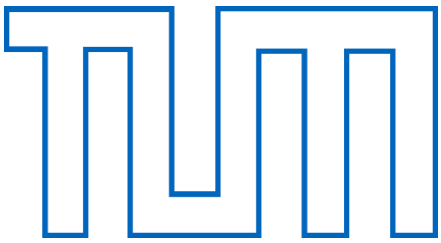
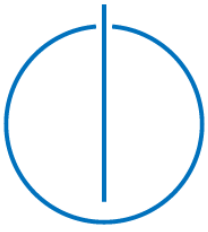


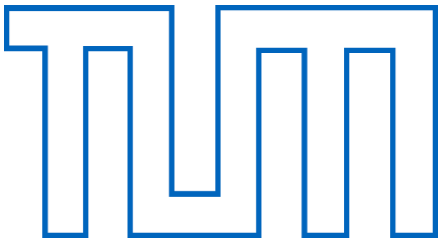
FAKULTÄT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Comparison Between Cloud-based and Offline Speech Recognition Systems

Elma Gazetić





FAKULTÄT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Vergleich zwischen Cloud-basierten und Offline Spracherkennungssystemen

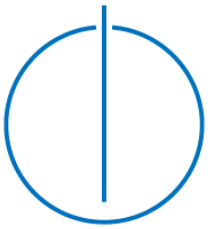
Comparison Between Cloud-based and Offline Speech Recognition Systems

Autor: Elma Gazetić

Supervisor: Prof. Bernd Brügge, Ph.D.

Advisor: Sajjad Taheritanjani

Submission date: Oct 16, 2017



I confirm that this master's thesis is my own work and I have documented all sources and material used.

16.10.2017 Elma Gazetić

Acknowledgments

I am sincerely and heartily grateful to my advisor Sajjad Taheritanjani for his advice guidance, encouragment, friendship and the time he has invested in me during these six months. I also wish to thank Prof. Bernd Brügge for his valuable suggestions.
Finally, special thanks to my parents and my brother for their support all the time. Without their support and unconditional love I would not even be able to study in Munich at TUM.
Access to IBM Watson is contributed by company „Pentos“, which is also greatly appreciated.

Abstract

Speech recognition system (SR) defines a system that converts a speech sample to text with the goal to be as accurate as a human listener. It has been used in different areas like robotics, health-care, and automotive.

As speech recognition is becoming more accurate in understanding what users say, more developers integrate it in their application. The goal is to make the SP system as accurate as possible for the user. If the trained network continually misinterprets what was said, it can drive the customer away. The developers have a choice between offline and cloud-based implementations of speech recognition systems. When the privacy matters, or we work on a very specific domain, then the offline speech recognition systems are favored. However, the accuracy of this SR system is not adequate in compare with cloud-based solutions.

In this thesis, first the training and recognizing in a SR system is observed. Then an open source SP system is chosen and reviewed in detail. The accuracy of the system is observed and steps for improvement of the accuracy are implemented. Finally, there is a comparison between the accuracy of both open source and cloud-based speech recognition systems.

Keywords: speech recognition systems, open-source speech recognition systems, IBM Watson, Kaldi, deep neural networks, SRILM, accuracy comparison

Table of Contents

List of Figures

List of Tables

1 Introduction

Speech recognition (SR) system is a rising core technology for next generation smart devices. Commercial usage scenarios are appearing in the industry as broadcast news transcription, voice search and real-time speech translation. Voice-based user interfaces are becoming increasingly popular, significantly changing the way people interact with computers. Assistant tools like Google Now from Google, Cortana from Microsoft, Watson from IBM, Siri from Cortana are advertising their SR systems more than ever in the technology division. But users are not always satisfied with the privacy and security of cloud-based SR systems. When the privacy matters or working on a very specific domain, then the offline speech recognition systems are favored. However, the accuracy of offline SR systems is not adequate in compare with cloud-based solutions. Often happens that user tries to recognize an instance that is not trained in the acoustic model. Therefore, to keep the users pleased, the developers have to find a way how to tune the accuracy of offline SR system, so that they can give results close to those from cloud-based systems.

1.1 Related Works

A wide variety of approaches for speech recognizers have been proposed in the literature for the past six decades.

Early work in the 1950s focused on developing of rules based on acoustic-phonetic knowledge or ad-hoc measurements of properties of the speech signal. (*1*) The intention was to decode the signal directly into a sequence of phoneme-like units. These methods had poor results mainly because of co-articulation causes and the inflexibility of rule-based hard decisions. (*1*)

An alternative to the rule-based methods is to use pattern-matching techniques, which were investigated at around the same time as early-based methods, but major improvements did not occur until 1960s. Early attempts to recognize human speech using pattern-matching techniques goes back to 1952, when Davis, Biddulph and Balaset of Bell Laboratories built a system for isolated digit recognition for a single speaker. (*2*) Their system relied heavily on measuring frequencies of each digit. Digits were recognized by comparing acoustic characteristics of the input signal with reference patterns, which was also called pattern matching. Pattern matching involves comparing the data acquired from the unknown signal with the corresponding data of a number of known signals considered as standards to determine which one of the standards was intended by the talker when speaking the unknown digit. (*2*) This system is trained by having the talker to be recognized generate a set of reference patterns or templates which are digitized and stored. It means that a training session is needed for each new user, who is required to speak examples of all the vocabulary words.

In the 1970s the main research direction in Automatic Speech Recognition (ASR) changes from template matching and rule-based to the statistical framework. One of the significant shifts in speech recognition has been the introduction of statistical methods, especially stochastic processing with Hidden Markov Models (HMM). (*3-5*) The earliest work on the theory of probabilistic functions of a Markov chain was published by Baum and co. (*6*) HMM provides a highly reliable way of modeling the intrinsic variability of the speech signal as well as the structure of spoken language, in an integrated and consistent statistical modeling framework. (*7*)

Jelinek of the Speech Processing Group in IBM's Computer Sciences Department is credited with being an early supporter of using statistical methods is speech recognition. (*5*) The standard algorithm for HMM training is the forward-backward or Baum-Welch algorithm (*8*), which is the special case of the Expectation-Maximization (EM) algorithm. (*9*) In the trained model, there must be enough states to capture all acoustically distinct regions in the words. A word of two or three syllables needs around 10-20 states to model the acoustic structure appropriately. (*1*)

At that same time, some research programs were funded in interest of searching the best strategies for building speech recognition. Defense Advanced Research Project Agency (DARPA) [1] established a funding for their Speech Understanding Research (SUR). The efforts of this project resulted in developing a few systems, where all the systems did not meet the performance goals of SUR.

Another system under the name DRAGON was developed by James Baker. (*10*) DRAGON introduced new techniques for speech processing, some of which are used in today's modern speech recognition systems. It used statistical techniques to make guesses about the most probable strings of words that might have produced the observed speech signal. DRAGON was the first example of use of HMMs in Artificial Intelligence (AI). (*10*)

From the end of 1980s systems were moving toward recognizing more natural speech, such as telephone conversations, interviews and radio broadcasts. Important milestone was the introduction of a broad-based speech recognition service within the telecommunications community called Voice Recognition Call Processing (VRCP) by AT&T. (*11, 12*) This automatic speech recognition (ASR) system provides users with more freedom regarding when, where and how they access information. (*12*) It enables 100 million customers of AT&T to identify the type of call they wished to make without speaking directly to a human operator. (*11*) Using a simple vocabulary of five phrases, users could speak any of the phrases and be reliably and accurately recognized more than 95% of the time. (*11*) Concept for speech recognition was keyword spotting. The technique of keyword spotting aimed at detecting a keyword in a utterance where there was no semantic significance to other words in the utterance. (*13*)

IBM's researchers aimed to create a "voice activated typewriter" system that could convert a spoken sentence into a sequence of words and letters that could be shown on a display. (*14*) The system created was named IBM Tangora system. It was the first real-time PC-based large vocabulary isolated word dictation system with 20,000-word vocabulary. (*15*) A study says that

[1] Retrieved from: https://www.darpa.mil/. On Date: 20.09.2017

Tangora performs better on sentences from some sources than others, which indicates that it is not a system that supports a general-purpose listening typewriter. (*15*) Tangora also uses the n-gram model, which defines probability of occurrence of an ordered sequence of n words. (*16*) In other words, the n-gram model characterized the word relationship within a span of n words. The n-gram model has started to be used successfully by speech recognition systems. (*3, 5, 17*)

Chen and Goodman produced a comparison of different language models, where they performed a number of experiments. (*18*) They showed the advantages of Interpolated Kneser-Ney, which became one of the most popular current methods for language modeling. (*19*)

In the earlies 1980s, Bell Laboratories extended the theory of HMM to mixture densities which has since proven importance in ensuring satisfactory recognition accuracy, particularly for speaker independent, large vocabulary speech recognition tasks. (*20*) This was a breakthrough as Gaussian Mixture Models (GMMs) could be used with the EM algorithm. GMMs are flexible, but that also can give them trouble because they can easily overfit to the data if there is not enough training data. (*21*)

A number of algorithms have been incorporated to work with HMM. These algorithms are important to accommodating a variable condition for the noise, channel, speaker, accent and so on. Most popular algorithms are Maximum a Posterori probability (MAP), Maximum Likelihood Linear Regression (MLLR), Vocal Tract Length Normalization (VTLN).

For search (decoding) strategies in speech recognitions algorithms like Viterbi have been applied.

Development of HMM-based systems benefited also to appearance of publicly available audio transcribed corpora for research purposes. Popular examples of corpora in English include acoustic-phonetic continuous speech corpus TIMIT, DARPA Resource Management continuous speech corpora, radio broadcast data from the Linguistic Data Consortium (LDC), telephone data Switchboards, Fisher large conversational telephone corpora, corpus based on TED-talks TED-LIUM (*22*) , LibriSpeech, Voxforge.

National Institute of Standards and Technology[2] (NIST) published benchmarks for previous mentioned corpora. [3] This evaluation shows that the performance of ASR systems was improving over time with respect to the size and difficulty of the task.

There have not been any challengers for the GMM-HMM based ASR system until the Neural Networks. In late 1950s and early 1960s a number of labs developed research into neural networks, (*23, 24*) but failed to produce notable results. Learning in NN has gone through many different names, and has only recently become called “deep learning”.

[2] Retrieved from: https://www.nist.gov/. On Date: 21.09.2017

[3] National Institute of Standards and Technology (NIST), Retrieved from: https://www.nist.gov/itl/iad/mig/rich-transcription-evaluation. On Date: 21.09.2017

First wave of started with cybernetics in the 1940s-1960s with the development of theories of biological learning (*25*) and implementation of the first models such as Perceptrons which trained neurons. (*26*)

McCulloh and Pitts built an automata-like model of the neuron, called the McCulloch-Pitts neuron. The model was a binary device which recognized two different categories of inputs by testing weather the output is positive or negative. (*25*) This model it didn't have much success, because the weights needed to be set by the human operator (*25*) which meant that it lacked mechanism for learning.

Later in the 1950s a psychologist Rosenblatt came up with a way to make an artificial neuron learn. (*27*) He conceived the Perceptrons as a simplified mathematical model of how the neurons in our brains operate, inspired by the work of psychologist Hebb. Hebb's Rule says that the learning in brain occurs through the formation and change of synapses between neurons. (*28*) The problem of Perceptron was that it didn't work for multiple layers, but only for a single layer. (*29*)

The second wave started with connectionist approach of the 1980-1995 period with back-propagation to train a neural network with one or two hidden layers. (*30*) The three authors in (*30*) were successful of using back-propagation to train deep neural networks and the popularization of the back-propagation algorithm.

A. Waibel introduced Time Delay Neural Network (TDNN) approach which manages a challenge of an input being a long sequence. (*31*) Instead of looking at the whole input at once, each unit presented a subset of the input at a time. A comparison was made between TDNN and HMM, where TDNN achieved a better reduction in error. (*32*)

But still many networks had problems with back-propagation, because back-propagation did not work well with NN with many layers and NN was known for having many layers. This problem was solved by Hochreiter and Schmidhuber that introduced the long short-term memory (LSTM) (*33*) to resolve difficulties in modeling long sequences. (*34*)

The current and third wave which is called deep learning, started around 2006. (*35-37*)

Geoffrey Hinton showed that Deep Belief Networks (DBN) could be trained efficiently using a strategy called greedy layer-wise pretraining. The idea of the strategy was to initialize the weights in a clever way rather than randomly. (*36*) Meanwhile authors in (*35*) and (*37*) showed that many other deep networks could be trained with that same strategy. This wave was emphasized under the term deep learning because the researchers were able to train deeper NNs than before. (*37-39*)

CPUs were hitting their limit in speed growth in NN learning. To use deep neural networks, CPU parallelism had to be replaced with computing powers of GPU. In 2009 authors of (*40*) discovered that using GPUs over CPUs was 70 times faster.

In 2010 a group of researchers explored old approaches for training NNs. (*41, 42*) They came to the conclusion that in order to use supervised learning with back-propagation the labeled datasets had to be larger, the computers had to be faster, the weights need to be initialized in a clever way and to use the right type of non-linearity.

The active research of speech recognition led to arrival of different public and licensed software tools. Most popular public software tools are Sphinx (*43*), HTK (*44*), RWTH (*45*) and Kaldi (*46*). Most popular licensed software tools are Siri, IBM Watson, Google Speech API, Bing Speech API, Amazon Alexa, Cortana.

1.2 Goal of The Thesis

The focus of this master's thesis is training acoustic models and recognizing speech samples using offline speech recognition systems in favor of privacy, managing specific domain acoustic models and executing offline tasks. It brought us to the idea to develop an application which can help us to compare the offline speech recognition systems to cloud-based recognition systems. The speech recognition trainer is going to collect the dataset for training and then continue to the next step which is training the acoustic model. After the acoustic model is trained, the user is able to use the speech recognition application. With the speech recognition application, the user can record a speech sample and navigate the application to get the output results. If the user is not satisfied with the results, a feedback can be sent to the speech recognition trainer. The accuracy of the system is observed and steps for improvement of the accuracy are implemented. Finally, there is a comparison between the accuracy of both offline and cloud-based speech recognition systems.

This thesis does not go in depth with next topics as text-to-speech, natural language classifier, language translator and tone analyzer speech systems.

2 Speech Recognition Systems

Speech recognition (SR) systems has been traditionally defined as the ability of a computer to recognize speech. (*47*) More recently SR is defined as a system which accurately and efficiently converts a speech signal into a text message transcription of the spoken words, independent of the device used to record the speech, the speaker's accent or the acoustic environment in which the speaker is located. (*48*) The goal of SR is to perform as well as a human listener, which has not been achieved yet. (*49*) Despite the flaws, SR is an established technology which is integrated into computers, mobile platforms, automobile systems and many other devices.

2.1 Speech Recognition Architecture

A typical SR system is shown in Figure 2.1, which emphasizes the two main steps of the architecture, namely the training and recognition tasks. (*50*) The figure shows that the SR system does not model speech directly at the waveform level. The feature extraction block extracts specific features which are further used in the acoustic model creation in the training stage. The same block is also used in the recognition process for speech representation.

