first and last names, company names, titles of books, movies, CDs, product names etc.

Recognition of English words by a Dutch speech recognizer is not straightforward because English has phonemes which are untypical for Dutch. In addition there is the difficulty that many Dutch people do not have a perfect English pronunciation and have tendencies to borrow the closest Dutch phoneme.

The problem definition for this topic is to find out whether multilingual recognition techniques can be used to improve recognition of non-native English spoken by Dutch people. The focus is specifically on name dialing applications with the goal of being able to handle foreign names and non-native Dutch speakers. The main performance meter of the evaluation is the speech recognizer's performance on single word recognition.

The goal is not to build such a system completely, but to find out whether multilingual techniques are a feasible option for recognition, which is the case if there is an improvement in the recognition of English words spoken by Dutch people using multilingual models at little or no cost for the recognition of normal Dutch utterances.

1.2 Related Work

Various papers have been written on multilingual acoustic models, and for each purpose the best method to use differs. As for phoneme mapping, the most common method used are the phonetic and native approaches, since these are simple and require no speech data to perform. The phonetic approach finds similar phonemes by looking at common phonetic features. The native approach looks at the phonemes non-natives are likely to use when trying to pronounce phonemes not in their native inventory. Data-driven methods to find phoneme mappings are only possible when sufficient data is available.

For simultaneous recognition of languages the multilingual acoustic model is trained on, it is best that information about the language is preserved. [Schultz98a]'s Langtag and [Köhler01]'s IPA-OVL are good examples, but also in [Cohen97] the language information preserving approach is slightly better than not preserving the information. However these techniques are not up to par with the performance of traditional monolingual recognizers.

For multilingual cross-lingual recognition and bootstrapping new languages [Schultz98a] shows that simply sharing training data (Lang-mix) without preserving language information works best, and this works better than performing cross-lingual recognition from only one language. [Uebler01] found similar results regarding cross-language recognition from a mono or multilingual model. [Kunzmann04] shows that by adding more and more training data from other languages step by step gradually improved recognition of Spanish, which was not included in the training.

For non-native recognition the performance is as follows. [Kunzmann04] shows that adding more and more training data from other languages step by step gradually improved recognition of both native and non-native speech. [Fischer03] show that performing pronunciation modeling for phonemes not in the native set in combination with multilingual acoustic outperforms adaptation techniques on an English digit recognition task with speakers from various countries. [Stemmer01] implemented a movie name recognition service for a German audience with many of the titles having English names, thus non-native pronunciations. The mapping of English phonemes to German with a German recognizer performed the worst, and they got the most improvement using knowledge-based merging of phonemes in combination with a multilingual model, followed by data-driven merging of phonemes. [Uebler99] found that it is useful to add speech data of the language which the

speaker is native from to the training in order to recognize non-native speakers of that origin better.

1.3 Methodology

Throughout this part of this work the SONIC speech recognizer [Pellom01] is used. Recognition of both Dutch and English phonemes is required, so it is reasonable to use Dutch and English training data. Swiss French data is also added to the mix, because [Kunzmann04] shows that adding data even not belonging to the languages to be tested might improve performance, as it simply provides more training samples and allophonic variations for the phonemes the languages have in common. So in fact 3 languages were used for training.

As the corpora came from different sources different transcription guidelines were used. To unify them phoneme sets were converted to IPA, sometimes simplified to make the granularity the same, and converted back into a machine readable form, X-SAMPA [Wells97b]. Statistics regarding the used training corpora can be found in Table 1. In this text phonemes are generally described using X-SAMPA symbols, not IPA ones, and are surrounded by slashes, for example /p/. This section gives an overview of all acoustic models trained (Table 2) and describes how they were trained.

Acoustic models were trained initially according to the Langmix acoustic modeling strategy, meaning data is shared for phonemes with the same IPA symbol.

Since much of the training data did not come from the same sources, did not have time aligned phonetic transcriptions but only word-level transcriptions, and a relatively large amount of data was available for Dutch and relatively little for the other training languages, an iterative approach to aligning the training data was chosen, similar to [Walker03].

The iterative procedure ensures that alignments are as consistent as possible across corpora and is as follows:

1. Initialize the multilingual set with a reference corpus C_0 and train a recognizer R_0. Initialize $x = 0$.

2. C_{new} is the new corpus not yet in the reference corpus.

3. Use recognizer R_x to align the data C_{new}. If C_{new} has phonemes which are not in the reference corpus, the aligner is configured to initialize this phoneme with the closest phoneme in C_x.

