



**FACULTY
OF MATHEMATICS
AND PHYSICS**
Charles University

MASTER THESIS

Vojtěch Hudeček

**Exploiting user's feedback to improve
pronunciation of TTS systems**

Institute of Formal and Applied Linguistics
Faculty of Mathematics and Physics
Charles University in Prague

Supervisor of the master thesis: doc. Ing. Zdeněk Žabokrtský, Ph.D.

Study programme: Informatics

Study branch: Artificial Intelligence

Prague 2017

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In date

signature of the author

Title: Exploiting user's feedback to improve pronunciation of TTS systems

Author: Bc. Vojtěch Hudeček

Department: Name of the department

Supervisor: doc. Ing. Zdeněk Žabokrtský, Ph.D, Institute of Formal and Applied Linguistics

Abstract: Although spoken dialogue systems have greatly improved, they still cannot handle communications involving unknown topics and are very fragile. We will investigate methods that can improve spoken dialogue systems by correcting or even learn the pronunciation of unknown words. Thus we will provide better user experience, since for example mispronounced proper nouns are highly undesirable. Incorrect pronunciation is caused by imperfect phonetic representation, typically phonetic dictionary. We aim to detect incorrectly pronounced words by exploiting user's feedback as well as using prior knowledge of the pronunciation and correct the transcriptions accordingly. Furthermore, the learned phonetic transcriptions can be used to improve speech recognition module by refining its models. Models used in speech recognition cannot handle words that are not in their vocabulary or have phonetic representation. Extracting those words from user's utterances and adding them to the vocabulary should lead to a better overall performance.

Keywords: text-to-speech, automatic speech recognition, user's response, phonetic dictionary, machine learning, mel cepstral distortion

Dedication.

Contents

1	Introduction	3
2	Overview of used techniques and algorithms	6
2.1	Audio signal processing	6
2.2	Mel Cepstral distortion	7
2.3	Finite state transducers	7
2.4	Automatic speech recognition (ASR)	9
2.5	Text-to-speech (TTS)	10
2.5.1	TTS system architecture	10
2.5.2	Grapheme to Phoneme conversion and its drawbacks . . .	11
2.6	Algorithms description	12
2.6.1	Dynamic Time Warping (DTW)	12
2.6.2	Clustering	13
2.6.3	Basic notions in speech communication	15
2.7	Related Work	17
2.7.1	Grapheme-to-phoneme conversion	17
2.7.2	Learning pronunciation from spoken examples	19
2.7.3	Conclusion	21
3	Proposed approaches	22
3.1	Basic overview	22
3.2	Used techniques	22
3.2.1	Phoneme recognition	22
3.3	Text-to-speech systems' evaluation	27
3.4	Evaluating classifiers	28
4	Experiments	30
4.1	Used Datasets	30
4.1.1	Dataset D_1 - artificial	30
4.1.2	Dataset D_2 - artificial	30
4.1.3	Dataset D_3 - Autonomata Spoken Name Corpus	30
4.1.4	Annotation procedure	31
4.2	Identifying difficult words	31
4.2.1	Measuring the difficulty	31
4.2.2	Measure combination	38
4.2.3	Discussion	40
4.3	Improving the pronunciation	42
4.3.1	Overview	42
4.3.2	Exploiting the speech recognizer	42
4.3.3	Combining acoustic and textual information	49
	Conclusion	52
4.4	Summary and discussion	52
4.5	Future Work	52
	Bibliography	54

List of Figures	56
List of Tables	59
List of Abbreviations	60
Attachments	61

1. Introduction

Voice communication is a common interface of many systems nowadays. Voice communication applications range from simple one-word control commands to complex communication in spoken dialogue systems. In this work, we consider mainly such complex systems. We now briefly introduce setting of such system.