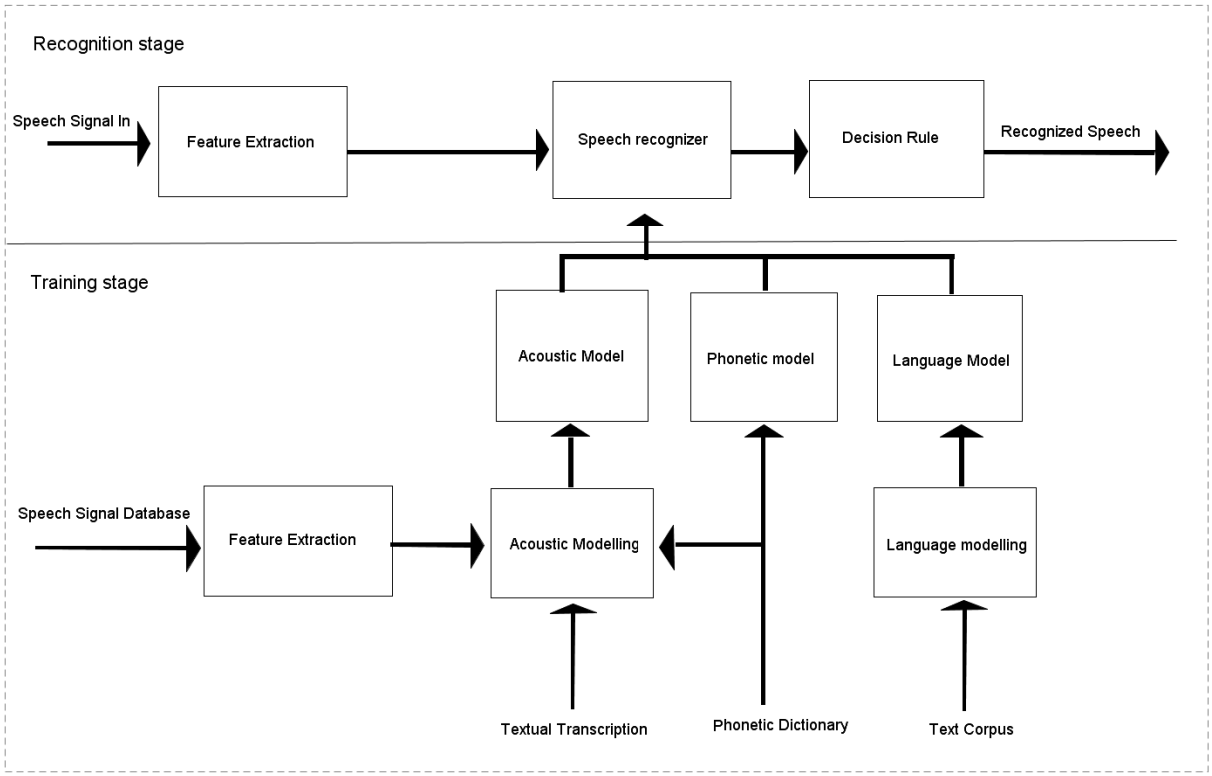


Figure 2.1 General architecture of a speech recognition system

2.2 Feature Extraction

The speech signal is processed in order to extract information for further analysis. (*19*) This is called feature extraction and is a processing stage that samples the acoustic waveform into a sequence of acoustic feature vectors where each vector represents a unique information. These feature vectors are modelled by the acoustic model. Speech signal is a slowly timed varying signal. Over a sufficiently short period of time (between 5 and 20ms), its characteristics are fairly stationary. However, over long periods of time, the signal characteristic change to reflect the different speech sounds being spoken. (*50*) There are many techniques used to represent a voice signal for speech recognition tasks, for digital use. Most common feature representation in speech recognition is the Mel Frequency Cepstrum Coefficients (MFCC). MFCC helps the SR system to emulate the human auditory system. It uses signal processing to match the frequency response of the eardrum. Sound waves travel down the auditory canal causing eardrum to vibrate. (*19*) The frequency response of the eardrum is modeled using the Mel scale. Modeling this property of human hearing improves recognition performance. (*19*)

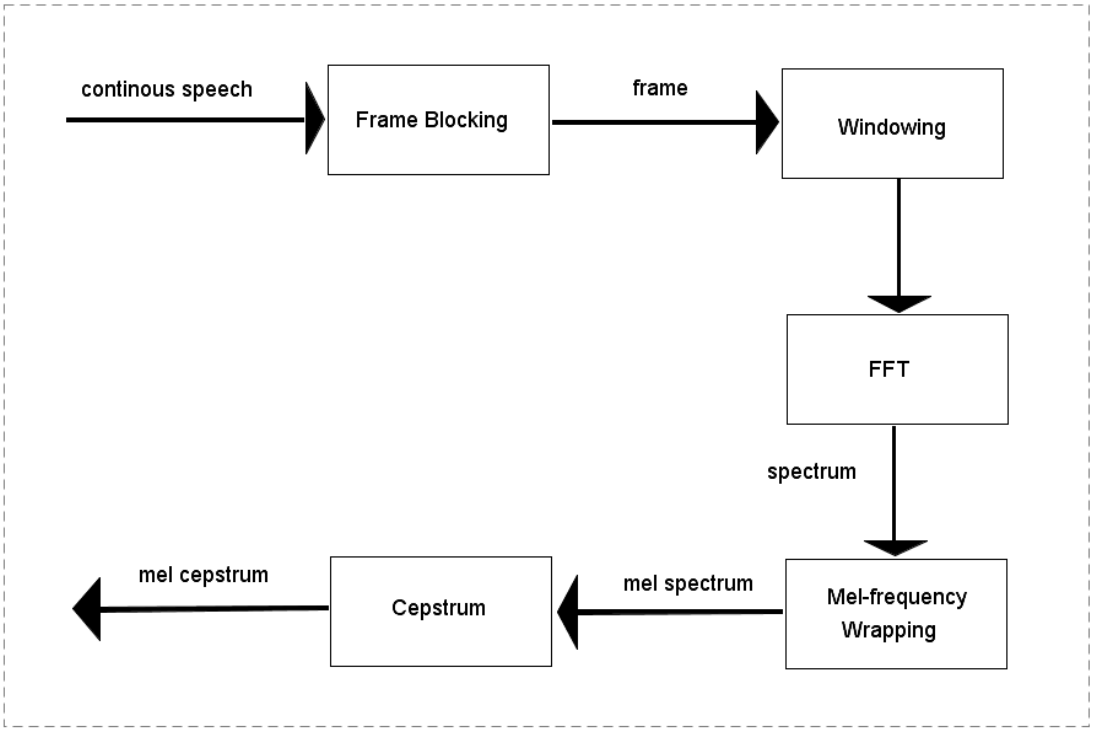


Figure 2.2 Block diagram of the MFCC feature extraction module

The steps of the MFCC process are shown in Figure 2.2. In the frame blocking section, the speech signal is broken into frames. The windowing block minimizes the discontinuities of the signal by tapering the beginning and end of each frame to zero. The Fast Fourier Transformation (FFT) block converts each frame from the time domain to the frequency domain representation. The result of the FFT is information about the amount of energy at each frequency band. Human hearing, however, is not equally sensitive at all frequency bands. It is less sensitive at higher frequencies,

roughly above 1000 Hertz. Modeling this property of human hearing during feature extraction improves speech recognition performance. (*19*) In the Mel-frequency wrapping block, the signal is plotted against the Mel-spectrum and as a result mimics the human hearing. And the last block Cepstrum is used to obtain cepstral coefficients. After the steps above, for each frame, a set of MFCC features are computed, which are called an acoustic vector. (*19*) These features extracted out of the speech signal are used for further modelling, in the training stage, or for speech recognition, in the recognition stage.

2.3 Language modelling

The language model P(W), models a word sequence by providing a predictive probability distribution for the next word based on a history of previously observed words. Since this probability distribution does not depend on the acoustics, language models may be estimated from large textual corpora. (*19*) One of the most commonly used language models is the N-gram model. (*51*) The conventional N-gram language model, which approximates the history as the immediately preceding n – 1 words, has represented the state of the art for large vocabulary speech recognition for last 25 years. (*51*) They are generally stored in ARPA format. Due to computational reasons, the history of preceding words cannot extend to include an indefinite number of words and has to be limited to m (3 to 5) words. Only a limited number of previous words affect the probability of the next word. Most commonly, trigram language models are used. (*50*) They consider a two-word history to predict the third word. This requires the collection of statistics over sequences of three words, so called 3-grams (trigrams).

2.4 Acoustic modelling

The most probable sentence W given some observation sequence O can be computed by taking the product of two probabilities for each sentence, and choosing the sentence for which this product is greatest. (*19*) The general components of the speech recognizer which compute these two terms have names. The first component marked as P(W), the prior probability, is computed by the language model. While the second component marked as P(O|W), the observation likelihood, is computed by the acoustic model. (*52*) The language model (LM) prior P(W) expresses how likely a given string of words is to be a source sentence. Such a language model prior P(W) can be computed by using N-gram grammars. (*19*) Given the AM and LM probabilities, the probabilistic model can be operationalized in a search algorithm so as to compute the maximum probability word string for a given acoustic waveform. Hidden Markov Model (HMM) are the most widely used representation of AM. For speech recognition, the probabilities of HMM can be modeled by using Gaussian Mixture Models (GMM) or Deep Neural networks (DNN). (*53*)

2.5 Speech recognizer

The task of speech recognition is to take as input an acoustic waveform and produce as output a string of words. Hidden Markov Model (HMM) based speech recognition view this task using the metaphor of the noisy channel. (*19*) Noisy channel treats the acoustic waveform as an "noisy" version of the string of words. Then through space of all possible sentences, the one which has the highest probability given the waveform is chosen. The implementation of noisy channel is requiring solutions to two problems. (*19*) First problem is the special case of Bayesian inference and second problem is the search problem or recognizing speech. First problem is solved with Bayes' rule which is used to estimate the probability of a noisy acoustic observation-sequence given a candidate source sentence. Second problem is solved searching through the space of all sentences and choosing the sentence with the highest probability. An efficient algorithm is needed for speech recognizing, so that search does not go through all sentences, but only those who have a good chance of matching the input. SR systems use Viterbi algorithm for speech recognizing with wide variety of sophisticated augmentations such as pruning. The main purpose of pruning is the limitation of the search space size which removes the sentences and words from the search space when their likelihood falls outside a beam relative. (*54*) But Viterbi approximation is not always giving the best results for SR systems. Problems appear when the phonetic model has multiple pronunciations for each word or when the language model is more complex than a bigram grammar. The simplest solution to this problem is to modify the Viterbi algorithm to return the N-best sentences for a given speech input.

2.6 Learning with Speech Recognition Systems

The existing techniques for learning in SR systems are based primarily on the Hidden Markov Model (HMM) with Gaussian mixture output distributions. (*55*) The downside of this technique happens when a SR system is developed with bigger computations and data resources. It causes accuracy improvements to slow down. To overcome this problem the SR systems are taking advantage of machine learning techniques. (*55*) Machine Learning is the science and art of programming computers so they can learn from data. (*56*) A more general definition was given in 1959 by Arthur Samuel who defined machine learning as the field of study that gives computers the ability to learn without being explicitly programmed. Even though SR is available commercially for some applications, the performance of SR is not on par with human performance. Insight from modern machine learning methodology shows great premise to advance the state-of-art in SR technology. (*55*) There are different types of machine learning systems.

There are different varieties of learning data for speech recognition. Systems where human supervision is of importance is called supervised learning, while human supervision is not needed by unsupervised learning. They are described more in chapter 2.6.1. Another type of machine learning is batch or online learning, which are explained in chapter 2.6.3. Chapter 2.6.4 defines a way how to test the performance of the trained model after the learning if finished.

2.6.1 Supervised and Unsupervised Learning

In supervised learning, the training data that is being fed to the algorithm includes also the solutions which are called labels. (*56*) Most important supervised learning algorithms are: k-Nearest Neighbors, Linear Regression, Logistic Regression, Support Vector Machines and Neural networks.

In unsupervised learning, the training data is unlabeled. (*56*) In general, it refers to learning with input data only. This learning aims at building representations of the input that can be used for prediction, decision making and data compression. (*55*)

2.6.2 Neural Networks

Most current speech recognition systems use hidden Markov models (HMMs) to deal with temporal variability of speech and Gaussian mixture models (GMMs) to determine how well each state of HMM fits the frames of coefficients that represent the acoustic input. (*57*) GMMs have a number of advantages that makes them suitable for modeling the probability distributions over vectors of input features that are associated with each state of an HMM. But despite all those advantages they are inefficient for modeling data that lie on a nonlinear manifold in the data space. (*57*) Therefore, other types of models need to be used for these kinds of problems.

Deep neural networks (DNN) is a feed-forward artificial neural network that has more than one hidden layer, where training is typically initialized by a pretraining algorithm. (*58*) Each hidden unit of the network uses a nonlinear function to map the feature input from the layer below to the current unit. (*59*) DNN can be trained by back-propagating (BP) derivatives of a cost function that measures the difference between the target outputs and the actual outputs produced for each training case. (*58*) But BP can easily get trapped in poor local optima for deep networks, and because of that is helpful to pretrain the model. There are two methods to pre-train DDNs, the unsupervised pre-training method (*57*) and the supervised pre-training approach also called discriminative pre-training method. (*58*)

2.6.2.1 Unsupervised pre-training

Recent researches in Machine Learning have led to the development of algorithms which can be used to train deep neural networks. One of these approaches is the Deep Belief Network (DBN), a multi-layered generative model which can be trained greedily, layer by layer using Restricted Boltzmann Machine (RBM) at each layer. (*57*) Unsupervised pre-training methods uses RBM to initialize the DNN. RBM is a type of undirected graphical model constructed from a set of visible and hidden units that allow interactions between the two sets of variables, but not within each set of nodes. Symmetric weight matrix defines the connection between the two sets of variables.

If many layers have to be initialized, the parameters of the given layer are fixed and its output is used as the input to the higher layer which is optimized as a new RBM. This can be repeated as

many times as desired to produce many layers of nonlinear feature detectors that represent progressively more complex structure in the data. Once the unsupervised pre-training is done, the obtained parameters are used as the initial values of the DNN, and BP is adopted to fine-tune the network. It has been observed that using parameters of a DBN to initialize a DNN before fine tuning with backpropagation leads to a better performance of a DNN. (*58*)

2.6.2.2 Supervised pre-training

One of the main challenges of the DNN training is initialization. Using pre-trained DBN is one of several initialization methods. There is another method called discriminative pre training which has been proposed in (*58*). In this approach, a one-hidden-layer DNN is trained to full convergence first using the softmax function [4] which creates a new type of output layer for the neural network. Then, the softmax layer is replaced by another randomly initialized hidden layer and a new random softmax layer is added on top of this. Afterwards, the network is discriminatively trained again until full convergence. This process is repeated until the desired number of hidden layers is reached. In (*58*) is also shown that there is no significant difference in terms of performance between using pre-trained DBN and discriminative pre-training techniques. Moreover, using discriminative pre-training is even slightly better than using pre-trained DBN when the number of hidden layers increases.

2.6.3 Batch and Online Learning

In batch learning, the SR system is incapable of learning incrementally. (*60*) The systems must be trained using all the available data which generally takes a lot of time and computing resources. When the system is trained, launched into production and then runs without learning anymore implies that the system is learning offline. (*56*) Using batch learning system to know about new data, a new version of the system is needed to be trained from scratch. Then the old system is stopped and replace with the new one. (*60*) This solution works but the training takes many hours. In case the system needs a rapid change, then a more reactive solution is needed.

In online learning, the SR system is trained incrementally by data instances which are fed to the system sequentially. (*56*) Each learning step is fast and cheap, so that the system can learn about new data on the fly as it arrives. Online Learning is also known as incremental learning algorithm. Big challenge with incremental learning is that if bad data is fed to the system, the system's performance will gradually decline. (*61*)

[4] Michael Nielsen. (2017). Improving the way neural network works (Chapter Softmax). Retrieved from: http://neuralnetworksanddeeplearning.com/chap3.html. On date 10.10.2017

2.6.4 Testing and Validating

For performing supervised learning two types of data set are needed: the training set and the test set. As these names imply, the training set is used for training a model, and test set is used for testing the model. (*62*) The error rate that is calculated from the testing is called the generalization error. (*56*) This error tells how well the model performs on instances it has never seen before. However most modelling techniques have parameters. The question is how to choose the right value for the parameter? One way is to train 100 different models using 100 different values and then chose the model with the lowest generalization error. The major problem with this is that the chosen model is produced as the best model only for that set. A common solution to this problem is to have another set which is called validation set. (*56*) Training set is used for training again, but now instead of using test set to determine the parameters for the model, the validation set is going to be used for that purpose. (*62*) After the model is chosen, test set is only used in the single final test to get an estimate of the generalization error.

2.7 Evaluating the Performance of Speech Recognizer

A commonly metric used to evaluate the performance of SR systems is the word error rate (WER). (*19*) For simple recognition systems, as systems with isolated words, the performance is simply the percentage of misrecognized words. However, in continuous speech recognition systems, such measurement is not efficient. The sequence of recognized can contain three types of errors. First error, known as substitution (S), happens when an incorrect word is recognized in place of a correctly spoken word. The second error, known as deletion (D), happens when a spoken word is not recognized. Finally, the third error, known as insertion (I), happens when extra words are estimated by recognizer. WER is the edit distance between the reference word sequence and its automatic transcription by the SR system. The WER is defined as

$$\text{WER} = 100\% \times \frac{\text{S}+\text{D}+\text{I}}{|\text{W}|} \quad (63)$$

where |W| is the number of words in a sentence. In WER, all words are considered equally important and all errors are considered equally bad. But in practice that is not true because the impact of all errors is not the same. If an error does not hurt a person's ability to understand a transcription, such an error has a lower impact than an error that distorts the meaning of the message. (*63*)

3 Analysis

During requirement elicitation, the purpose of the speech recognition (SR) system is defined. The analysis focuses on producing a model of the SR system, which is called the analysis model. (*64*) This chapter begins by examining the main requirements elicitation concepts like functional requirements and non-functional requirements in chapters 3.1 and 3.2. In the final chapter 3.3, the analysis object model of SR system is discussed.

3.1 Functional Requirements

Functional requirements describe the interactions between the system and its environment independent of its implementation. The environment includes the user and any other external system with which the system interacts. (*64*) SR system interact with the speech recognition trainer who sets up the system. The speech recognition trainer has the assignment to collect and prepare the data, train and test the model. The trainer also monitors the unknown instances that appear over time. For every unknown instance retraining of acoustic model and updating of language model is required. After the training, the SR system is interacting with the user that sends audio files to the speech recognition application and finally gets the text results.