4. Expand the reference corpus C_x with C_{new}, creating C_{x+1}.

5. Train a new recognizer R_{x+1} on the new reference corpus C_{x+1}. If phonemes from the existing set and new corpus have the same phonetic symbol, they will share the same acoustic model.

6. Increase x by 1.

7. If there are still corpora not in the reference set go to step 2.

8. Else the procedure is finished.

The Dutch training data was chosen as the reference corpus, since the performance of the model trained with this data would be used for baseline testing.

The Dutch training data is taken from Dutch Polyphone [denOs95], but is different from the standard training set. The used training set is a modified one which performs particularly well on short utterances [Sturm00]. The Dutch lexicon was transcribed with the PHICOS phoneme set, an extension of SAMPA [Wells97a] with postvocalic /L/ and /R/. PHICOS to IPA mappings were taken from [SpeechPearlManual] and [Wester02].

The English training data consisted of the default TIMIT [Garofolo93] training set. The audio was filtered with a low-pass of 3.4 kHz and a high-pass of 0.3 kHz and down-sampled to 8 kHz to simulate telephone speech as much as possible. The lexicon provided with TIMIT was converted to IPA using the table found in [Keating98]; some symbols additionally had lengthening marks added.

The Swiss French data consists of a subset of the phonetically rich sentences from Swiss French Polyphone [Chollet96]. No lexicon was readily available for this corpus, so letter to sound rules were trained using the SONIC text to phone tool. The BRULEX lexicon [Content90] was used as training material for the tool. It was not determined how accurate the generated transcriptions are, but viewing the transcriptions empirically did not reveal any errors of significance.

1.4 Tests

Several corpora were used for testing and they are shown in Table 3. The VoiceConnect97 database is a native Dutch language corpus and is used to test the performance on Dutch. The TIMIT database is an American English database and is used to test the English phonemes which were trained with the multilingual acoustic models. The DDAC2000 database is a database consisting of utterances which are similar to our problem description, namely Dutch people attempting to speak English.

The main goal of these tests are to determine whether performance increases on this corpus using multilingual techniques and that any performance decrease on native Dutch is minimal. Additional details about the tests using the corpora are given in the corresponding sections.

1.4.1 VoiceConnec97 test

This test measures the performance of the various models on Dutch only. As it is known in literature, the monolingual recognition performance tends to drop compared to the multilingual model. Since the recognizer must recognize Dutch in addition to the English words, it is essential to measure the model's performance on Dutch.

The results of the performance of the acoustic models are in Table 4.

The performance degrades when more languages are added to the mix, thus making the best model for this test AMO_NL2 which is only trained on Dutch data, followed by the combination of English and Dutch (AMO_NLEN2).

Although adding more languages degrades performance, the performance of the best model with English phonemes (AMO_NL2) remains acceptable compared to the baseline, as

Table 1: Overview of training corpora

Corpus	Language	Utterances	Hours	Type
Dutch Polyphone	Dutch	42101	48.92	Short utterances
TIMIT	American English	3696	3.27	Sentences
Swiss French Polyphone	Swiss French	7961	10.60	Sentences

Table 2: Acoustic model overview

Code	Trained on	Phonemes	Alignments	Comments
AMO_NL2	Dutch	45	2	baseline acoustic model
AMO_NLEN2	Dutch, English	66	2	
AMO_NLEN2q	Dutch, English	66	2	language question
AMO_NLENFR2	Dutch, English, French	78	2	
AMO_NLFR2	Dutch, French	78	2	Eng. phonemes not trained

Table 4: Voiceconnect97 results

Model	Correct
AMO_NL2	93.36%
AMO_NLEN2	92.66%
AMO_NLEN2q	92.72%
AMO_NLENFR2	91.34%
AMO_NLFR2	91.02%

the degradation is fairly small. The 3 language combination (AMO_NLENFR2) performs worse but not unacceptable.

Adding French seems to drop performance the most. This is indicated by the fact that the Dutch-French (AMO_NLFR2) model performs worse than all other models, including the Dutch- English-French combination. This is strange because the first has only 2 languages in the mix, and the latter 3.

Adding the language question (AMO_NLEN2q) improved the recognition performance of the combined Dutch English acoustic model (AMO_NLEN2) only marginally. Lang-tag in literature improves performance of monolingual recognition so this is also expected.