Leading a reasonable dialogue with a user is a substantially difficult composed of many subtasks. High level overview of such system is given in Figure 1. First, the input from the user has to be processed. Spoken dialogue systems typically contain Automatic Speech Recognition (ASR) module for this purposes, so the natural speech can be recognized and translated into words. Afterwards, some Spoken Language Understanding (SLU) technique has to be employed in order to extract essential information from the input. In plain words, we have to understand the meaning of the utterance obtained from the user. The dialogue system then derives a relevant response using its policy. The process of this derivation depends on particular implementation. A *rule – based* policy can be used, however it is more common nowadays to employ *data – driven* approaches.

The derived response has to be translated into human readable language, using Natural Language Generation (NLG) techniques. The response can be displayed in textual form, however, it is more common to generate audio recording with human voice reading the response. Although it is possible to use a set of prerecorded utterances, this approach has obvious limitations since it is not able to read an arbitrary phrase. Particularly, it may be difficult to read named entities and numerical values such as time and date. Also, the usage of variable utterances provides better user experience.

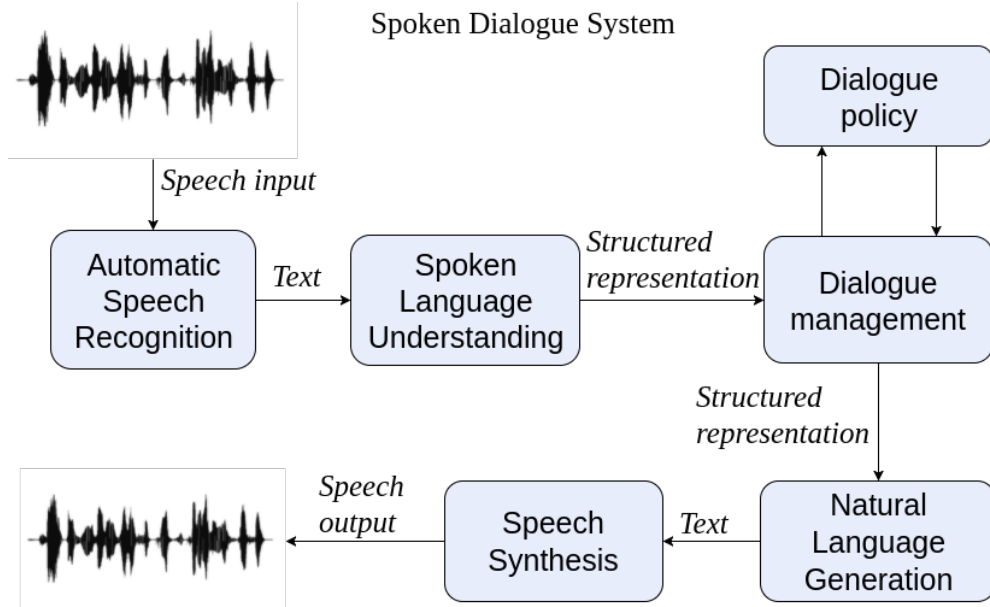


Figure 1.1: A high level architecture of spoken dialogue system.

To introduce a variability and allow to use nearly arbitrary phrase a Text-To-Speech (TTS) module is usually also part of dialogue systems. The purpose of this module is to transform written text utterance to natural speech. Modern TTS

systems produce audio waveforms that sound naturally and the pronunciation is sufficiently good. Nevertheless, it may experience some difficulties, mainly when it comes to unknown words. This may happen, because although some multilingual TTS systems exist ([1], [2]), the system is usually trained using certain set of words, typically from one language. But some applications often require to pronounce named entities or other language- or domain-specific words, that cannot be present during the training phase. Furthermore, the system needs to obtain information, which language it should use. This does not make much sense when considering individual proper names. Such words are called Out-Of-Vocabulary (OOV), because the TTS systems are vocabulary based, i.e. they contain set of known words and their phonetic transcriptions. Occurrences of OOV words cause situations, when the system has to employ some mechanism to derive the pronunciation and the words may be mispronounced. The derivation of the pronunciation is inherently imperfect and may cause errors. Although this does not occur often, the negative effect can be quite strong, since it is inconvenient for the user, especially when known propername, e.g. his or her name is pronounced with mistakes.