3.1.1 Scenarios

A scenario is “a narrative description of what people do and experience as they try to make use of computer systems and applications”. (*64*) A scenario is a concrete, focused, informal description of a single feature of the system from the viewpoint of a single actor. In this section the scenarios for the speech recognition trainer and the user are shown.

Scenario Name	trainAcousticModel
Participating Actor	max: speechRecognitionTrainer deep neural network: learningFramework
Pre-condition	Max gets the dataset needed for training and testing from external sources on the internet [5] [6]
Flow of events	1. The speech recognition trainer prepares the training, testing and validation data for training and testing
	2. The speech recognition trainer trains the acoustic model using deep learning framework
	3. Deep neural network creates an acoustic model from the training data
	4. The speech recognition trainer tests the trained acoustic model
Post-condition	Deep neural network has created an acoustic model.

Table 1 Scenario "trainAcousticModel"

Scenario Name	recognizeSpeech
Participating Actor	ana: user
Pre-condition	The speech recognizer application uses a trained acoustic model
Flow of events	1. Ana records a speech sample and sends it together to the speech recognizer application
	2. Ana gets the results from the speech recognizer application
Post-condition	The speech recognizer application gives an output to Ana.

Table 2 Scenario "recognizeSpeech"

Scenario Name	retrainAcousticModel
Participating Actor	max: speechRecognitionTrainer
Pre-condition	1. Model does not recognize instances 2. Max collects audio and text data for the unknown instances
Flow of events	1. The speech recognition trainer prepares the training, testing and validation data for training and testing
	2. The speech recognition trainer updates the language model
	3. The speech recognition trainer trains the acoustic model
	4. The speech recognition trainer updates the language model and acoustic model
Post-condition	The speech recognizer application recognizes the unknown instances.

Table 3 Scenario "retrainAcousticModel"

[5] Open Speech and Language Resources. Retrieved from: http://www.openslr.org/. On date 5.10.2017

[6] Linguistic Data Consortium. Retrieved from: https://www.ldc.upenn.edu/. On date 5.10.2017

3.1.2 Use Cases

A scenario is an instance of a use case which means that the use cases in this chapter will specify the scenarios which were described in section before. A use case represents a complete flow of events through the system in the sense that it describes a series of related interactions that result from its initiation. (*64*)

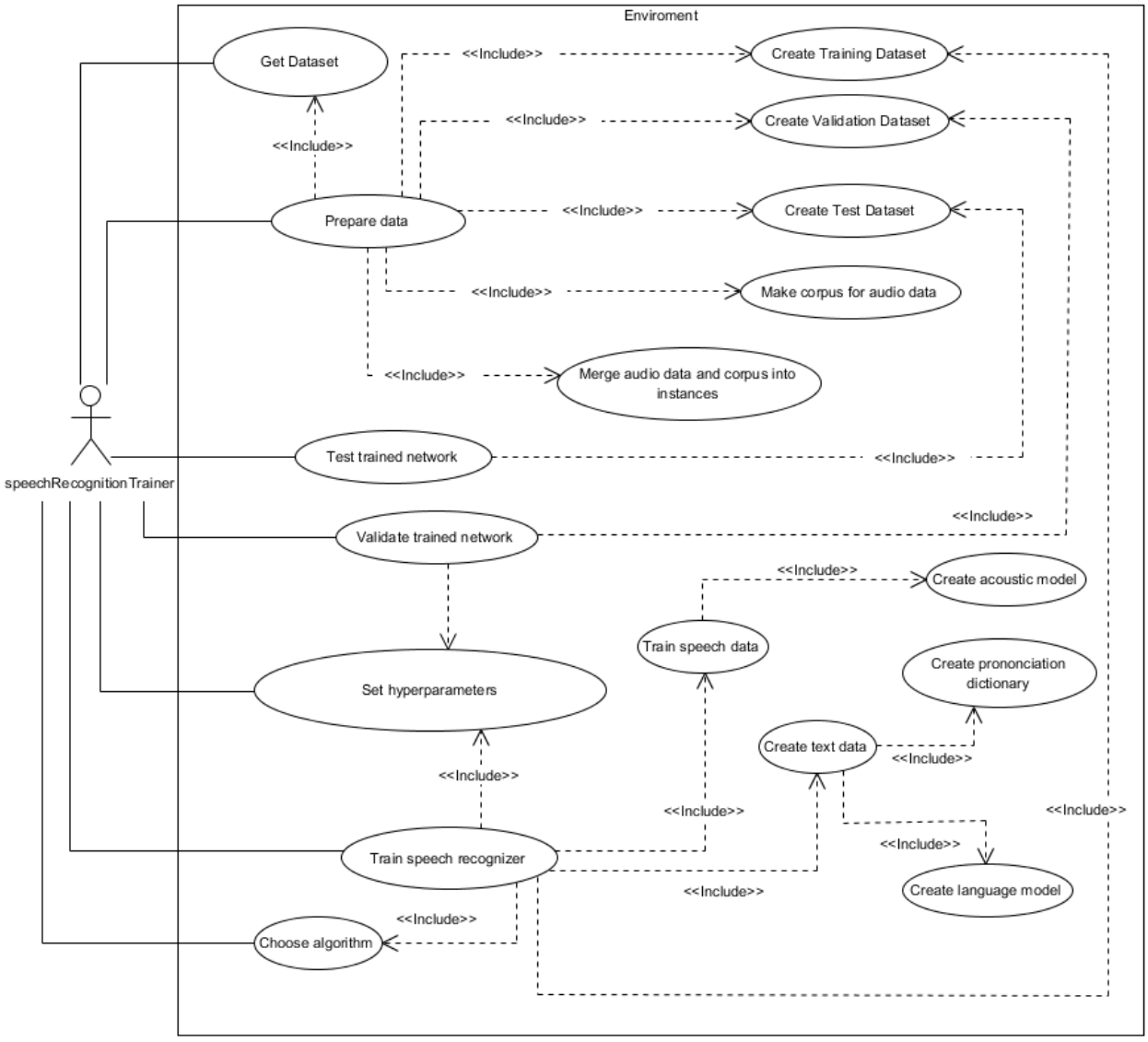


Figure 3.1 Use case model for actor "speechRecognitionTrainer"

First challenge for the speech recognition trainer is to choose the algorithm by which the data is going to be trained. For supervised learning algorithms data needs to be labeled, while for the unsupervised learning algorithms the data does not need to be labeled. With the chosen

algorithm, next phase would be gathering the data for the task in a specific domain the SR system is supposed to train on. The data should be a good representation of what speech it is going to recognize. The data should be split into three parts, train, test and validation data. (*62*) Some SR systems have specific structures of how the data should look like, which means that there is also a data preparation step as merging data into separate instances, creating files for extra information about data and so on. Before the actual training of speech recognizer starts, hyperparameters need to be fixed as shown on Figure 3.1. Hyperparameters cannot be learned directly from the data in the standard process training and need to be predefined. Multiple acoustic models with various hyperparameters should be trained using the training dataset, and then the hyperparameters that performs the best on the validation dataset should be selected. (*56*) Training the speech recognizer, also a critical component of a SR system, can take a few hours or a few weeks, all depending on the machine's speed and the data size. Sometimes the speech recognition trainer wants to make changes to the speech recognizer that is already trained. This happens when instances or unknown words appear that are not already in the trained acoustic model and language model.

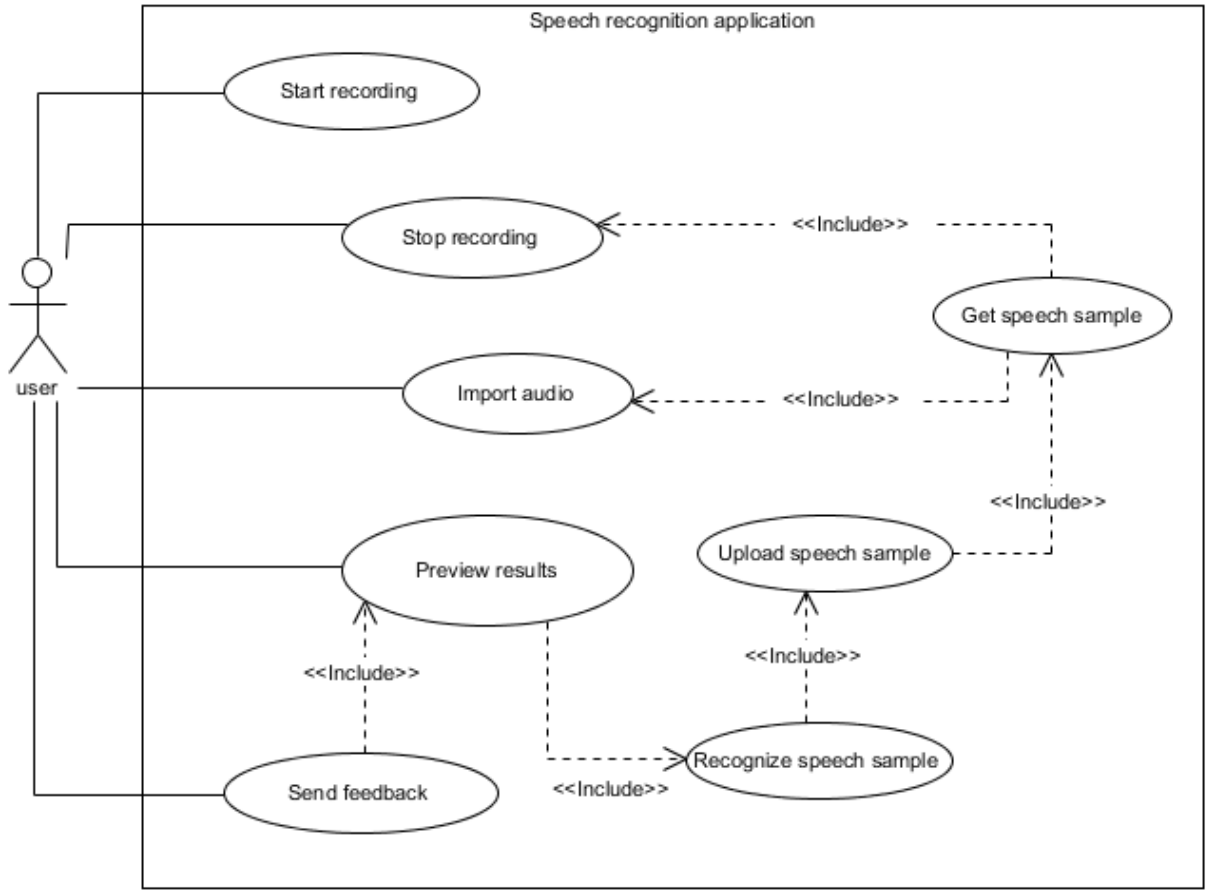


Figure 3.2 Use case model for actor "user"

The user uses the speech recognition application, which takes as an input a speech sample and gives as an output the text of the speech sample. The user can start recording a speech sample

with the “Start Recording” button. To stop the speech recognition, the user needs to activate the “Stop Recording” button. Otherwise the user can also upload an already recorded speech sample and the application gives a preview of the result on the interface. If there are unknown instances in the text, then the users can send a feedback to the speech recognition trainer and inform the trainer which instances need to be added to the acoustic and language model. This user case model is shown in Figure 3.2.

When the speech recognition trainer gets an information about unknown instances, the data about the unknown instances need to be collected. For the acoustic model to be retrained audio data is required, while for the language model to be updated text data is required. After the data is acquired, the language model is updated and the acoustic model is retrained. Next step is to validate and test the trained neural network with the audio data that was collected.

3.2 Non-functional Requirements

Non-functional requirements describe aspects of the system that are not directly related to its functional behavior. They are addressed consistently during the realization of the system. (*64*) There is a broad variety of requirements that describe nonfunctional requirements: architecture requirements, latency, accuracy, precision and customization. (*65*) For nonfunctional requirements two different sessions are identified as training and speech recognition. These nonfunctional requirements are described in following subchapters.

3.2.1 Architecture requirements

The term open source refers to something people can modify and share because its design is publicly accessible. To be able to research more about the topic of this thesis the data sets for training should be open source and also the speech recognition system which train the data should be a publicly accessible. When risk of privacy, working on very specific domains and offline recognition matters then the open-source systems are favored.

3.2.2 Latency

Speech recognition systems depend on two stages which are training a model and recognizing a speech sample. While training is often computationally expensive and takes hours to days to complete, recognizing a speech sample is often assumed to be inexpensive and is required to be run in real-time. (*66*) Latency is the total time it takes from when a user makes a request until they receive a response. This is an important performance measurement from a user’s perspective. (*67*) For best user experience, a real-time speech recognizer application should have low latency time as in tens of milliseconds. If the latency exceeds the targeted time, it may be noticeable and likely distracting to a user. (*67*) Latency of the speech recognizer processing may

be caused by a number of factors as unclear speech, delays introduced when the voice signal is transmitted to the remote speech recognition device.

3.2.3 Accuracy

Accuracy refers to the closeness of a measured value to a standard or known value. It is often the most intuitive performance measure, but it should not be relied too much on because most data sets are far from symmetric. Accuracy refers to the ratio of correctly predicted observations. A common metric that is used for reporting the accuracy of a speech recognition system is word error rate (WER). It computes a value in percentage by comparing a machine transcription to a human transcription. Today the WER has reached a breakthrough of 5.5% on a specific domain in cloud based SR systems. [7] The WER of offline SR systems is between 6.5% and 22.9% according to speech recognition tests that were taken in (*68-70*).

3.2.4 Precision

Precision is the ratio of correct positive observations (true and false positives). For a higher precision the goal is to reduce false positives. Precision refers to the percentage of the hits that were valid hits. This measure is important because important terms need to be recognized correctly. If a speech recognizer cannot recognize an important term for a use case, then the accuracy is irrelevant.

3.2.5 Customization / Extensibility

In a perfect world where unlimited storage, processing power, time and other resources are available, a speech recognizer would have access to a customized speech recognition model for every possible domain of interest, but this includes collecting all domain-specific data which is very expensive. And that is why if a sufficient amount of domain-specific data is available, then a domain-specific model is built. (*71*) The challenge is to provide a customized model with as little domain-specific data as possible, which can result in missing some important terms and lower precision. Customization is important for benefit of a lower WER and higher precision. If the acoustic and language model become “outdated” and unable to recognize important terms then the acoustic and language model need to be updated.

[7] G. Saon. (2017). Reaching new records in speech recognition. Retrieved from: https://www.ibm.com/blogs/watson/2017/03/reaching-new-records-in-speech-recognition/. On date (06.10.2017)

3.3 Analysis Object Model

Figure 3.3 shows the main components in the speech recognition system and speech recognition application.

The *AcousticModelTrainer* can have different *Algorithms* by which the dataset can be trained. This process is implemented by the strategy pattern which is marked by an orange box in Figure 3.3. Strategy pattern is a behavioral software design pattern that enables selecting an algorithm at runtime. (*64*) Therefore the speech recognition trainer selects one of the *Algorithms* that is available in *AcousticModelTrainer*. With the expectation-maximization (EM) *algorithm* it is possible to develop speech recognition systems for tasks using the GMMs to represent the relationship between HMM states and the acoustic input.(*57*) Error back propagation (EBP) *algorithm* trains the *DNN-HMM* acoustic model by backpropagating derivatives of a cost function that measures the discrepancy between the target outputs and the actual outputs produced for each training case. (*30*)

The *Dataset* consists of *Instances* which include AcousticWave and Corpus. FeatureExtraction processes the acoustic waveforms into a sequence of *FeatureVectors.* These *FeatureVectors* are used for training the *AcousticModel.* The *Corpus* of the *Instance* is used in *TextModelFactory* to create a *PrononciationDictionary* and *LanguageModel*.