1.4.2 TIMIT test

The TIMIT test set was used to evaluate the native English performance of the acoustic model. If every Dutchman happened to have a perfect (American) English pronunciation, this test would be suitable, but there are some flaws. Because the pronunciation of a non-native speaker can be very off or very good, the results of this experiment are not directly relevant to foreign English word recognition, but it might provide some useful information, since the extra English phonemes are heavily tested. Also a comparison is made to the baseline English recognizer, trained on English data only, to see how much performance drops with Dutch data added. The results for the different acoustic models are in Table 5.

Table 5: TIMIT test results

Model	Correct
AMO_EN	91.8%
AMO_NLEN2	91.1%
AMO_NLEN2q	90.8%
AMO_NLENFR2	91.2%

The best model in this test is the model trained on English data only, the monolingually trained AMO_EN. This is expected because as seen before, adding additional languages decreases performance.

Again the mixed English-Dutch models, but especially AMO_NLEN2, does not perform significantly less than the English only recognizer. This indicates that the English phonemes are recognized well.

The Lang-tag strategy appears not to increase performance of the monolingual English recognition, and performance is in fact less. This might be explained by the fact that there was a lot more Dutch training data used compared to the amount of English data, therefore resulting in a higher bias toward recognition of Dutch phonemes.

Adding French to the multilingual model did not seem to have a conclusive effect, compared to the best English Dutch model.

1.4.3 DDAC2000 test

This test set is the most important one, since it most closely resembles the goals of the experiment: to see if recognition of English foreign words improves with a multilingual model.

A subset of the DDAC2000 [Sturm00] corpus was used for this test. The test set consisted of 1018 audio files, and a vocabulary of 272 items of which a complete utterance is regarded as an item.

The items in this lexicon were often a combination of Dutch and English words, but always at least one English word, therefore the phonetic transcriptions were allowed to have a mix of both Dutch and English phonemes, with English phonemes used whenever a word was English. The initial lexicon was generated using letter to sound rules trained on the American English TIMIT lexicon. This obviously produced a lot of junk when generating transcriptions for Dutch words, so all entries were double checked and fixed manually.

After applying fixes, the resulting lexicon is called LEX_EN. This lexicon represents the phoneme expansion approach. The LEX_EN lexicon was then converted to one with Dutch phonemes only using the mapping in Table 6, resulting in LEX_NL and represents the phoneme mapping approach. The mapping table was constructed such that each English phoneme not present in the Dutch set should have a representation using the Dutch set which matches as close as possible. One way to come up with such a mapping, and which

Table 3: Overview of testing corpora

Corpus	Language	Utterances	Type
VoiceConnect97	Dutch	5300	Short utterances
TIMIT	American English	2604	Sentences
DDAC2000	Dutch mixed with English words	1018	Short utterances

was applied here was by asking the following question: What combination of Dutch phonemes is necessary to produce an acceptable pronunciation for an English word using Dutch text to speech?

Table 6: English to Dutch Phoneme mapping table

Eng.	Dut.	Eng.	Dut.	Eng.	Dut.	Eng.	Dut.
1	I	3`	Y r	@`	@ r\	A:	A
D	d	OI	O j	T	t	U	u
V	O	aI	A j	aU	Au	dZ	d Z
eI	e:	i:	i	l=	@ l	m=	@ m
n=	@ n	oU	o:	tS	t S	u:	u
{	E						

More preliminary tests on AMO_NLEN showed that LEX_NL outperformed LEX_EN. This means that the recognition of Dutch phonemes only (LEX_NL) works better than a Dutch phoneme set expanded with English ones (LEX_NL). However, an error analysis showed that different errors were made in both cases. This suggests that people often use a Dutch-like pronunciation which is recognized well by the LEX_NL lexicon, and some an English-like pronunciation which is recognized well by LEX_EN. To get the best of both worlds, the two lexicons were merged together resulting in LEX_NLEN, which indeed resulted in another performance improvement. This was repeated for all acoustic models which also show this behavior.

The complete results are shown in Table 7.