In this work, we aim to improve the TTS system pronunciation of OOV words. First, we explore methods that can identify words that are potentially difficult to pronounce. The identification is first step towards the improvement, because we can then ask the user for proper pronunciation. We propose and explore several measures that can reflect badly pronounced words without any prior language knowledge.

Next, we try to improve the TTS system's pronunciation of the desired words. To achieve this, we ask the user and obtain correct pronunciations from him. So we get training examples and we are able to improve the TTS system by processing the obtained recording, deriving a phonetic transcription (i.e. pronunciation) and adding it to the TTS vocabulary. Moreover, the derived pronunciations can be used to improve the recognition ability of the ASR module, since it is also dictionary-based. Thus if we enlarge the vocabulary, the ASR has got access to more possibilities and can recognize the new words correctly.

```
System: Hello, /AANDRZHEZH/.
User: You said it wrong, my name is /ONDRZHEI/.
System: /ANDREY/, correct?
User: No, it is /ONDRZHEI/.
System: Oh, /ONDRZHEI/?
User: That's right.
```

Figure 1.2: Sample dialogue illustrating how the pronunciation correction process may look like. The dialogue system pronounces user's name incorrectly. User then correct it and the system learns. The transcriptions of the user's name are given in ARPABET.

There are several issues, that are related with the OOV words so methods of deriving correct pronunciations has got potentially very useful applications. As it has been said, it can be used to enlarge vocabularies of TTS or ASR systems.

This may potentially lead to better pronunciation in case of TTS. It is because the extended vocabulary implies, that a number of occurrences of the badly pronounced words decreases. In case of ASR systems, bigger vocabulary can cause, that less words are recognized incorrectly. There exist several ways how to obtain such feedback, however, this is not a subject of this work. It is related with respective dialogue policy and concrete applications. Theoretically, the method can work with just one gold example, however, it is better to obtain more recordings in general. In Figure 1.2 we provide basic example of simple dialogue, illustrating how real application could look like. However, in this work we assume the user's recording(s) have been gathered already and we do not discuss the dialogue policy.

2. Overview of used techniques and algorithms

2.1 Audio signal processing

We sketch here the basic principles as described in [4], so we can understand our input data. Speech signal in the real world are mechanical waves, so it is obviously analogous quantity. When we process the speech signal with computers, it is assumed to be digitised, so it is converted into discrete form. When performing digital signal processing, we are usually concerned with three key issues.

1. To remove the influence of phase, because the ear is not sensitive to phase information in speech, and we can use frequency domain representation.
2. Performing source/filter separation, so that we can study the *spectral envelope* of sounds. Spectral envelope characterizes the frequency spectrum of the signal, which is essential for the speech recognition and processing.
3. We often wish to transform these spectral envelopes and source signals into more efficient representations.

Overview of the main standard processing steps follows.

The input signal is divided into *windows*. Windowing considers only some part of the signal. Usually, the signal is transformed at window borders to prevent discontinuities. This is achieved for example by use of *Hamming* window. The complete waveform is therefore a series of windows, that are sometimes called *frames*. The frames overlap, this is influenced by the size of the *frame shift*. Inside one frame, we assume, that the speech signal is stationary and we transform it to a frequency domain by Discrete Fourier Transformation¹.

It is better for observing the natural speech characteristics, to use a logarithm scale instead of linear. In fact, the *mel scale*² is frequently used, which even better corresponds with the human perception. Further, the so called *cepstrum*³ is computed from the log magnitude spectrum, by performing inverse Fourier transformation. Cepstrum can be represented by coefficients, number of which can be chosen. Those are called the *Mel Frequency Cepstral Coefficients(MFCCs)*. To sum up, we have described how to process the digital audio signal and convert it into discrete series of overlapping frames, each of which is represented by a fixed-length vector of MFCCs.

We gave a brief overview of the signal processing procedure that is essential to work with the audio signal. When we mention the input speech signal, we mean series of MFCC vectors, unless explicitly stated otherwise.