The speech recognition application which can be used by the user, has three buttons for importing a speech sample: *StartRecording*, *StopRecording* and *ImportAudio*. In case that there is a Microphone connected then the buttons *StartRecording* and *StopRecording* can be used to start and stop the recording. After recording a *SpeechSample*, the output of the speech engine is sent to the interface of the application, in this case *OutputGUI*. On the assumption that the microphone isn't connected or that the user has already a recorded *SpeechSample*, then the sample can be imported with button ImportAudio. After the speech sample is imported, the output of the speech engine is sent to the *OutputGUI*. Command pattern is used to trigger the appropriate Command for the user, as shown in gray box in Figure 3.3. In object-oriented programming, the command pattern is a behavioral design pattern in which an object is used to encapsulate all information needed to perform an action or trigger an event at a later time. (*64*)

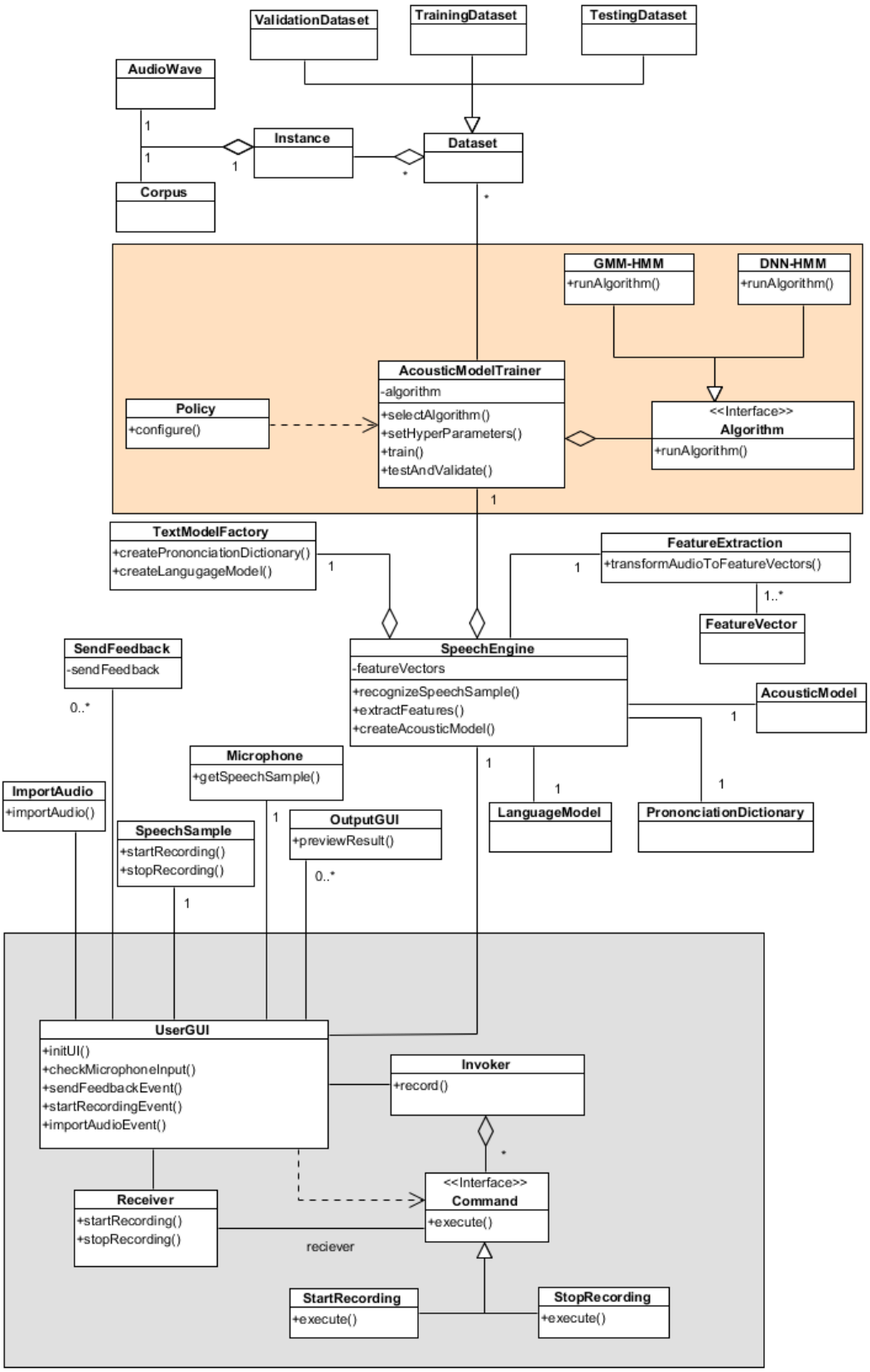


Figure 3.3 Class Diagram for analysis object model

4 System Design

During the system design, the design goals of the project are defined and the system is decomposed into smaller subsystems. (*64*) In addition, the strategies for building the system are selected, such as the hardware/software mapping and the handling of boundary conditions. The result of system design is a model that includes a subsystem decomposition and a clear description of each of these strategies. (*64*) In this chapter, a design model of an SR system is developed from previously mentioned non-functional requirements. Since the definition of design goals is the first step of system design, the chapter begins with identifying design goals of the SR system. Once there is a clear idea of the design goals, next step of designing an initial subsystem decomposition can be taken.

4.1 Identifying Design Goals

The design goals are derived from the nonfunctional requirements. Design goals guide the decisions that need to be made when trade-offs are needed. They also describe the qualities of the system that should be optimized. (*64*) There are 5 types of possible design criteria from which the design goals can be selected: performance, dependability, cost, maintenance and end user criteria. (*64*) Summary of the non-functional requirements are shown in next table.

Requirement	**Description**	**Criteria**
Security	The user wants privacy and security. It can be accomplished with offline speech recognition systems.	End user
Latency	For best user experience, a real-time speech recognizer application should have low latency time as in tens of milliseconds. If the latency exceeds the targeted time, it may be noticeable and likely distracting to a user.	Performance
Accuracy	Acceptable WER is between 6.5% - 15%	Performance
Precision	For a higher precision the results should not have false positives.	Performance
Extensibility	If the acoustic and language model become “outdated” and unable to recognize important terms then the acoustic and language model need to be extendible.	Performance

Table 4 Design goals

4.2 Subsystem Decomposition

During system design, subsystems are defined in terms of the services they provide. Two properties of subsystems are coupling and cohesion. (*64*) Coupling measures the dependencies between two subsystems, whereas cohesion measures the dependencies among classes within a subsystem. Ideal subsystem decomposition should minimize coupling and maximize cohesion. (*72*) A subsystem is a replaceable part of the system with well-defined interfaces that encapsulates the state and behavior of its contained classes. (*64*)

The decomposition should split the SR system into multiple subsystems where each subsystem consists of components. Each component should realize a certain service. Referring to the functional requirements subsystems should be created. (*64*)

UserInterfaceSubsystem	*UserInterfaceSubsystem* provides the user interface where speech recognition can be run. This component also connects to a microphone device.
DataSubsystem	On the other hand, *DataSubsystem* is responsible for collecting all the data for training and testing. Once the data is collected, the training in SR system can start.
SpeechRecognitionSystem	For training, testing and retraining is responsible *SpeechRecognitionSystem*. After the training is finished, the user interface can be used to start processing the audio file in the recognition process.

Table 5 Subsystem decomposition

All identified subsystems can be visually seen on

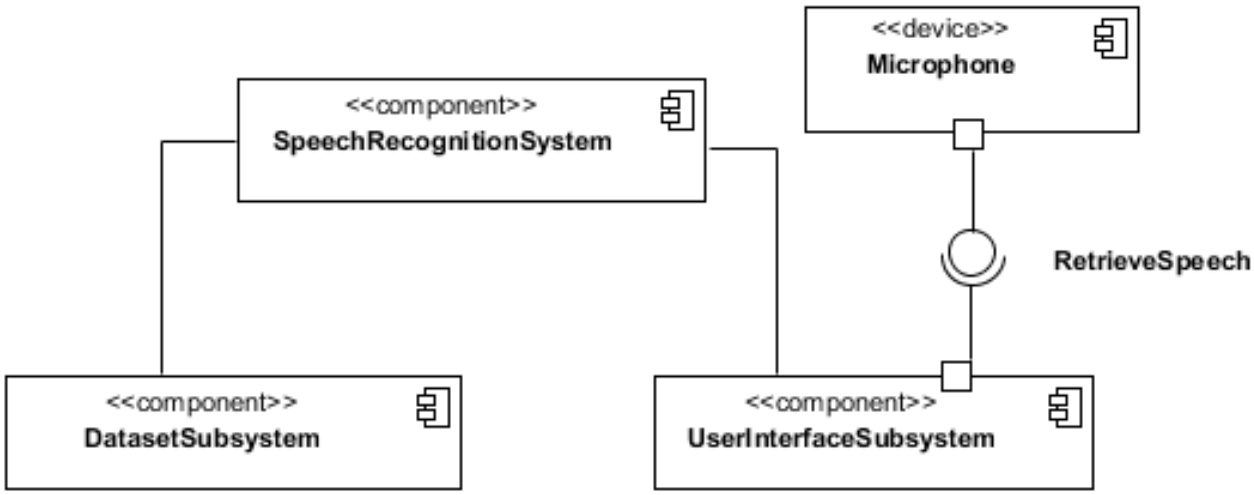


Figure 4.1 UML component diagram of identified subsystems of SR system

Next step is to identify components of each subsystem.

UserInterfaceSubsystem	*UserInterfaceSubsystem* contains the input content for the *RecognizerSubsystem* component. It also gets the output of the *Recognizer* component.
DataSubsystem	*DataSubsystem* contains all the data components which is needed for training and testing a model in SR system. Those are *TrainingData*, *TestingData* and *ValidationData*.
SpeechRecognitionSystem	*SpeechRecognitionSystem* contains the *SpeechTrainingSubsystem* which creates the *AcousticModel, PrononciationDictionary* and LanguageModel component. It also contains the *RecognizerSubsystem* component which contains the *Recognition* component which recognizes the input audio file.

Table 6 Subsystem decomposition

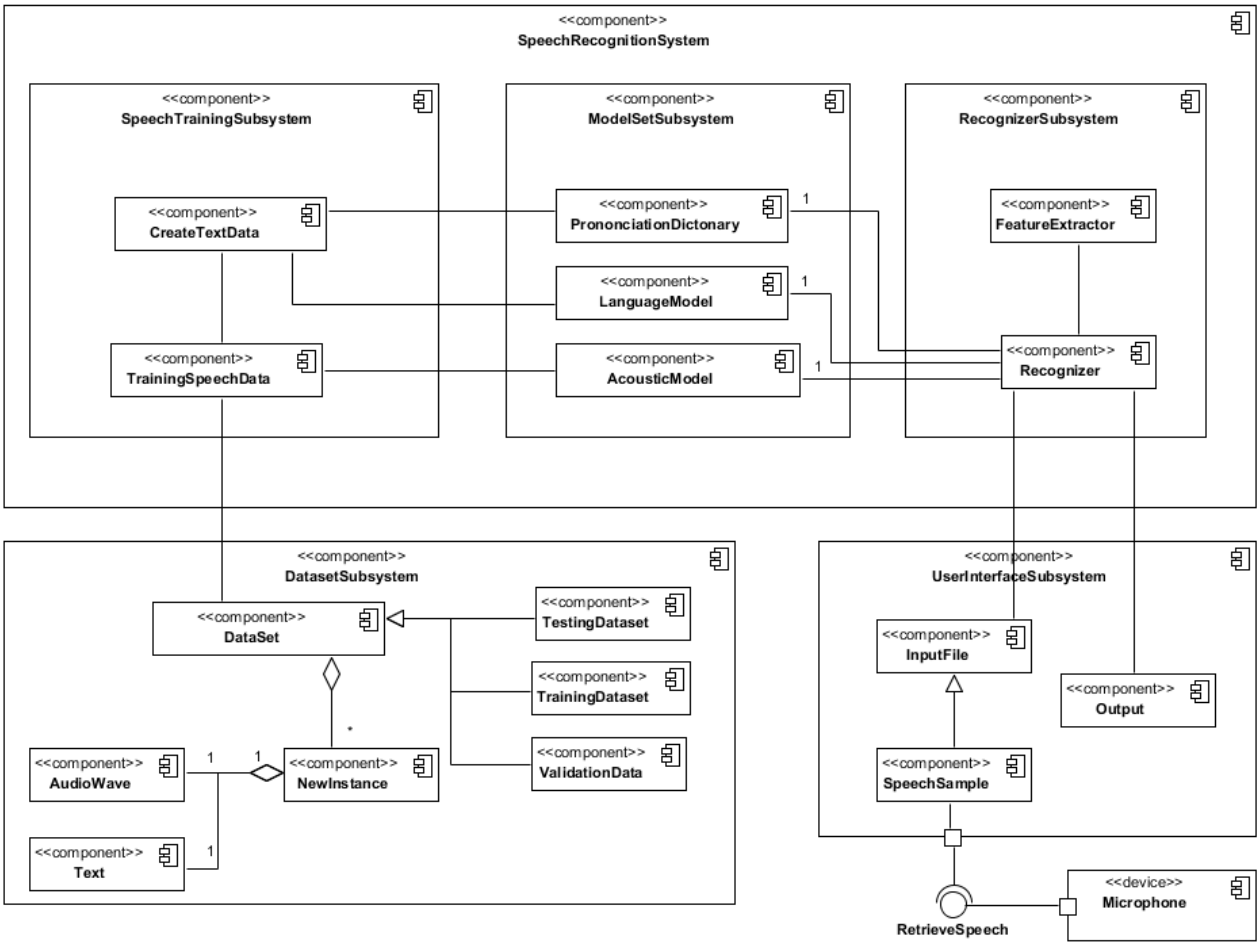


Figure 4.2 UML component diagram of subsystems and their components of SR system

4.3 Addressing Design Goals

In the previous chapter the concepts of design goal and system decomposition were identified. In this chapter subsystem decomposition is refined until all design goals are addressed. There are different design goals which need to be addressed: hardware/software mapping, data management, access control, control flow and boundary conditions. (*64*) Design goals which are addressed for SR systems is data management. While other four design goals are not interesting for SR systems.

4.3.1 Data Management

In data management, first we have to identify which data must be persistent. A persistent object continues to exist even when the application that created or used the object has finished execution. (*73*) Persistence can be implemented in many ways like:

- Storing objects in a simple file (Flat files)
- Storing object in relational database
- Storing in an Object-Oriented database (*64*)

The training, retraining and testing dataset is persistent data. SR system needs the training data for training the model and testing dataset for testing the model. These datasets take usually a lot of storage and that is why this data should be stored on a disk with bigger capacity. This brings to the conclusion that this data should be stored as flat file. Results of the training are also saved as flat files.

4.3.2 Boundary Conditions

The boundary conditions of a system are the ones which decide how the system started, initialized and shut down. (*64*) Also the boundary conditions defines how to deal with major failures such as data corruption and network outages, which are caused by software error or power outage. (*64*)

4.3.2.1 Configuration

In this section, each persistent object should be examined to check use cases it is created, destroyed or archived. Configurations for two sessions are going to be analyzed, training and speech recognition (SR) session.

For the training session, the SR system uses the provided dataset to train a model. The model which is trained is archived as an output for further use.

For the SR session, the application uses the archived acoustic model to start the recognizing of the speech sample. After the recognition phase, the result is archived in an output file.

4.3.2.2 Start-up and Shutdown

In this section, the start and shut-down process should be determined for each component. Two components are going to analyzed, training and deployment component.

The training session is started by the speech recognition trainer. The commands are run from the terminal and together with the proper input for the commands. The training session is also stopped by the speech recognition trainer, when terminal gets closed.

In this scenario, the speech recognition session is started by the user by opening the application and uploading a speech sample. The user can also press the start recoding button and record his voice for the application to start the execution. The user stops the recording by pressing the stop recording button.

4.3.2.3 Exception Handling

An exception is an event or error that occurs during the execution of the system. (*64*) Exceptions are caused by three different sources: a hardware failure, changes in the operating environment and software fault. (*64*) Error types are discussed in next sections.

– Error Caused by Hardware Failure

A hardware error is a malfunction of a hardware component in a computer system. They happen due to physical causes like corrosion, thermal stressing and wear-out. Some of the errors which can appear in SR systems are insufficient memory in Graphic Processing Unit GPU) card, insufficient memory in Hard Drive Disk (HDD) while training. Some SR systems also need to have internet connection, without internet errors could be triggered.

– Error Caused by Changes in the Operating Environment

An environmental error is an error in calculations that are being part of observations due to environment. Causes of environmental errors are: temperature, humidity, magnetic field or wind. Any experiment has its surroundings, therefore they cannot be eliminated from the system.

– Error Caused by Software Fault

As the primary cause of software defects, human error is the key to understanding and preventing software defects. Some human error categories appear to be very closely related to a learning process. (*74*) To cope with human errors and to reduce them in systems, some guidelines should be followed.

The first guideline for system is to accept that experiments are necessary for operators to be able to optimize skill. Thus, interface design should aim at making the limits of acceptable performance visible to the operators, while the effects are still observable and reversible. (*74*) The SR system

has to be tested by the speech recognition trainer and the user beforehand to make sure they learn how to use it correctly.

The second design guideline is to provide the actors with feedback on the effects of actions so as to allow them to cope with the time delay between the execution of an intention and the observation of its effect. (*74*) The speech recognition trainer and user should know what are the consequences and effects of each action performed when the system is trained and used.

The third guideline mentions that when designers assign the task to complete the design of control strategies during unforeseen situations to plant operators, they should supply them with the tools to make experiments and test hypotheses without having to do this on a high risk and possibly irreversible plant. (*74*) It is important that the user can give feedback to the speech recognition trainer about the trained model. In that way the speech recognition trainer can retrain the model to user's liking.

4.3.2.4 Error handling

Exception handling, as a subset of error handling is discussed in this section. Error handling refers to the anticipation, detection, and resolution of programming, application, and communications errors. (*75*) Next chapters describe how to deal and handle the errors and exceptions in SR system.