Table 7: DDAC2000 results ordered by acoustic model

Acoustic model	Lexicon	Correct
AMO_NL2	LEX_NL	91.45%
AMO_NLEN2	LEX_EN	90.96%
AMO_NLEN2	LEX_NL	92.24%
AMO_NLEN2	LEX_NLEN	93.81%
AMO_NLEN2q	LEX_EN	84.87%
AMO_NLEN2q	LEX_NL	92.24%
AMO_NLEN2q	LEX_NLEN	93.71%
AMO_NLENFR2	LEX_EN	87.52%
AMO_NLENFR2	LEX_NL	92.17%
AMO_NLENFR2	LEX_NLEN	93.03%
AMO_NLFR2	LEX_NL	89.88%

As noted earlier, because LEX_NL outperforms LEX_EN on all acoustic models it is reasonable to assume that Dutch people mainly use a "Dutch" way of pronouncing English rather than a more English like way. It also so means that the training of English phonemes is of relatively less importance than the training of Dutch phonemes.

However some utterances recognized well using LEX_NL are recognized poorly when using LEX_EN and vice versa, meaning that extra pronunciation variants with English phonemes might increase recognition rate.

When comparing the performance of LEX_NL on different acoustic models AMO_NLEN2, AMO_NLEN2q and AMO_NLENFR2 outperform AMO_NL2, suggesting that training Dutch phonemes with native English data is beneficiary when recognizing English spoken by Dutch, since only Dutch phonemes were used for recognition. This might be explained by the fact that although some IPA symbols are the same in English and Dutch, the exact realizations and allophones within a language are still different. Training with English data causes the phoneme to be "broadened" with these allophones and also causes increases in the number of triphones.

Adding French to the training set however decreases performance. This can be seen by comparing the performance of the AMO_NLENFR to the other acoustic models. This can be explained by the fact that Dutch phonemes are important for recognition for in test, and that this acoustic model is not so good at it (see VoiceConnect97 test results).

The Lang-tag strategy with language question has minimal impact and sometimes decreases recognition performance. This can be explained by the fact that the Lang-tag question is generally used for recognizing a single language the multilingual model is trained on, but in this case it is a mix of phonemes from different languages which need to be recognized. The Lang-mix strategy is better.

Finally, the best performing combination, LEX_NLEN on AMO_NLEN performs 2.54% better than the baseline performance of LEX_NL on AMO_NL. There are 2 explanations for this increase:

- The acoustic model: Even if using only Dutch phonemes for recognition (LEX_NL), the performance is better with AMO_NLEN2, which is trained additionally on English data than, with AMO_NL2.
- Pronunciation variants made possible by extra phonemes: When using a lexicon with additional entries using English phonemes (LEX_NLEN) the recognition rate improves over using a lexicon with only Dutch phonemes (LEX_NL)

1.5 Conclusion

The VoiceConnect97 tests showed that the multilingual model could be used to recognize Dutch, at the cost of a small performance degradation, but it could not be described as major, as it is only 0.60% when comparing AMO_NLEN2 to AMO_NL2.

When testing on TIMIT the recognition of native English decreased when compared to an English data only trained model, but this test proved that English phonemes could be recognized well by the multilingual model.

Compared to the mapping approach using a pure Dutch acoustic model and a Dutch phoneme only lexicon, performance for English foreign word recognition improved by 0.79%

by simply using a multilingual model in stead of the monolingual model. The increase can mount up to 2.54% when also using a more extensive lexicon, made possible by the extra phonemes of the multilingual set.

So in summary from these tests we can derive that a lot of English spoken by Dutch can be recognized using Dutch phonemes and a Dutch recognizer only by using the mapping approach. However some typical English phonemes are still difficult to recognize and are better recognized by using phoneme models trained on English data. An overall benefit is therefore only possible when a sizable amount of utterances consists of English words, and therefore it is still viable to use a Dutch only recognizer to recognize English foreign words, instead of a multilingual model, since the performance increase is not very much. The analysis of the differences in recognition results when using Dutch and English phonemes to recognize the same words prove that using English data in the training of the recognizer is indeed useful for some specific English phonemes in the test case.

2 Call classification

2.1 Introduction

The idea of a call classifier is an automated system which can understand callers in their own natural language, and based on the first sentence spoken can redirect the call the person who can help him best.

The problem definition for this topic is to find out how to make a good call router using machine learning techniques which also performs well in the initial phase of the system's life. Obviously performance will increase as more data becomes available, but it is the initial performance which is of most interest.

A system must be built that:

1. Is be able to recognize long free speech utterances:
   For this the SpeechPearl 2000 speech recognizer is used which is able to recognize sentences.

2. Works using Machine Learning Techniques:
   A machine learning algorithm called RIPPER, which is known for it's human understandable rules will be tested.