¹https://en.wikipedia.org/wiki/Discrete_Fourier_transform

²https://en.wikipedia.org/wiki/Mel_scale

³<https://en.wikipedia.org/wiki/Cepstrum>

2.2 Mel Cepstral distortion

Mel Cepstral Distortion [5] is a well described measure, that should mirror differences between speech samples. There are some caveats worth stating at the outset. First, there are many other factors that contribute to the perception of voice quality. For example it takes no account of speech dynamics, either short-range differentials or long-range prosodic effects. Moreover, distortions in the pitch contour are ignored. However, it should reflect the differences in speech quality and pronunciation.

As we have described in the previous section, the audio sample can be described by its *cepstrum*. This cepstrum may be then represented by Mel Frequency Cepstral Coefficients (MFCCs). Its order can be chosen arbitrarily, we used 35 coefficients length vectors. Thus we can transform each audio record into a sequence of float vectors with length 35. The MCD [5] of two sequences is basically normalized Mean Squared Error between those sequences after aligning. We define it as follows:

$$MCD(v^1, v^2) = \frac{\alpha}{T'} \sum_{ph(t) \neq SIL} \sqrt{\sum_{d=1}^D (v_d^1(t), v_d^2(t))^2} \quad (2.1)$$

The T' stands for the number of frames of the shorter of the recordings. The common use of MCD involves omitting the 0-th coefficient, since it corresponds to the intensity of signal and therefore it is not usually desired. The scaling factor α is present for historical reasons.

The MCD measure has one substantial disadvantage: it does not reflect, that the two compared recordings may not have the same length and, what is more, they can be misaligned. Thus, we use a modified algorithm that first aligns the sequences using Dynamic Time Warping and then computes MCD with the result.

2.3 Finite state transducers

Finite state transducers (FSTs) [6], are basically finite state automata that are augmented by addition of output labels to each transition. We can further modify the FST and add a weight to each edge. We define a semiring $(S, \oplus, \otimes, \bar{0}, \bar{1})$ as a system that fulfills the following conditions:

- $(S, \oplus, \bar{0})$ is a commutative monoid with identity element $\bar{0}$
- $(S, \otimes, \bar{1})$ is a monoid with identity element $\bar{1}$
- \otimes distributes over \oplus
- $\forall a \in S : a \otimes \bar{0} = \bar{0} \otimes a = \bar{0}$

With this definition of the semiring, we can describe a weighted FST [7] T over a semiring S formally as a 8-tuple $(\Sigma, \Delta, Q, I, F, E, \lambda, \rho)$ provided that:

- Σ, Δ are finite input and output alphabets

- Q is a finite set of states
- $I \subset Q$ is a set of initial states
- $F \subset Q$ is a set of final states
- E is a multiset of transitions
- $\lambda : I \rightarrow S$ is an initial weight function
- $\rho : F \rightarrow S$ is a final weight function.

Each transition is element of $Q \times \Sigma \times \Delta \times S \times Q$. Note, that since E is a multiset, it allows two transitions between states p and q with the same input and output label, and even the same weight. However, this is not used in practice. An example of weighted FST is given in Figure 2.3.

Finite state transducers have been used widely in many areas, especially linguistics. They can represent local phenomena encountered in the study of language and they usage often leads to compact representations. Moreover, it is also very good from the computational point of view, since it is advantageous in terms of time and space efficiency. Whole area of algorithms has been described that consider FSTs. One of the most important is *determinization*, which determinizes paths in the FST and thus allows the time complexity to be linear in the length of input. Weighted FSTs have been widely used in the area of automatic speech recognition. This allows to compactly represent the hypothesis state of the acoustic model as well as combining it with the information contained in the language model.