Errors that can happen in training sessions are connection-loss and out-of-memory errors. The only way to handle them is to give a detailed report to the speech recognition trainer what the error was and why it happened. The speech recognition trainer then needs to fix the connection of the network and to get big enough memory.

Errors can happen while recording the user speaking into the microphone and uploading a corrupt audio file. The application needs to check if a microphone is available and if it is working properly. If there are errors, then it needs to inform the user that the microphone is broken. In case the audio file uploaded in the application is corrupted then the user also needs to be informed with a notification in the application

4.4 Kaldi - The Open Source Speech Recognition System

The term open source refers to something people can modify and share, because its design is publicly accessible. There are a few open source recognition systems, but the one with the best performance is Kaldi according to experiments in (*68*). The experiment shows that Kaldi outperforms all other recognition toolkits like CMU Sphinx, HTK, Julius. And that is why as main open source SR system in next chapters Kaldi is used.

4.4.1 Hardware/Software Mapping Design

Hardware/Software mapping design is represented by deployment diagrams. (*64*) In this section, a design of a deployment diagram for each of the scenarios is described. Deployment diagram design identify the important mappings between hardware and software components without considering the implementation details. Some of the components in the diagram are changed into systems and devices which are used in the implementation.

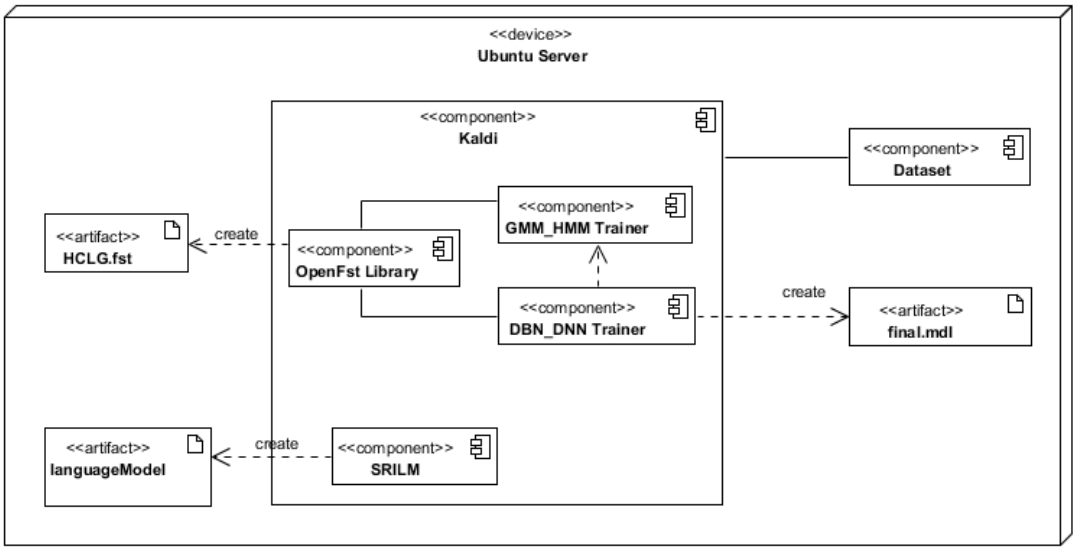


Figure 4.3 Deployment diagram for training acoustic model

Figure 4.3 shows the deployment diagram design for the training acoustic model scenario. The left and the right side of the diagram show the software components which perform as the input and output of the Kaldi component.

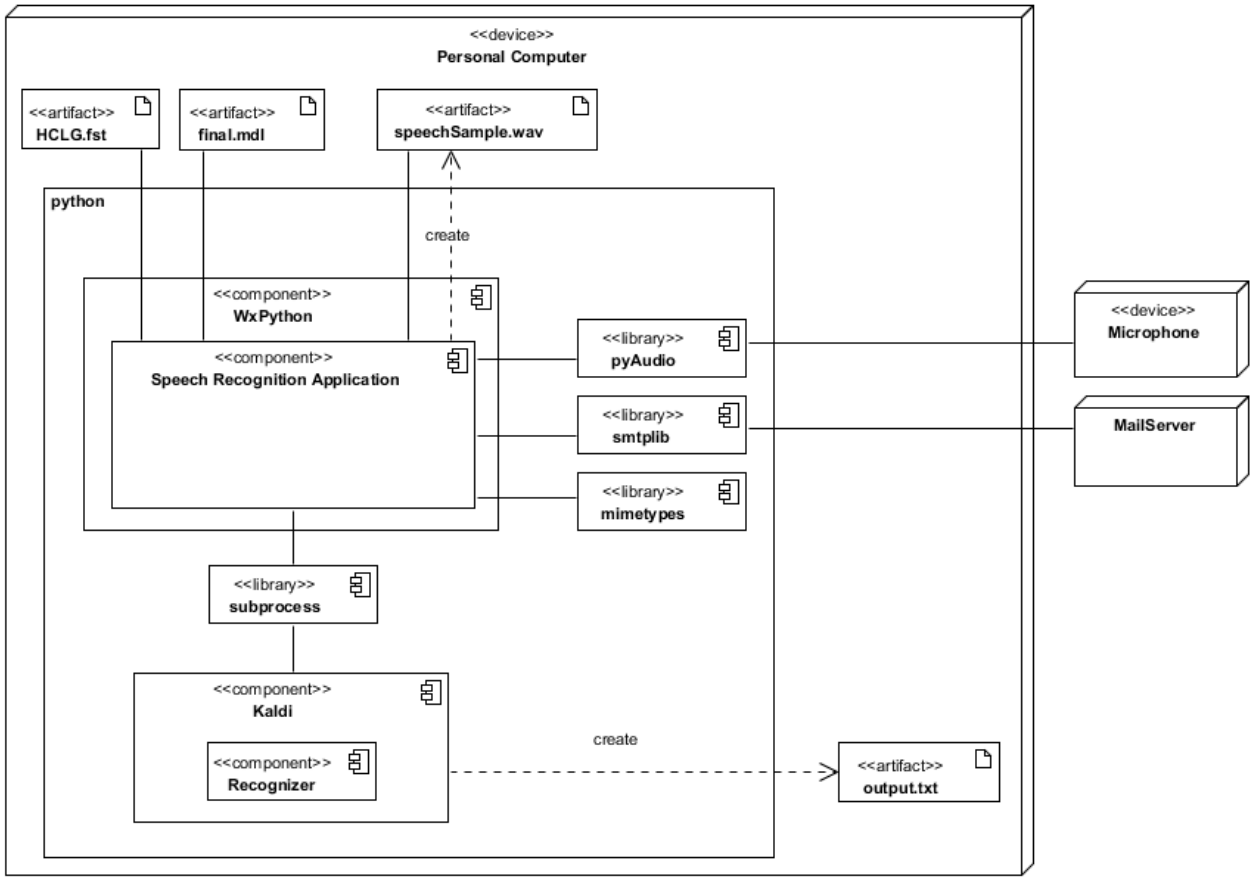


Figure 4.4 Deployment diagram for recognizing speech sample

Figure 4.4 shows the deployment diagram design for the recognizing speech sample scenario. The left side of the diagram shows the software components which perform as the input and the output of the SR application. The right side of the diagram shows which hardware it needs for operating.

5 Object Design

Object design defines a solution that should bridge the gap between analysis model and the hardware/software platform defined during system design. This includes precisely describing object and subsystem interfaces. (*64*) This section describes the key features of the Kaldi trainings. Currently Kaldi contains three parallel implementations for deep neural network (DNN) training. These trainings support deep neural network training which is done on top of the standard HMM/GMM training. Object and subsystem interfaces are designed more in detail in Figure 5.1.

5.1 Speech recognition process

Typical procedure of speech recognition in Kaldi consists of processing an input speech sample and identifying a sequence of words from it. The process is based on audio feature extraction implemented by a DNN or GMM and a search phase implemented with the Viterbi search algorithm.

5.2 Kaldi DNN implementations

The first implementation supports Restricted Boltzmann Machines (RBM) pretraining, stochastic gradient descent training using Nvidia graphics processing units (GPUs) and discriminative training such as boosted MMI and state level minimum Bayes risk (sMBR). (*76*) It is located in sub-directories *nnet/* and *nnetbin/* and is maintained by Karel Vesely. The second implementation supports parallel training on multiple CPUs. Instead of RBP pre-training, the layer-wise backpropagation is used. It was originally written to support parallel training on multiple CPUs. (*69*) It is located in code sub-directories nnet2/ and nnet2bin/ and is primarily maintained by Daniel Povey. Regarding these two setups, Karel's setup gives generally a better result, but it only supports training on a single GPU card. It supports also training on CPU, but that is very slow. Dan's setup supports training on multiple GPU's or multiple CPUs and they do not need to be on the same machine. The reasons for the performance difference is unclear since there are many differences in the implementation of each setup. (*77*)

5.2.1 Karel's DNN training implementation

The implementation starts with Mel-Frequency Cepstral Coefficients (MFCC), Linear Discriminant Analysis (LDA), Maximum Likelihood Linear Transform (MLLT), feature space Maximum Likelihood Linear Regression (fMLLR) with Cepstral Mean Normalization (CMN) features. (*78*) 40-dimensional features that are now obtained, are stored to the disk in order to simplify the training scripts. Whole DNN training is run in a single GPU using CUDA (Compute Unified Device Architecture).

Next step is pre-training phase where the training algorithm is Contrastive Divergence with 1-step Markov Chain sampling. The training is unsupervised, so it is sufficient to provide single data-directory with input features. RBM is trained with Gaussian-Bernoullli units. After training an RBM on the data, the inferred states of the hidden units are used for training another RBM that learns to model the dependencies between the hidden units of the first RBM. (*57*) This is repeated until a complex statistical structure in data is created. The RBMs in a stack can be combined to produce a multi-layer generative model called a deep belief net (DBN). Once the unsupervised pre-training is done, the obtained parameters are used as the initial values of the DNN. In sequence-discriminative training phase the neural network is trained to classify correctly the whole sentences, which is done by Stohastic Gradient Descent (SGD) by using the state-level minimum Bayes risk (sMBR) criterion. [8]

5.3 Weighted Finite State Transducers

Speech recognition initially uses hidden Markov models (HMMs) to define the relationship between hidden and observable variables, hidden variables being sequences of words, and the observable ones being the acoustic features. But this increased the complexity of the models. A more recent approach to reduce the complexity of the HMM are the weighted finite state transducers (WFST) state machines. (*79*) WFST is an automaton whose state transitions are labeled with the input, output and weight. The representation of the acoustic model with a WFST state machine is made up of different information sources which at the same time are represented with a WFST. Usually, the model is made up of four WFST: an HMM (H) which encodes the pronunciation of each phone, a context dependency (C) which establishes the context rules between each phone, a lexicon (L) defines the relations between words and phones, and finally a grammar (G) for the language model. The final model is the composition of all of them (H o C o L o G). (*79*) A WFST state machine has arcs with five attributes: weight, source and destination states, and input and output labels. The WFST of an SR is constructed offline and is used by the Viterbi search algorithm for the translations of phonemes to words. The weights of the WFST represent probabilities between 0 and 1 as using floating points is not reliable due to floating point underflow problem. To prevent this problem, probabilities are represented in log-space, a side effect of this is that probabilities multiplications become additions. (*80*) Even though the WFST model reduces the complexity and lowers the redundancy of previous approaches using HMM, the resulting size is significantly big. For example, the WFST of Kaldi can contain more than 13 million states and more than 34 million arcs. (*80*) OpenFst is an open-source library for WFSTs, which Kaldi uses for designing models. OpenFst consist of a C++ template library with efficient WFST representations. It is also designed to be very efficient in time and space and to scale to very large problems. (*81*)

[8] Karel Vesely. (2017). Karel's DNN implementation. Retrieved from http://kaldi-asr.org/doc/dnn1.html. On date: 04.10.2017

5.4 Viterbi Search Algorithm

The Viterbi search algorithm finds the most likely sequence of hidden variables resulting from a sequence of observed variables on an HMM. (*82*) In the context of speech recognition search algorithm is responsible for the translation of a sequence of phonemes, obtained from the processing of the audio in the DNN phase, to a sequence of words. The Viterbi search algorithm is hard to parallelize and, therefore, a software implementation cannot exploit all the parallel computing units of modern multi-core CPUs and many-core GPU. (*80*) The Viterbi search algorithm uses the information provided by the WFST state machine to determine the most likely sequence of words according to it. For every frame of audio, a given phoneme is detected. The Viterbi search algorithm uses the phoneme as an input label on the WFST state machine to expand its corresponding output labels and detect the most likely sequence of words.

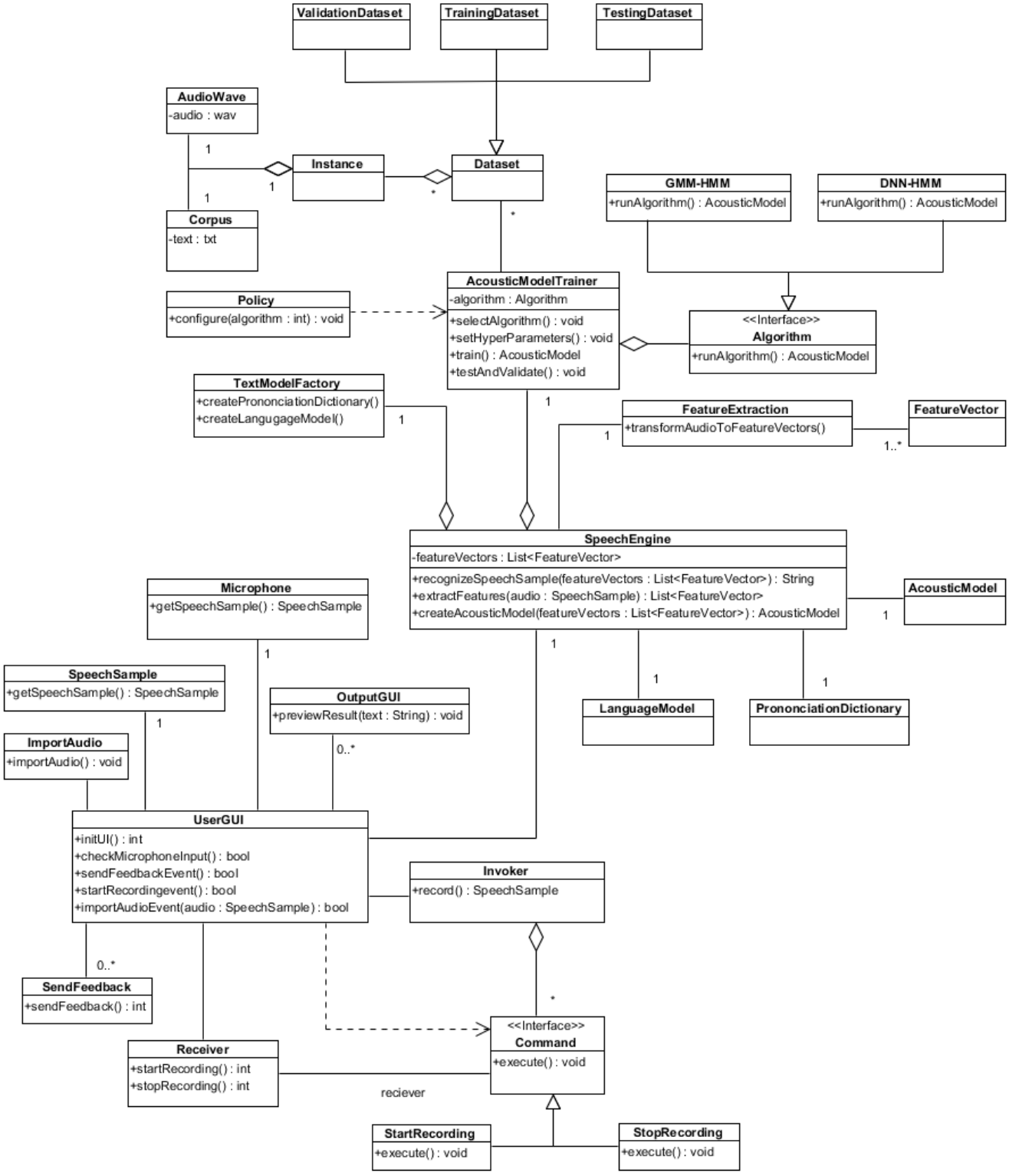


Figure 5.1 Class diagram for Kaldi SR

6 Implementation

This chapter shows how to download and install Kaldi. Also, an experiment with Kaldi is described, including which data was used, how the data was trained and finally used for recognizing a speech sample. The toolkits which are used for the implementation are also described in this chapter.

6.1 Software requirements

The bare minimum computing environment to run Kaldi is any Unix-like environment. Running Kaldi on a single machine makes everything slower. Ideal computing environment is a cluster of Linux machines running Sun GridEngine (SGE), with access to shared directories via some network filesystem. In ideal case, some computers on the grid have Nvidia GPUs.