3. Performs comparably with known techniques:
   The BoosTexter algorithm which is based on AdaBoost is used by AT&T in their call classification systems. The performance of RIPPER and BoosTexter will be compared.

4. Is easy to understand, and modifiable by an expert:
   This is required for the maintainability of the system. Also because in the beginning stages with little data available it is likely that rules generated by an expert will perform better than automatically generated rules.

5. Is able to deal with uncertainty:
   Most speech recognizers generate the result with a certain confidence for each recognized word. When the confidence is low there are probably some misrecognized parts which should be ignored or at least given lesser weight in the decision. It is desirable that the classifiers generated by the machine learners make use of this extra information.

Various experiments were done with the before mentioned machine learning algorithms, as well as several methods of training them.

2.2 Related Work

In a call classification context, a machine learner must find a function which maps the spoken utterance to an appropriate class so that a call-center agent specialized in the class can take care of the caller's needs. The classifier does not deal with the audio data directly, but uses the output of speech recognizer. In literature there are several machine learning algorithms used for call classification. Here are just some:

- Vector based call routing ([ChuCarroll99, Carpenter98]): Words are converted to more basic forms using morphological analysis and stopwords are filtered out. For classifying, a sentence is transformed into a vector similarly and a distance measure is calculated. The result is a score (confidence) of each destination, which can be compared to a threshold for making a decision, or which can be sent to the disambiguation module if more categories have high scores.

- AT&T's "How may I help you", "Help Desk" and BoosTexter. The BoosTexter algorithm developed by Schapire and Singer ([Schapire00]) is based on AdaBoost and originally written to classify text, but has been tested successfully for classifying speech as well ([Schapire00, DiFabbrizio02, Rochery02]). Boosting makes a strong classifier out of several weak ones by "boosting" it with weights. The weak classifiers are decision stubs asking whether a word is present or not. The boosting training algorithm recalculates the weights by focusing on the examples which are hardest to classify, giving certain questions an extra "boost". In the end the class with the largest total of weights "wins".

In this work classification will be attempted with BoosTexter, as well as with RIPPER. RIPPER (Repeated Incremental Pruning to Produce Error Reduction) was developed by W. Cohen [Cohen95] and is a rule learning algorithm. The resulting classifier is a series of if-then-else rules and is easy to understand by humans. RIPPER is a modification of IREP (Incremental Reduced Error Pruning) which uses a growing and pruning process to generate rules. A rule learning system is attractive because the resulting classification algorithm is a series of if-then-else rules which test for the presence or absence of a word and are easy to understand and modify by humans. JRIP is WEKA's [Witten05] implementation of the RIPPER algorithm.

2.3 Methodology

A dummy system built with the Philips Speechpearl 2000 package was used to make recordings. In the first two days of data collection 558 calls were made over all categories and transcribed. The next few days an additional 916 calls were made and transcribed, also over all categories. These two sets are

the train and test sets respectively. The distribution of calls in the test and train set are not representative of the real-world situation, but are meant to provide each category with at least some training data.

A lexicon and language model were extracted from the training set, and all the data was passed through the speech recognizer. Some time was spent trying to get optimal performance from the speech recognizer, but the effort did not lead to results which would be considered "human readable". On the training set the speech recognizer had a word-error-rate of 25.60% and a sentence-error-rate of 64.34%. On the test set these values were 51.49% and 81.55% respectively. For speech recognition these scores are considered bad, but for classification it may not matter, as long as certain key words are recognized.

The recognition results are passed to the classification algorithm as-is. No stop word filtering or stemming is applied, partly because the machine learning algorithm should be able to decide itself what to consider important, and partly because an earlier research with a Dutch stemmer did not seem to produce better classification results [Gaustad02]. Furthermore the training and test sets are relatively small so the vector lengths are still acceptably small.

2.4 Tests (1)

Two algorithms were tested. These were JRIP, based on RIPPER and known for generating human-readable rules, and BoosTexter which is based on the Adaboost algorithm but adapted for text. BoosTexter was already used in literature for similar tasks of classifying calls. JRIP is particularly attractive because the output is very easy to understand, and in the future could serve as a basis for a human expert to continue with. The algorithms were tested by training on transcriptions, training on recognition output and training on scored recognition output.