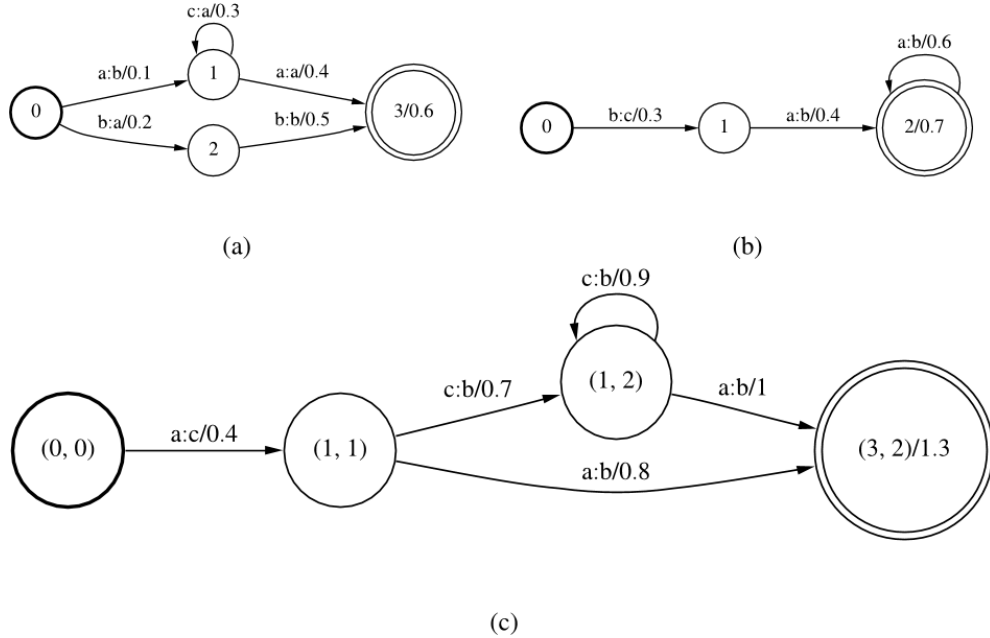


Figure 2.1: Example of Finite State Transducers and their composition

2.4 Automatic speech recognition (ASR)

Overview

The task in ASR is quite clear: An utterance spoken in natural language should be translated into its text representation. Formally, we are given a sequence of acoustic observations $\mathbf{X} = X_1 X_2 \dots X_n$ and we want to find out the corresponding word sequence $\mathbf{W} = W_1 W_2 \dots W_m$ that has a maximum posterior probability $P(W|X)$ i.e., according to the Bayes rule:

$$\hat{\mathbf{W}} = \underset{w}{argmax} \frac{P(\mathbf{W})P(\mathbf{W}|\mathbf{X})}{P(\mathbf{X})} \quad (2.2)$$

The problem's solution consists of two subtasks. First, we have to build accurate acoustic model that describes the conditional distribution $P(W|X)$. The second problem is to create a language model that reflects the spoken language that should be recognized. Usually, a variation of the standard N -gram approach is sufficient.

Individual words are composed of phonetic units, which are in turn modeled using states in probabilistic machinery such as a finite state transducers. Because the speech signal is continuous, it is transformed to discrete sequence of samples. MFCC's are commonly used to represent the signal. The process of deriving this sequence is described in further detail in section 2.1. In general, it is difficult to use whole-word models for the acoustic part, because every new task may contain unseen words. Even if we had sufficient number of examples to cover all the words, the data would be too large. Thus, we have to select basic units to represent salient acoustic and phonetic information. These units should be *accurate*, *trainable* and *generalizable*. It turns out that the most appropriate units are phones. Phones are very good for training and generalization, however, they do not include contextual information. Because of this, so called triphones are commonly used.

To model the acoustics, Hidden Markov Models(HMM) are commonly used, because they can deal with unknown alignments. In recent years, neural networks have experienced big breakthrough and they found application even in the area of speech recognition. Namely, recurrent neural networks are able to work with an abstraction of memory and process sequences of variable length, so it overcomes the HMM's in terms of accuracy. The language and acoustic models are traditionally trained independently and they are joined during the decoding process. The decoding process of finding the best matched word sequence \mathbf{W} to match the input speech signal \mathbf{X} in speech recognition systems is more than a simple pattern recognition problem, since there is an infinite number of word patterns to search in continuous speech recognition. A lattice is built in order to make the decoding. Its nodes represent acoustic units and edges are assigned costs. This assignment corresponds to the likelihood determined by the acoustic model as well as the language model, so the FST is a composition of theses two. Decoding the output is realized as searching the best path through this lattice.