6.2 Software packages installed by Kaldi

The following tools and libraries come with installation scripts in the tools/ directory:

- OpenFst is a library for constructing, combining, optimizing, and searching weighted finite-state transducers (FSTs)
- IRSTLM is a language modeling toolkit. Some of the example scripts require it but it is not tightly integrated with Kaldi. It can convert any ARPA format language model to an FST.
- SRILM - is used by some example scripts. It is generally a better and more complete language modeling toolkit than IRSTLM. The only drawback is the license, which is not free for commercial use. For downloading the source code it requires human interaction on the download page. It requires the name of the user.
- sph2pipe is used for converting SPH format files into other formats such as wav.
- ATLAS is a linear algebra library. If ATLAS it not already available on the system then it can be compiled as long as the system does not have CPU throttling enabled.
- CLAPACK is also a linear algebra library. This is useful only on systems where ATLAS is not available.
- OpenBLAS is an alternative to ATLAS or CLAPACK. The scripts do not use it by default, but installation scripts are provided for installation

6.3 Downloading Kaldi

The repository of Kaldi is located at GitHub. After the terminal is directed to where Kaldi should be installed, the download can be done with the following commands:

```
git clone https://github.com/kaldi-asr/kaldi.git kaldi --origin upstream
```

For the latest updates of Kaldi next command should be run:

```
cd kaldi
git pull
```

6.4 Installing Kaldi

Downloaded repository from Git has two important folders: src/ and tools/. These folders include scripts for installing Kaldi. First step is running scripts from the tools/ folder.

```
cd tools
```

Next step is checking the prerequisites for Kaldi. It can be done with command:

```
extras/check_dependencies.sh
```

This script checks if there are any system-level installations that need to be done. The output on the terminal informs if something is missing. If the system default C++ compiler is not supported then a check with another compiler can be done by setting the CXX environment variable.

```
CXX=g++-4.8 extras/check_dependencies.sh
```

After dependencies are checked, next step is to install ATLAS headers, OpenFst, SCTK and sph2pipe. If the computer has multiple CPUs then the command can do a parallel build by supplying the *-j* option to command and the number of the CPUs, otherwise the option can be left out.

```
make -j 4
```

To be able to use the recognition scripts, PortAudio [9]library needs to be installed.

```
./install_portaudio.sh
```

After all the installation steps from above are complete, only then other scripts should be installed. Next step is installing scripts from the folder called /src.

```
cd src
```

If the computer has multiple CPUs then the scripts from /src can also be run in parallel. To use 8 CPUs, the commands would be:

```
./configure –shared
make depend -j 8
make -j 8
```

6.5 Dataset

Dataset for the training is a corpus which has been created from TED (Technology, Entertainment, Design) conferences, called TED-LIUM corpus. (*22*) This corpus is composed of a total of 774 talks, representing 118 hours of speech: 82 hours of male and 36 hours of female. The corpus is English-language transcription, sampled at 16 kHz and the language model are obtained from Open Speech and Language Resources (OpenSLR). OpenSLR is a web-page devoted to hosting speech and language resources, such as training corpora for speech recognition. [10]

6.6 Acoustic Model Training

Kaldi has an example how to train the TED-LIUM dataset and build a deep neural network (DNN) based on on state-level minimum Bayes risk (sMBR). The example is in folder *egs/tedlium/s5.* Training can be computationally expensive. To identify pauses in data, forced alignment needs to be performed. The idea is to train a preliminary set of models with no pauses marked in transcript, and then with forced alignments the models are updated with location of silences. To make training and alignment more efficient, the dataset is split into smaller chunks and processed in parallel. The number of jobs *nj* is specified in the training and alignment steps. The corpus and

[9] Retrieved from: http://www.portaudio.com/ . On date 20.10.2017
[10] Daniel Povey. About OpenSLR. Retrieved from: http://www.openslr.org/. On date 06.10.2017

language model are downloaded using *wget* command from OpenSLR. This is done in script *download_data.sh* which is located in the folder */local*.

```
local/download_data.sh
```

The training and testing datasets are prepared, and also extra files as utt2spk, spk2utt which are needed as input files. Utt2spk and spk2utt are files which map each utterance id to a speaker id.

```
local/prepare_data.sh
```

For the training is also needed a directory which contains information about silence, non-silence phonemes and the phonetic dictionary which includes all the words that can appear in the output. These are created using the information from the language model that is downloaded from the OpenSLR web-site. This script can be found in the folder */utils*.

```
utils/prepare_lang.sh
local/prepare_lm.sh
```

The following scripts extracts the MFCC acoustic features and compute the cepstral mean and variance normalization (CMVN) stats. The MFCC computation works with the number of frames in the file. For each frame it extracts the data, multiplies it by a windowing function, works out the energy at each point, does the FFT and computes the power spectrum. Finally, the energy is computed, and coefficients are specified.

```
steps/make_mfcc.sh
steps/compute_cmvn_stats.sh
```

Training monophone models are the first part of the training procedure. This model is used to bootstrap training for later models. For the training of this model some arguments are required.

```
steps/train_mono.sh –nj 20 –cmd “$train_cmd” data/train data/lang exp/mono
```

Location of the acoustic data	data/train
Location of the lexicon	data/lang
Destination directory for the model	exp/currentmodel
Number of jobs	--nj
Defines which machine should handle the processing (default is run.ph)	"$train_cmd"

Table 7 Parameters for script train_mono.sh

Just like the training scripts, the alignment scripts have the same argument structure.

```
steps/align_si.sh
```

Training a triphone model starts with delta and delta-delta features. These scripts need additional arguments for the number of leaves or HMM states on the decision tree and the number of Gaussians distributed across the leaves. The exact number of leaves and Gaussians is often decided based on heuristics. The numbers largely depend on the amount of data, number of phonetic questions, and goal of the model.

```
steps/train_deltas.sh –cmd "$train_cmd" 2500 30000 data/train data/lang
exp/mono_ali exp/tri1

steps/train_lda_mllt.sh –cmd "$train_cmd" 4000 50000 data/train
data/lang exp/tri1_ali exp/tri2

steps/train_sat.sh –cmd "$train_cmd" 5000 100000 data/train data/lang
exp/tri2_ali exp/tri3

steps/train_mmi.sh –cmd "$train_cmd" 5000 100000 data/train data/lang
exp/tri3_denlets exp/tri3_mmi_b0.1
```

Location of the acoustic data	data/train
Location of the lexicon	data/lang
Source directory for the model	exp/lastmodel
Destination directory for the model	exp/currentmodel
Number of jobs	--nj
Defines which machine should handle the processing (default is run.ph)	"$train_cmd"

Table 8 Parameters for delta training scripts

After each training the triphone models are aligned with the data. Models with LDA-MLLT, SAT and MMI triphones are built on top of fMLLR features therefore they are aligned with next script:

```
steps/align_fmllr.sh
```

With the SAT GMM system and corresponding fMLLR transformations, Karel's version of deep neural network code can be run with script:

```
local/nnet/run_dnn.sh
```

This script is split into several stages. For a more simplified training the fMLLR features are stored to the disk.

```
steps/nnet/make_fmllr_feats.sh
```

RBM pre-training is then implemented with the training algorithm Contrastive Divergence with 1-step of Markov Chain Monte Carlo sampling (CD-1). The first RBM has Gaussian-Bernoulli units and the following RBMs have Bernoulli-Bernoulli units. It creates a stack of RBMs.

```
steps/nnet/pretrain_dbn.sh
```

Frame cross-entropy training trains a DNN which classifies frames into triphone states. This is the main neural network script. It requires CUDA, otherwise the script exits. The neural network is stored into *exp-dir/nnet*.

```
steps/nnet/train.sh data/train data/dev lang-dir ali-train \
ali-dev exp-dir
```

Location of the acoustic data	data/train, data/dev
Important only when using LDA feature-transform	lang-dir
Location of training targets	ali-train, ali-dev
Location of the output	exp-dir

SMBR training trains the neural network to optimize the whole sentence. The training is done by Stochastic Gradient Descent with pre-utterance updates.

```
steps/nnet/train_mpe.sh
```

6.7 Speech Recognition Application

The speech recognition (SR) application is implemented in Linux 16.04. The user interface for SR application is shown in Figure 6.1. It is implemented using wxPython. WxPython [11] is an open source GUI toolkit for the Python language, which is used for development of applications that need a user interface.

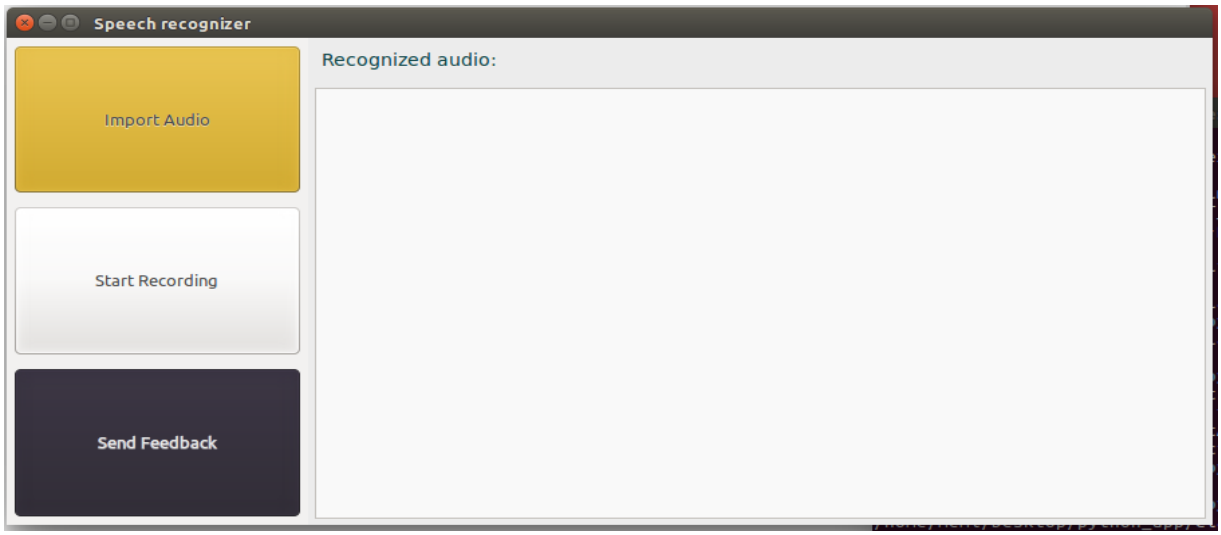


Figure 6.1 User Interface of Speech Recognition Application

The user can import a speech sample by clicking on button “Import Audio”. This button triggers opening of a dialog box, which allows an audio to be chosen. The user chooses a speech sample and sends it to the speech recognizer.

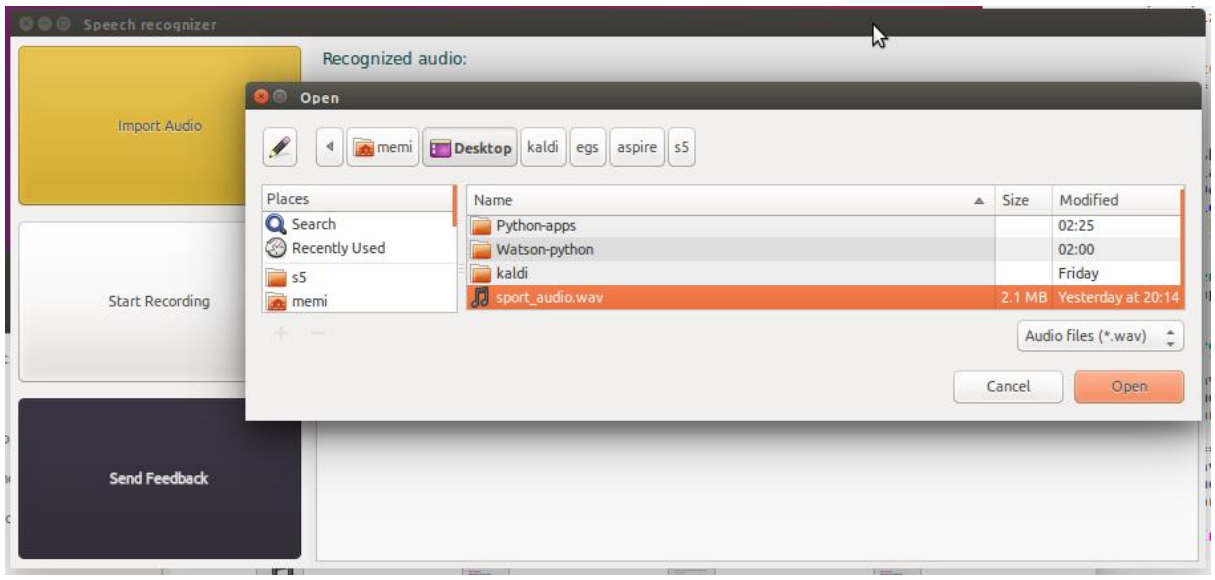


Figure 6.2 Dialog box of Speech Recognition Application

[11] Retrieved from: https://www.wxpython.org/. On date: 29.10.2017

After the speech sample is chosen, speech recognition is triggered with a command shown in Table 9. Another way to import audio is available with a button “Start Recording”. Triggering this event checks if the microphone is available and starts recording the audio, and the button changes the label to “Stop Recording” as shown on Figure 6.3. To stop recording the audio from microphone and start the recognition process, the button “Stop Recording” needs to be activated.

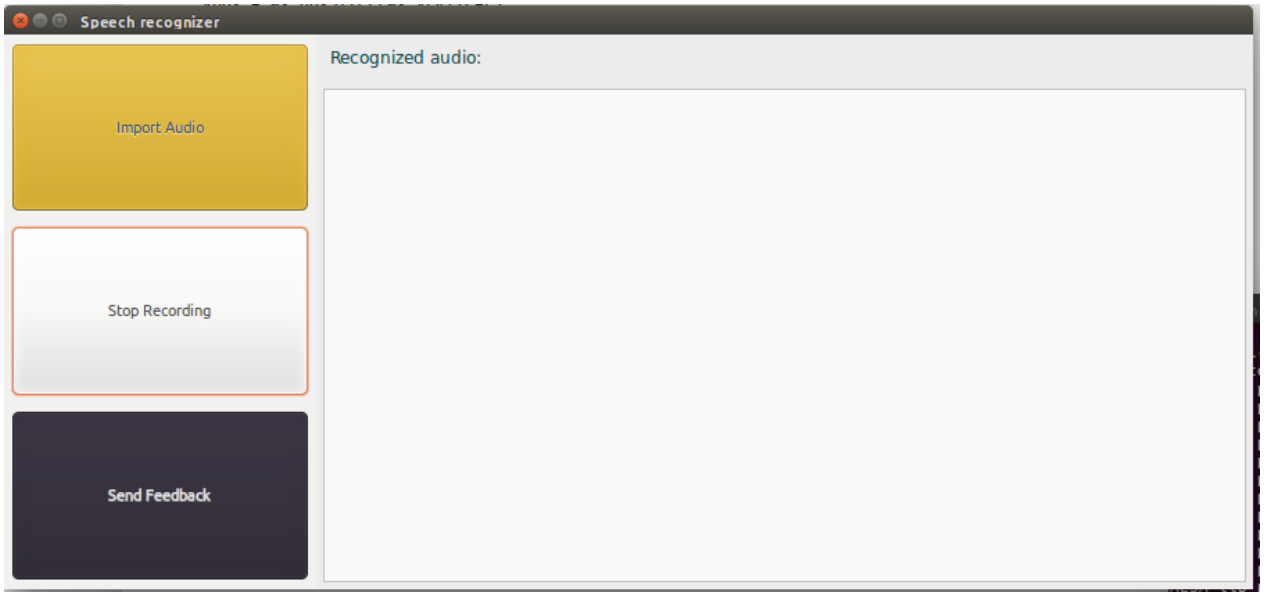


Figure 6.3 User interface of SR application

After the audio is imported to the application, the Kaldi engine is invoked and the command from Table 9 is called. The result of the speech recognition is shown on the user interface as displayed on Figure 6.4.