The algorithms were tested by training on transcriptions, training on recognition output and training on scored recognition output. A summary of the results is shown in Table 8. For both algorithms, training on recognition output yielded the best results, with JRIP getting the highest score of 42.27% and BoosTexter 36.03%, followed by training on transcriptions. Training on scored output (with recognizer confidences) gave scores significantly lower. Overally the scores are much higher than the majority baseline of 12.12% and a bit lower than the maximum scores of 44.65% (JRIP) and 42.47% (Boostexter) which can be achieved with "perfect" recognition (using human made transcriptions).

Table 8: Accuracy of classifiers when tested on speech recognition output

Trained on	JRIP	BoosTexter
Transcriptions	41.26	35.37
Recognized output	42.47	36.03
Scored output	37.77	29.15

In general the impression is that JRIP can generalize better than BoosTexter, is more robust to noise, and has the better scores on the tests set. BoosTexter performed a lot better than JRIP when testing on the training set again, which is perhaps a sign of overfitting. Training on scored (with recognizer confidences) texts disappointedly gave scores significantly lower than when training on just the output without scores. This is probably attributed to the fact that the algorithm not only needs to learn the keywords but also needs to find suitable thresholds, meaning that more training data is needed.

2.5 Improvements

So far the results are far from good, and in practice this would mean that many calls would be directed to the wrong agent. All calls are classified regardless if the classification is of low confidence or not. It would be more meaningful to classify only on the calls which have a fairly high accuracy, and reject the others. The people whose calls are rejected can try again, possibly after being asked by the system to reformulate their request in another way. Also performance might improve when the number of classes is reduced.

Analyzing the rules in the classifier, the default JRIP output is a classifier which consists of a series of if-then-else rules with majority class taken as the default rule. This means that any utterance which does not match any of the rules will automatically be classified as the majority class, in this case "saldo". But logically if it were an unclear sentence it should have been rejected.

Some rules are very logical, for example "(inboedel >= 1) => categorie=inboedelverzekering" seems pretty straightforward, but some seem to make no sense at all, for example "(mij >= 1) => categorie=rechtsbijstand". The latter indicates overtraining or lack of enough data to make good general rules.

Some modifications to the JRIP classifier were made in order to add rejection capabilities. To implement a rejection facility a classifier is trained for the majority class only, namely "saldo". The training data is relabeled so that all except the majority class belong to one class, in essence creating a binary classification problem. The default rule for "saldo" is then replaced by the rules of this second classifier. The end result is that the classifier will now reject any utterance which does not trigger any of the rules, instead of assuming a "saldo" when there is no match.

Next modifications were needed with regard to how to deal with recognizer confidences. Using them to in the training stage did not have the desired effect, but there might be other ways. First an approach was tried where recognition results were filtered before being passed to the classifier based on a threshold. This way only words with sufficient confidence are passed to the classifier.

As the classification task is fairly complex and the performance reached up till now has not been spectacular, the problem was simplified. Classes with similar subjects were grouped together reducing the amount of classes by a factor of 3. For example the many classes for insurance were merged into one class etc. So the number of classes decreases and generally the amount of training data per class increases.

2.6 Tests (2)

The improvements mentioned in the foregoing section are tested now. First the JRIP with rejection facility and configurable threshold is tested, and after that the same problem with reduced classes.

2.6.1 Improved JRIP rules with rejection facility

Since words are now filtered before they are passed to the classifier, and the default output from JRIP was changed so that it doesn't classify what it does not recognize as the majority class, the classifier must be tested again.

Two JRIP models were trained. One using the same method as in the first series of tests. The other was trained by first renaming the data so that it becomes a binary classification problem by renaming all tags which are not "saldo" to "not_saldo". The resulting rules from both classifiers were merged into one classifier.

Tables 9 show the corresponding results after the modifications for various values of the threshold. The relative accuracy is the ratio of correctly classified calls from those which were not rejected, and absolute accuracy is the ratio of correctly classified calls from all calls, rejected and non-rejected.

Table 9: JRIP trained on training-set recognizer output with rejection

Threshold	Rel. Acc.%	Rej.%	Abs. Acc.%
0.000	52.93	23.69	40.39
0.100	53.31	24.24	40.39
0.200	54.86	25.98	40.61
0.300	56.21	27.95	40.50
0.400	57.75	30.24	40.29
0.500	58.51	32.64	39.41
0.600	60.45	36.24	38.54
0.700	62.13	40.61	36.90
0.800	63.94	45.20	35.04
0.900	65.96	48.69	33.84
1.000	69.45	54.26	31.77

Again the classifier trained on recognized output does slightly better than the one trained on transcriptions at comparable rejection levels. The rejection facility works, and it adds another parameter for configuring the classifier. Higher classification accuracy can be gained by setting the recognition threshold higher at the cost of more rejected calls.