The decoding graph is constructed in such a way that the path can contain only

words present in the vocabulary the language model is built on. Thus, each possible path consists of valid words. Because of that, we can see the word as a basic unsplittable unit, although it is actually composed from phones, or triphones respectively. However, we can build an FST that allows to construct the path from phones and thus recognize the input on the phonetic level. Also the search algorithm preserves alternative paths so in the end it gives us not only the best path but also list of alternative hypotheses. We call it the n -best list. Each hypothesis in the n -best list is associated with its likelihood that expresses information from both the language and acoustic model. In TODO:chapter we explore the n -best list and propose method that exploits it.

2.5 Text-to-speech (TTS)

Overview

[4] The text-to-speech problem can be looked at as a task of transforming an arbitrary utterance in natural language from its written form to the spoken one. In general, these two forms have commonalities in the sense that if we can decode the form from the written signal, then we have virtually all the information we require to generate a spoken signal. Importantly, it is not necessary to (fully) uncover the meaning of the written signal, i.e. employ Spoken Language Understanding (SLU).

On the other hand, we may need to generate prosody which adds some information about emotional state of the speaker or emphasizes certain parts of the sentence and thus changing the meaning slightly. To obtain prosodic information, sophisticated techniques need to be involved, since its not a part of common written text, except some punctuation. Another difficulties stem from the fact, that we often need to read numbers, or characters with special meanings such as dates or mathematical equations. The problem of converting text into speech has been heavily explored in the past and the TTS systems (engines) are very sophisticated nowadays. The subsequent section, describes briefly the typical architecture of TTS systems.

2.5.1 TTS system architecture

Let us now describe a common use of TTS system. The architecture usually consists of several modules, although some end-to-end systems base on neural networks have appeared recently ([8], [9]). We give an overview of the typical TTS architecture in Figure 2.5.1. First, the input text is divided into sentences and each sentence is further tokenized, based on whitespace characters, punctuation etc. Then, the non-natural language tokens are decoded and transformed into text form. We then try to get rid of ambiguity and do prosodic analysis. Although much of the information is missing from the text, we use algorithms to determine the phrasing, prominence patterns and tuning the intonation of the utterance.

Next is the synthesis phase. The first stage in the synthesis phase is to take the words we have just found and encode them as phonemes. There are two main approaches how to deal with the synthesis phase. More traditional approach is so

called **unit concatenation**. This approach uses a database of short prerecorded segments of speech (roughly 3 segments per phoneme). These segments are then concatenated together with some use of signal processing so they fit together well. An alternative is to use machine learning techniques to infer the specification-to-parameter mapping from data. While this and the concatenative approach can both be described as data-driven, in the concatenative approach we are effectively memorizing the data, whereas in the statistical approach we are attempting to learn general properties of the data.

Trained models are able to transform the phonemes into audio signal representation. This representation is then synthesized into waveforms using a vocoder module. Although unit concatenation approaches generally achieve better results, the statistical ones are more flexible and have good possibilities to fine tuning or postprocessing.

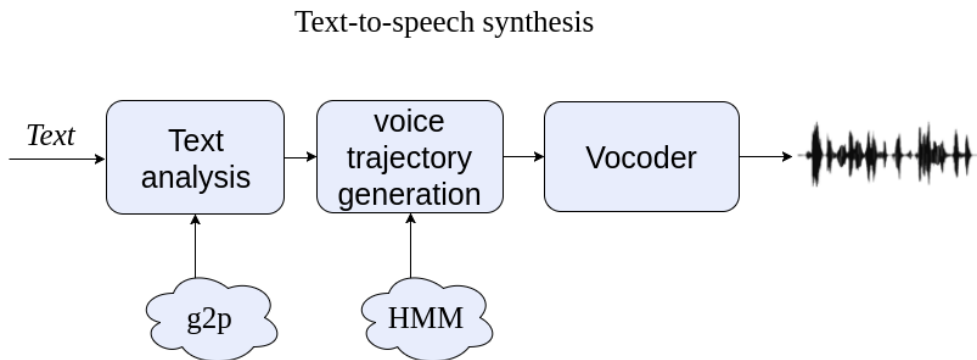


Figure 2.2: Overview of a typical statistical Text-to-speech system architecture. The text is first analysed and translated to phonetic transcription. Then, the acoustic parameters are estimated. Finally, the speech is synthesized.