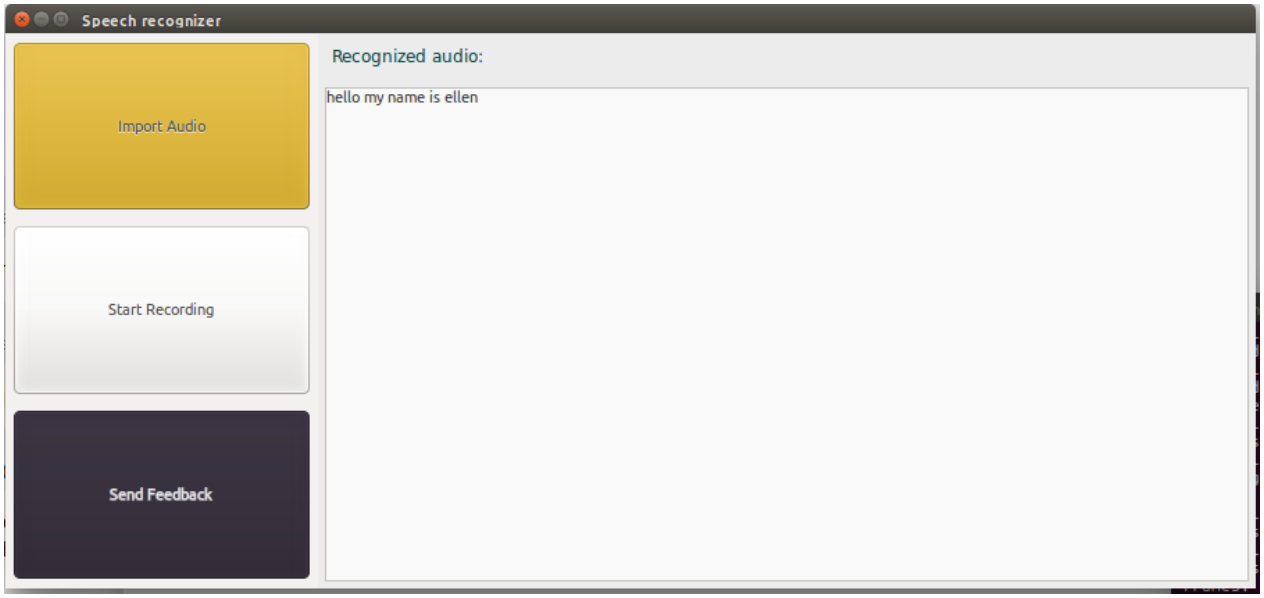


Figure 6.4 Result shown in SR application

```
online2-wav-nnet2-latgen-faster --do-endpointing=false \
    --online=false \
    --config=conf/decoding.conf \
    --max-active=7000 --beam=15.0 --lattice-beam=6.0 \
    --acoustic-scale=0.1 --word-symbol-table=graph/words.txt \
   exp/final.mdl exp/graph/HCLG.fst "ark:echo utterance-id1 \
   utterance-id1|" "scp:echo utterance-id1 speech_sample.wav|" \
   "ark,t:output.lat"
```

Minimum and maximum decoding run time factor	--lattice-beam
Maximum number of states that can be active	--max-active
Value for beam pruning	--beam
Value for lattice beam pruning	--lattice-beam
Acoustic model	exp/final.mdl
Weighted Finite State Transducer	exp/graph/HCLG.fst
File that contains the output result	"ark,t:output.lat"
Speech sample	speech_sample.wav

Table 9 Parameters for recognition speech command

If the speech application cannot recognize some utterances uploaded from the user, then the user can upload a feedback in the application which sends the feedback to the speech recognition trainer.

The SR application includes also a command pattern, which is used to trigger the appropriate Command for the user. Small part of the implementation of the command pattern is shown in Figure 6.5.

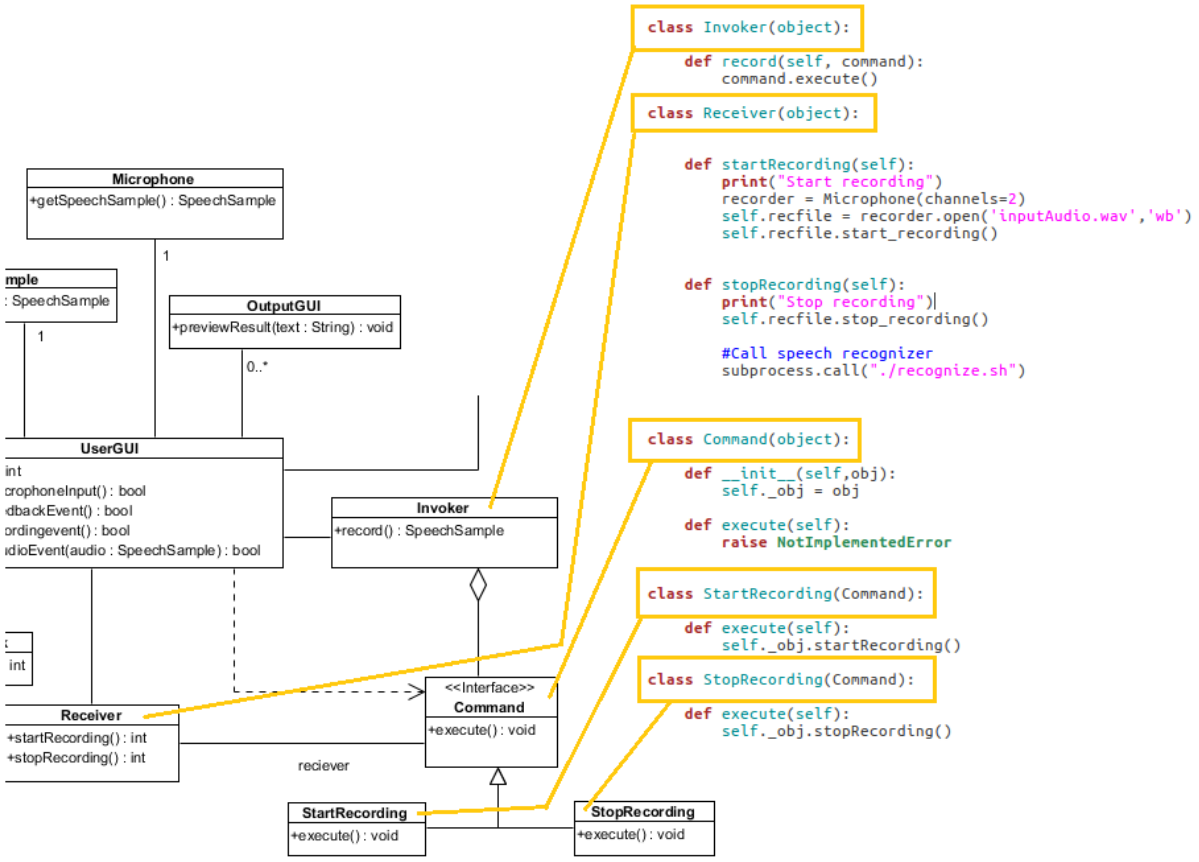


Figure 6.5 Implementation of command pattern in SR application

6.8 Speech Recognition Results

Because of limited resources of GPU computation, the results are illustrating a partly trained acoustic model. This acoustic model was trained until model boosting with maximum mutual information (MMI) objective function (*83*), which estimates the probabilities in the hidden Markov model. It maximizes the mutual information between an acoustic observation sequence and the corresponding word sequence. (*84*) The results are taken from a detailed overall report in file *exp/tri3_mmi_b0.1/decode_test_it4/score_17_1.0/ctm.filt.* This file describes the performance of the trained acoustic model, which is tested with the testing dataset.

Sentence recognition performance is shown in Table 10.

Total sentences: 1155	Number of sentences	Percent error
Sentences with error	1025	88.7 %
Sentences with substitutions	928	80.3 %
Sentences with deletions	698	60.4 %
Sentences with insertions	303	26.2 %

Table 10 Sentence recognition performance for TED-LIUM

Word recognition performance is shown in Table 11.

Total words: 27500	Number of words	Percent error
Percent total error	5114	18.6 %
Percent correct	22786	82.9 %
Percent substitution	3026	11.0 %
Percent deletions	1688	6.1 %
Percent insertions	400	1.5 %
Percent word accuracy	81.4%	

Table 11 Word recognition performance for TED-LIUM

Commonly confused words with biggest substitution totals are displayed on Figure 6.6, where word “in” was confused with 64 times with words as “and”, “on”, “of”, “been” and “him”. Other example is word “know” which was confused 33 times with words as “now”, “and”, “no”, “go”, “your” and “not”.

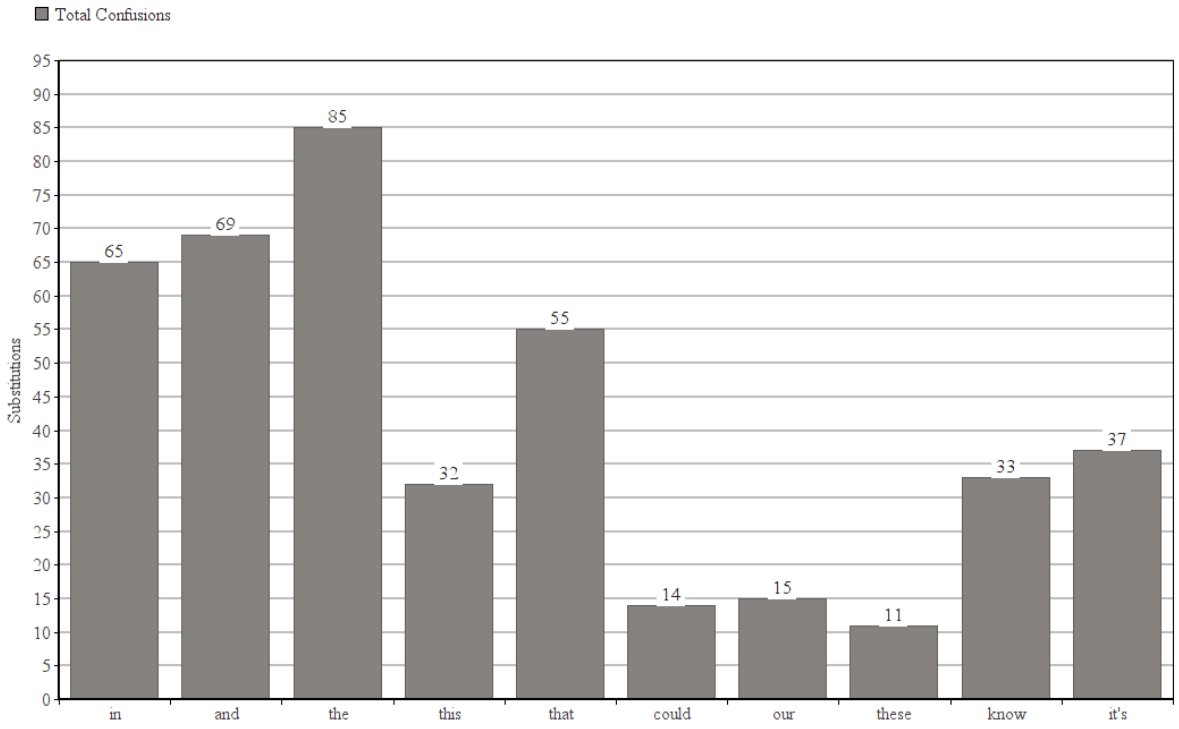


Figure 6.6 Chart of commonly confused words

To tune the accuracy of the acoustic model, the acoustic model needs to be retrained. Corpus TED-LIUM consists of audio and text files. Audio files are of format sphere waveform (.sph), and text files are of format segment time mark file (.stm). The segment time mark file consists of a concatenation of text segment records from a waveform file. Each record is separated by a newline and contains: the waveform's filename and channel identifier, the talker’s id, begin and

end times (in seconds), optional subset label and the text for the segment. [12] After the speech samples are recorded, they are converted to .sph format.

```
sox -t wav speechSample1.wav -t sph speechSample1.sph
```

For each sphere waveform file, a segment time mark file is manually made. The recorded data is split up for training and testing. Important to note is that if the corpus has new instances, which are not available in the acoustic model, then also the language and phonetic model need to be updated. Language model can be updated with SRILM tool, by merging the new corpus with the old corpus and creating a new language model.

```
ngram-count -text corpus.txt -order 3 -vocab words.txt -kndiscount1 \
-interpolate -lm lm.arpa
```

Corpus from which the N-gram is generated	-text
Maximal order of N-gram to count	-order
Vocabulary file	-vocab
Kneser-Ney smoothing of order 1	-kndiscount1
Output language model with order n	-lm

Table 12 Parameters for ngram-count command

Finally, a new acoustic model can be trained using the scripts from section 6.6.

[12] Retrieved from http://www1.icsi.berkeley.edu/Speech/docs/sctk-1.2/infmts.htm. On Date 10.10.2017

7 Comparison

In this chapter a cloud-based system, IBM Watson is introduced and a deep neural network (DNN), which is trained with Kaldi, is introduced. Next chapters also describe how IBM Watson works. Finally, the comparison between cloud-based and offline SR system is presented.

7.1 Speech Recognition with Kaldi Engine

There a few acoustic models which are trained with Kaldi and are open sourced for everyone to use. [13] One of the model which works well with generic English speech recognition is the Automatic Speech Recognition In Reverberant Environments (ASpIRE) model. To test the acoustic model, a speech sample is obtained from an audio book "Emma" from Jane Austin [14] and is given as an input to the speech recognizer. Parts of the first chapter of the book are used for this experiment. The speech sample was recorded in a quiet environment with microphone. The corpus of the recorded speech sample has a total of 2060 words or 141 sentences, from which 679 words are unique. For the experiment the downloaded audio is imported into the SR application which was implemented in chapter 0.

Sentence recognition performance is shown in Table 13.

Total sentences: 141	Number of sentences	Percent error
Sentences with error	135	95.7 %
Sentences with substitutions	39	27.7 %
Sentences with deletions	15	10.6 %
Sentences with insertions	90	63.8 %

Table 13 Sentence recognition performance for ASpIRE with audio from book "Emma"

Word recognition performance is shown in Table 14.

Total words: 2060	Number of words	Percent error
Percent total error	205	10 %
Percent correct	1855	90 %
Percent substitution	51	2.5 %
Percent deletions	48	2.3 %
Percent insertions	106	5.1 %
Percent word accuracy	84.9 %	

Table 14 Word recognition performance for ASpIRE with audio from book "Emma"

[13] Retrieved from: http://kaldi-asr.org/models.html. On Date: 28.10.2017

[14] Retrieved from: http://www.openculture.com/free_ebooks. On Date: 28.10.2017

7.2 IBM Watson API

IBM Watson provides an API that enables speech recognition capabilities to an application. A client library that is provided for Node.js [15] can be downloaded.

```
npm install watson-developer-cloud
```

To use IBM Watson, authentication is needed by providing the username and password. The API uses HTTP basic authentication.

```
var SpeechToTextV1 = require('watson-developer-cloud/speech-to-
text/v1');
var speech_to_text = new SpeechToTextV1 ({
  username: '{username}',
  password: '{password}'
});
```

By replacing the username and password with service credentials, Watson provides services like speech to text, text to speech, visual recognition, natural language understanding and many other services. IBM Watson uses 90 IBM Power 750 servers and can operate at 80 Teraflops. [16] Which means Watson has a sufficient power to train networks with complex data.

7.3 IBM Watson Speech to Text Service

Speech technology has been around for long time, but current tools do not work for everyone. IBM Watson is trying to change that.[17] Most cloud-based tools like Siri, Alexa focus on transcription of short messages and search terms for high quality audio. Other types of audio like phone calls meetings, broadcasts are ignored because it is not easy to train such speech recognizers. IBM Watson develops a speech to text, where they claim that they use statistical modeling techniques developed over decades which are also refined by ideas from cognitive computing to transcribe high and lower quality audios. And not only from high quality microphones, but also from wide variety of source materials. [18] Watson determines which words are the most accurate for an utterance, and presents them with confidence scores and other meta data.

[15] Retrieved from https://nodejs.org/en/. On Date 29.10.2017
[16] Retrieved from http://static.usenix.org/event/lisa11/tech/slides/perrone.pdf. On Date 15.10.2017
[17] Retrieved from https://www.youtube.com/watch?v=_Xcmh1LQB9I. On Date 13.10.2017
[18] Referenced from https://www.youtube.com/watch?v=JWnLgZ58zsw. On Date 13.10.2017

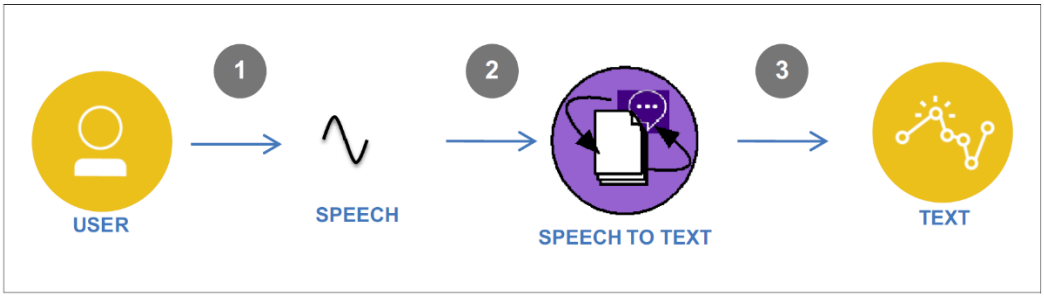


Figure 7.1 Watson speech to text flow

The flow of the speech to text service by Watson is shown on Figure 7.1. It shows the following flow:

1. User specifies an audio file, which can be recorded live or be previously prepared.
2. The speech is sent to the Speech to Text Service for processing.
3. The Speech to Text service generates the text based on the audio file.