Comparing the most similar tests in the previous test series which showed the best achievable accuracy result was around 42% (with no rejection allowed), the results are now ranging from 53% to 69% (relative) when the classifier is allowed to reject calls for varying values of the threshold. This improvement has a downside and will be explained further.

When counting rejections as misclassified, and thus computing the absolute accuracy, the scores in this test are ranging from 30% (highest threshold) to 40% (lowest threshold). The setting with the lowest threshold is the most comparable to the previous test series, and with 40% versus 42% in the previous test series this is a small decrease. The decrease can be explained by the fact there is now an additional classifier which distinguishes between "saldo" and "not_saldo" and the classifier can fail to recognize some calls as "saldo", whereas in the previous test series calls were given the label "saldo" by default if they did not match any other rule. However most of the calls which are now rejected would be false positives and wrongly classified as "saldo" in the previous test series.

Generally, for the sake of improving the dialog success rate, it is better to have a higher accuracy ratio when the classifier makes a classification, instead of having a lot of false positives where the caller is directed to the wrong agent, as would be the case in the previous test series. In the case of a rejection the caller can be asked to restate his request differently, and if it fails again the caller can ultimately be directed to a human agent. So overall the modifications are successful and a definite improvement, even if the absolute number of correctly classified calls is a bit lower.

2.6.2 Reduced classes

Because of overtraining and lack of enough data for each class, the data was relabeled to have fewer, but broader classes, and with more data for each broad class. This test serves to see how the performance of the classifier changes when it has to deal with the simplified problem.

The classifier was constructed much the same way as the other with rejection facility and consisting of rules from 2 classifiers.

The corresponding results are shown here in Table 10 for varying values of the threshold. The relative accuracy is the ratio of correctly classified calls from those which were not rejected, and absolute accuracy is the ratio of correctly classified calls from all calls, rejected and non-rejected.

Table 10: JRIP trained on reduced classes training-set recognizer output with rejection

Threshold	Rel. Acc. %	Rej.%	Abs. Acc.%
0.000	72.17	26.53	53.02
0.100	72.86	27.07	53.14
0.200	75.23	28.93	53.47
0.300	76.90	30.90	53.14
0.400	79.01	33.30	52.70
0.500	79.18	35.92	50.74
0.600	80.69	39.41	48.89
0.700	81.34	43.12	46.27
0.800	82.87	48.25	42.89
0.900	84.39	51.64	40.81
1.000	86.18	57.21	36.88

As this is now a simplified problem with less confusion between categories and more training data for each category, the accuracy is improved from 72% to 86% for varying values of the threshold. As expected, at comparable rejection levels in Table 9, both the relative and absolute accuracy scores gotten in this test are higher.

2.7 Conclusion

Overall the machine learning classifiers do not perform at an acceptable operational level yet, therefore these must be improved first. The apparent lack of data is a serious hindering factor in the training procedure. Using JRIP does not seem inferior to BoosTexter at this stage, but the performance of both leaves much to be desired. Unfortunately it is only possible to know once more data becomes available.

Yet, still some ways to improve performance can be thought of. Adding rejection and reducing the number of classes improved the performance, but the future several other improvements can also be done. With JRIP there is the possibility of easily modifying the auto-generated rules with some human

knowledge, especially in the beginning stages of a project, since it works pretty much like an expert system. Even though overall performance is still low, one can make use of the fact classification performance is not the same for all classes. By only allowing high scoring classes to be classified, thus further reducing the number of classes, the overall performance can go up even further, again at the cost of higher rejection.

References

[Carpenter98] - B. Carpenter, J. Chu-Carrol - Natural language call routing: a robust self-organizing approach - Proceedings of the 5th International Conference of Spoken Language, 1998.

[Chollet96] - G. Chollet, J. Cochard, A. Constantinescu, C. Jaboulet, P. Langlais, Swiss French PolyPhone and PolyVar: telephone speech databases to model inter- and intra-speaker variability - IDIAP-RR 96, 1996.

[ChuCarroll99] - J. Chu-Carrol, B. Carpenter - Vector-based Natural Language Call Routing - Computational Linguistics, 1999.

[Cohen95] - W. Cohen - Fast Effective Rule Induction - International Conference on Machine Learning, 1995.