The speech to text flow for cloud-based system is simpler than the speech to text flow by offline SR systems as shown on Figure 7.1.

After authentication, other API calls to IBM Watson can be made over that connection. Request and responses are enabled over a single TCP connection. In a speech to text request parameters can be set.

```
var params = {
  model: 'en-US_BroadbandModel',
  content_type: 'audio/mp3,
  'interim_results': true,
  'max_alternatives': 3,
  'word_confidence': false,
  timestamps: false,
  keywords: ['book', 'Emma', 'sister'],
  'keywords_threshold': 0.5
};
```

Parameter	Description
model	The identifier of the model to be used for the recognition request
content_type	The audio format (MIME type) of the audio
interim_results	Indicates whether the service is to return interim results. If *true*, interim results are returned as a stream of JSON objects
max_alternatives	The maximum number of alternative transcripts to be returned
word_confidence	Indicates whether a confidence measure in the range of 0 to 1 is returned for each word
timestamps	Indicates whether time alignment is returned for each word
keywords	A list of keywords to spot in the audio
keywords_treshold	A confidence value that is the lower bound for spotting a keyword

Table 15 Parameters for stream creation

With previous parameters a stream is created.

```
var recognizeStream = speech_to_text.createRecognizeStream(params);
```

After stream creation, an audio can be piped to the recognizer.

```
var fs = require('fs');
fs.createReadStream('audio.mp3').pipe(recognizeStream);
```

Sentence recognition performance is shown in Table 136.

Total sentences: 141	Number of sentences	Percent error
Sentences with error	28	95.7 %
Sentences with substitutions	23	67.3 %
Sentences with deletions	21	10.6 %
Sentences with insertions	10	27.7 %

Table 16 Sentence recognition performance for IBM Watson with audio from book "Emma"

Word recognition performance is shown in Table 147.

Total words: 2060	Number of words	Percent error
Percent total error	63	3.1 %
Percent correct	1997	96.9 %
Percent substitution	22	1.1 %
Percent deletions	24	1.2 %
Percent insertions	17	0.8 %
Percent word accuracy	96.1%	

Table 17 Word recognition performance for IBM Watson with audio from book "Emma"

Watson's performance is shown in Table 17. Since it trained models with a thousand of hours of data, it is hard to compare it to acoustic models trained with Kaldi. which trained are with only hundred hours of data. With this example Watson outperforms Kaldi, but these results are straightforward because Watson uses more hours of data for training. Total error by Watson trained models is 3.1%, where the error by Kaldi trained models is 10%. Kaldi and IBM Watson allow to add words and tune the accuracy of acoustic models. But IBM Watson can recognize speech samples only over web sockets, which does not provide always privacy and security. For privacy, offline SR systems like Kaldi are the solutions to choose.

8 Conclusion

In this master's thesis, a description of training an acoustic model has been provided. With that illustration an acoustic model has been trained. Using the functional and non-functional requirements a speech recognition application has been implemented, which uses the previous trained acoustic model to recognize speech samples. Problems with the accuracy and inefficiency of the current acoustic model of offline SR, made us think how to use these systems more efficiently. The accuracy of a trained acoustic model is illustrated in charts for a few chosen words. The words are chosen based on the WER of each word, where words with the highest WER are chosen. Because of limited resources, only the steps for training and tuning a DNN model are explained. To tune the recognition accuracy of the chosen words, a new acoustic model is created. New speech samples should be made and for each sample a corpus should be manually created.

Finally, a comparison between cloud-based SR system and offline SR systems is made. For cloud - based SR system IBM Watson is chosen. The conclusion is that Watson outperforms Kaldi, but these results are straightforward because Watson is trained with thousand hours of data for years, while Kaldi SR trained the acoustic model with a few hundred hours.

9 Future Works

The speech recognition application does not implement real-time recognition of speech sample. This was not the topic of this thesis. Possible addition to the SR application is the real-time recognition. The application should be able to start recognizing speech and giving results on the user interface as soon as the user starts speaking into the microphone. This should be possible with updating some of the recognizing scripts in Kaldi.

Another future work would be using proxy pattern to allow the speech recognition system to use the cloud-based SR system when internet is available, or to use offline SR system when internet is not available. Additional idea is to give the user the freedom in the SR application to choose which SR system to use.

Watson outperforms Kaldi, but these results are straightforward because Watson is trained with thousand hours of data than Kaldi SR. An additional approach would be to train Kaldi with more data than used for this comparison. I suggest this for a possible future work. More accurate results could be achieved, and the performance of Kaldi could match with the performance results from Watson.

10 Bibliography

1. W. Holmes, *Speech synthesis and recognition*. (CRC press, 2001), pp. 109-110.
2. K. Davis, R. Biddulph, S. Balashek, Automatic recognition of spoken digits. *The Journal of the Acoustical Society of America* **24**, 637-642 (1952).
3. J. K. Baker, Stochastic modeling as a means of automatic speech recognition. 1-10 (1975).
4. A. B. Poritz, Hidden Markov models: A guided tour. *Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on*, 7-13 (1988).
5. F. Jelinek, Continuous speech recognition by statistical methods. *Proceedings of the IEEE* **64**, 532-556 (1976).
6. L. E. Baum, T. Petrie, in *The annals of mathematical statistics*. (1966), vol. 37, pp. 1554-1563.
7. L. R. Rabiner, B.-H. Juang, *Fundamentals of speech recognition*. (1993), pp. 321-362.
8. L. E. Baum, An Inequality and Associated Maximization Thechnique in Statistical Estimation for Probabilistic Functions of Markov Process. *Inequalities* **3**, 1-8 (1972).
9. A. P. Dempster, N. M. Laird, D. B. Rubin, Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society. Series B (methodological)*, 1-38 (1977).
10. B. James, The DRAGON system--An overview. *IEEE Transactions on Acoustics, Speech, and Signal Processing* **23**, 24-29 (1975).
11. J. Wilpon, D. Roe, AT&T telephone network applications of speech recognition. *Proc. COST232 Workshop, Rome, Italy*, (1992).
12. J. G. Wilpon, in *Communications Between Humans and Machines*. (National Academy, Washington, 1994), pp. 280-310.
13. L. R. R. J.G. Wilpon, C.H. Lee, E.R. Goldman, in *IEEE Trans. Acoust. Speech Signal Process*. (1990), pp. 1870-1878.
14. F. Jelinek, L. Bahl, R. Mercer, Design of a linguistic statistical decoder for the recognition of continuous speech. *IEEE Transactions on Information Theory* **21**, 250-256 (1975).
15. N. R. Brown, A. M. Vosburgh, Evaluating the accuracy of a large-vocabulary speech recognition system. *Proceedings of the Human Factors Society Annual Meeting* **33**, 296-300 (1989).
16. A. Averbuch *et al.*, Experiments with the TANGORA 20,000 word speech recognizer. *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'87.* **12**, 701-704 (1987).
17. J. Baker, The DRAGON system--An overview. *IEEE Transactions on Acoustics, Speech, and Signal Processing* **23**, 24-29 (1975).
18. S. F. Chen, J. Goodman, An empirical study of smoothing techniques for language modeling. *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, 310-318 (1996).
19. D. Jurafsky, J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. (MIT Press, 2000), pp. 83-365.

20. B.-H. Juang, S. Levinson, M. Sondhi, Maximum likelihood estimation for multivariate mixture observations of Markov chains (corresp.). *IEEE Transactions on Information Theory* **32**, 307-309 (1986).
21. H. W. Sorenson, D. L. Alspach, Recursive Bayesian estimation using Gaussian sums. *Automatica* **7**, 465-479 (1971).
22. A. Rousseau, P. Deléglise, Y. Esteve, TED-LIUM: an Automatic Speech Recognition dedicated corpus. *LREC*, 125-129 (2012).
23. F. Rosenblatt, The perceptron: A probabilistic model. *Psychological review* **65**, 386-408 (1958).
24. B. Widrow, M. Hoff, August IRE WESCON Convention Record 1960. *Part* **4**, 96-104.
25. W. S. McCulloch, W. Pitts, A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics* **5**, 115-133 (1943).
26. F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review* **65**, 386-389 (1958).
27. F. Rosenblatt, The perceptron, a perceiving and recognizing automaton Project Para. 460- (1957).
28. D. O. Hebb, *The organization of behavior: A neuropsychological approach*. (John Wiley & Sons, 1949), pp. 17-38.
29. M. Minsky, S. Papert, Perceptrons. (1969).
30. D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning representations by back-propagating errors. *Cognitive modeling* **5**, 213 - 223 (1988).
31. A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, K. J. Lang, Phoneme recognition using time-delay neural networks. *IEEE transactions on acoustics, speech, and signal processing* **37**, 328-339 (1989).
32. A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, K. Lang, Phoneme recognition: neural networks vs. hidden Markov models. *Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on*, 107-110 (1988).
33. S. Hochreiter, J. Schmidhuber, Long short-term memory. *Neural computation* **9**, 1735-1780 (1997).
34. S. Hochreiter, Untersuchungen zu dynamischen neuronalen Netzen. *Diploma, Technische Universität München* **91**, 11-21 (1991).
35. C. Poultney, S. Chopra, Y. L. Cun, Efficient learning of sparse representations with an energy-based model. *Advances in neural information processing systems*, 1137-1144 (2007).
36. G. E. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets. *Neural computation* **18**, 1527-1554 (2006).
37. Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 153-160 (2007).
38. O. Delalleau, Y. Bengio, Shallow vs. deep sum-product networks. *Advances in Neural Information Processing Systems*, 666-674 (2011).
39. G. F. Montúfar, Universal approximation depth and errors of narrow belief networks with discrete units. *Neural computation* **26**, 1386-1407 (2014).
40. R. Raina, A. Madhavan, A. Y. Ng, Large-scale deep unsupervised learning using graphics processors. *Proceedings of the 26th annual international conference on machine learning*, 873-880 (2009).

41. X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 249-256 (2010).
42. V. Nair, G. E. Hinton, Rectified linear units improve restricted boltzmann machines. *Proceedings of the 27th international conference on machine learning (ICML-10)*, 807-814 (2010).
43. K.-F. Lee, H.-W. Hon, R. Reddy, An overview of the SPHINX speech recognition system. *IEEE Transactions on Acoustics, Speech, and Signal Processing* **38**, 35-45 (1990).
44. G. E. Steve Young, Mark Gales, Thomas Hain, Dan Kershaw,Xunying (Andrew) Liu, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey,, *The HTK*. (2006).
45. C. G. David Rybach, Georg Heigold, Björn Hoffmeister,, The RWTH aachen university open source speech recognition system. *Proc. INTERSPEECH*, 2111-2114 (2009).
46. A. G. Daniel Povey, Gilles Boulianne, Lukas Burget, The Kaldi speech recognition toolkit. *Proc. ASRU workshop*, 1-4 (2011).
47. D. P. Sharma, J. Atkins, Automatic speech recognition systems: challenges and recent implementation trends. *International Journal of Signal and Imaging Systems Engineering* **7**, 220-234 (2014).
48. L. R. Rabiner, R. W. Schafer, *Introduction to digital speech processing*. Foundations and Trends® in Signal Processing (2007), vol. 1, pp. 1-194.
49. B. Raj, R. M. Stern, Missing-feature approaches in speech recognition. *IEEE Signal Processing Magazine* **22**, 101-116 (2005).
50. A. Caranica, On the Design of an Automatic Speech Recognition System for Romanian Language. *Control Engineering and Applied Informatics* **18**, 65-76 (2016).
51. X. Wang, L. Li, An N-gram based Chinese syllable evaluation approach for speech recognition error detection. *Natural Language Processing and Knowledge Engineering, 2009. NLP-KE 2009. International Conference on*, 1-6 (2009).
52. A. G. Adami, Automatic speech recognition: From the beginning to the portuguese language. 2-6 (2010).
53. A. Senior, G. Heigold, M. Bacchiani, H. Liao, GMM-free DNN acoustic model training. *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, 5602-5606 (2014).
54. H. Van Hamme, F. Van Aelten, An adaptive-beam pruning technique for continuous speech recognition. *Spoken Language, 1996. ICSLP 96. Proceedings., Fourth International Conference on* **4**, 2083-2086 (1996).
55. L. Deng, X. Li, Machine learning paradigms for speech recognition: An overview. *IEEE Transactions on Audio, Speech, and Language Processing* **21**, 1060-1089 (2013).
56. A. Géron, *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. (O'Reilly Media, Sebastopol, 2017), pp. 7-31.
57. G. Hinton *et al.*, Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* **29**, 82-97 (2012).
58. F. Seide, G. Li, X. Chen, D. Yu, Feature engineering in context-dependent deep neural networks for conversational speech transcription. *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*, 24-29 (2011).

59. L. Li *et al.*, Hybrid Deep Neural Network--Hidden Markov Model (DNN-HMM) Based Speech Emotion Recognition. *Affective Computing and Intelligent Interaction (ACII), 2013 Humaine Association Conference on*, 312-317 (2013).
60. S. Jou, *Machine Learning: A Primer for Security*. Information Systems Security Association (2016), vol. 15, pp. 7-22.
61. G. Holmes, R. B. Kirkby, D. Bainbridge, Batch-incremental learning for mining data streams. 1-9 (2004).
62. I. H. Witten, E. Frank, M. A. Hall, C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. (Morgan Kaufmann, 2016), pp. 144-146.
63. T. Mishra, A. Ljolje, M. Gilbert, Predicting Human Perceived Accuracy of ASR Systems. *INTERSPEECH*, 1945-1948 (2011).
64. B. Bruegge, A. H. Dutoit, *Object-Oriented Software Engineering Using UML, Patterns and Java*. Upper Saddle River, NY: Prentice-Hall (2004), pp. 1-380.
65. I. Jacobson, G. Booch, J. Rumbaugh, J. Rumbaugh, G. Booch, *The unified software development process*. (Addison-wesley Reading, 1999), vol. 1, pp. 1-12.
66. D. Crankshaw *et al.*, Clipper: A Low-Latency Online Prediction Serving System. *NSDI*, 613-627 (2017).
67. M. M. E. Bisani, H. E. Secker-Walker, K. J. Basye, A. D. Rosen, Reducing speech recognition latency. 1-5 (2016).
68. C. Gaida *et al.*, Comparing open-source speech recognition toolkits. *Tech. Rep., DHBW Stuttgart*, 1-12 (2014).
69. D. Povey *et al.*, The Kaldi speech recognition toolkit. *IEEE 2011 workshop on automatic speech recognition and understanding*, 1-4 (2011).
70. W. Walker *et al.*, Sphinx-4: A flexible open source framework for speech recognition. 1-18 (2004).
71. S. Bangalore, R. Bell, D. A. Caseiro, M. Gilbert, P. Haffner, System and method for rapid customization of speech recognition models. 1-6 (2017).
72. J. Eder, G. Kappel, M. Schrefl, Coupling and cohesion in object-oriented systems. 6-22 (1994).
73. C. J. Tauro, R. K. Sahai, Object Persistence Techniques-A Study of Approaches, Benefits, Limits and Challenges. *International Journal of Computer Applications* **85**, 1-9 (2014).
74. J. Rasmussen, K. J. Vicente, Coping with human errors through system design: implications for ecological interface design. *International Journal of Man-Machine Studies* **31**, 517-534 (1989).
75. M. Talukdar, *Dictionary of Computer & Information Technology*. (Prabhat Prakashan, 2014).
76. K. Veselý, A. Ghoshal, L. Burget, D. Povey, Sequence-discriminative training of deep neural networks. *Interspeech*, 2345-2349 (2013).
77. P. Cosi, G. Paci, G. Sommavilla, F. Tesser, KALDI: YET ANOTHER ASR TOOLKIT? EXPERIMENTS ON ADULT AND CHILDREN ITALIAN SPEECH. *Proceedings of AISV 2015*, 430-436 (2015).
78. S. P. Rath, D. Povey, K. Veselý, J. Cernocký, Improved feature processing for deep neural networks. *Interspeech*, 109-113 (2013).
79. M. Mohri, F. Pereira, M. Riley, Weighted finite-state transducers in speech recognition. *Computer Speech & Language* **16**, 69-88 (2002).

80. R. Yazdani, A. Segura, J.-M. Arnau, A. Gonzalez, An ultra low-power hardware accelerator for automatic speech recognition. *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*, 1-12 (2016).
81. C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, M. Mohri, OpenFst: A general and efficient weighted finite-state transducer library. *Implementation and Application of Automata*, 11-23 (2007).
82. G. D. Forney, The viterbi algorithm. *Proceedings of the IEEE* 61, 268-278 (1973).
83. D. Povey *et al.*, Boosted MMI for model and feature-space discriminative training. *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, 4057-4060 (2008).
84. L. Bahl, P. Brown, P. De Souza, R. Mercer, Maximum mutual information estimation of hidden Markov model parameters for speech recognition. *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'86.* 11, 49-52 (1986).