[Cohen97] - P. Cohen, S. Dharanipragada, J. Gros, M. Monkowski, C. Neti, S. Roukos, T. Ward - Towards a universal speech recognizer for multiple languages - Proceedings of ASRU 1997.

[Content90] - A. Content, P. Mousty, M. Radeau - BRULEX, Une base de donnes lexicales informatise our le franais crit et parl - L'Anne Pschychologique, 1990.

[DiFabbrizio02] - G. Di Fabbrizio, D. Dutton, N. Gupta, B. Hollister, M. Rahim, G. Riccardi, R. Schapire, J. Schroeter - AT&T help desk - 7th International Conference on Spoken Language Processing, 2002.

[Fischer03] - V. Fischer, E. Janke, S.Kunzmann - Recent progress in the decoding of non-native speech with multilingual acoustic models - Eurospeech 2003.

[Freund97] - Y. Freund, R. Schapire - A decision-theoretic generalization of on-line learning and an application to boosting - Journal of Computer and System Sciences, 1997.

[Garofolo93] - J. Garofolo, L. Lamel, W. Fisher, J. Fiscus, D. Pallett, N. Dahlgren - The DARPA TIMIT Acoustic-Phonetic Continuous Speech Corpus CDROM, 1993.

[Gaustad02] - T. Gaustad, G. Bouma - Accurate Stemming of Dutch for Text Classification Alfa - LANGUAGE AND COMPUTERS, 2002.

[Keating98] - P. Keating, Word-level phonetic variation in large speech corpora - ZAS Papers in Linguistics 11, 1998.

[Köhler01] - J. Köhler - Multilingual phone models for vocabulary-independent speech recognition tasks - Speech Communication 2001.

[Kunzmann04] - S. Kunzmann, V. Fischer, J. Gonzalez, O. Emam, C. Günther, E. Janke - Multilingual Acoustic Models for Speech Recognition and Synthesis - ICCASP 2004

[denOs95] - E. den Os, T. Boogaart, L. Boves, E. Klabbers - The Dutch Polyphone corpus - Eurospeech 1995.

[Pellom01] - B. Pellom - SONIC: The University of Colorado Continuous Speech Recognizer - University of Colorado Technical report #TR-CSLR-2001-01, 2001

[Rochery02] - M. Rochery, R. Schapire, M. Rahim, N. Gupta, G. Riccardi, S. Bangalore, H. Alshawi, S. Douglas - Combining prior knowledge and boosting for call classification in spoken language dialogue - ICASSP 2002.

[Schapire00] - R. Schapire, Y. Singer - BoosTexter: A boosting-based system for text categorization - Machine Learning Vol 39 Issue 2/3, 2000.

[Schultz98a] - T. Schultz, A. Waibel - Language Independent and Language Adaptive Large Vocabulary Speech Recognition - ICSLP 1998.

[SpeechPearlManual] - Philips Dialog Systems - The Philips SpeechPearl 2000 manual, 2000.

[Stemmer01] - G. Stemmer, E. Nöth, H. Niemann - Acoustic Modeling of Foreign Words in a German Speech Recognition System - Eurospeech 2001.

[Sturm00] - J. Sturm, H. Kamperman, L. Boves, E. den Os - Impact of speaking style and speaking task on acoustic models - Proc. ICSLP, 2000.

[Uebler01] - U. Uebler - Multilingual speech recognition in seven languages - Speech Communication 2001.

[Uebler99] - U. Uebler, M. Boros - Recognition of non-native German speech with multilingual recognizers - Eurospeech 1999.

[Walker03] - B. Walker, B. Lackey, J. Muller, P. Schone - Language-reconfigurable universal phone recognition - Eurospeech 2003.

[Wells97a] - J. Wells - SAMPA computer readable phonetic alphabet - University College London, URL: http://www.phon.ucl.ac.uk/home/sampa/, 1997.

[Wells97b] - J. Wells - Computer Coding the IPA: a proposed extension of SAMPA - University College London, URL: http://www.phon.ucl.ac.uk/home/sampa/x-sampa.htm, 1997.

[Wester02] - M. Wester - Pronunciation Variation Modeling for Dutch Automatic Speech Recognition - PhD thesis, University of Nijmegen, 2002.

[Witten05] - I. Witten, E. Frank - Data Mining: Practical machine learning tools and techniques - 2nd Edition, Morgan Kaufmann, San Francisco, 2005.