2.5.2 Grapheme to Phoneme conversion and its drawbacks

In the stage of converting graphemes (i.e. text) to phonemes, a *g2p* module is commonly used. The *g2p* task is to derive pronunciations from orthographic transcription. Traditionally, it was solved using decision trees models. It can also be formulated as a problem of translating one sequence to another, so neural networks can be used to solve this task ([10]). However, in our work we use joint-sequence model ([11]), which estimates joint distribution of phonemes based on probabilities derived from *n*-grams. The *g2p* module is a crucial component of the system and it has great impact on the final pronunciation. Since we use machine learning techniques and train on the dictionary data, the model inherently adopts pronunciation rules from the respective language. Although it is in principle possible to train multilingual *g2p* ([12]), it is usually not the case in common TTS systems. The problem arises when words that originate in some foreign language should be pronounced. In this work we address this issue by deriving pronunciations from the ASR output.

```

<?xml version="1.0"?> <spek version="1.0" xmlns="..."
xmlns:xsi="..." xsi:schemaLocation="..." xml:lang="en-US">
    <voice gender="female">Mary had a little lamb,</voice>
    <voice gender="female" variant="2">
        Its fleece was white as snow.
    </voice>
    <voice name="Mike">I want to be like Mike.</voice>
</spek>

```

Figure 2.3: Example of SSML code

Speech Synthesis Markup Language (SSML)

[13] SSML is an XML standard that allows to specify aspects of the speech. It is an input to the synthesis module and many TTS engines support its use. We can specify emotions, breaks etc. with the help of SSML. More importantly, it can be used to input particular phonemes and thus circumvent the *g2p* module. In our work we use this method to feed our phonetic transcriptions into the engine. The disadvantage of this method is, that many TTS engines process the SSML input imperfectly or not at all. However, in principle it is possible to add the transcriptions directly into the system's vocabulary.

Overview of the used TTS systems

1. Cereproc⁴ This engine is a commercial software that is free for educational purposes. Although it is not open-sourced, we use it for its good quality and reliable outputs.
2. MaryTTS⁵ MaryTTS is a German open-source project written in Java. It is highly customizable and modular. Thus we can explore output of an arbitrary module or replace it in the processing pipeline. MaryTTS is based on client-server architecture.
3. gTTS⁶ is a TTS service provided online by Google. It achieves very good quality, however it allows only use of one voice in the free version.
4. Pico ⁷ SVOX Pico TTS is a lightweight engine that lacks a good quality of the gTTS, however it offers a large selection of voices and it is commonly used on the phones with Google's Android operating system.

2.6 Algorithms description

2.6.1 Dynamic Time Warping (DTW)

[14] The measurement of similarity between two time series is an important com-

⁴<https://www.cereproc.com/>

⁵<http://mary.dfki.de/>

⁶<https://pypi.python.org/pypi/gTTS>

⁷<https://github.com/stevenmirabito/asterisk-picotts>

ponent of many applications. Moreover, sometimes we need to align two sequences that describe same data but are of different lengths. Both this tasks can be solved with use of DTW. Suppose we have two time series, a sequence Q of length n , and a sequence C of length m , where

$$\begin{aligned} Q &= q_1, q_2, \dots, q_i, \dots, q_n \\ C &= c_1, c_2, \dots, c_j, \dots, c_m \end{aligned}$$

To align these two sequences using DTW, we first construct an n -by- m matrix where the (i^{th}, j^{th}) element of the matrix corresponds to the squared distance, $d(q_i, c_j) = (q_i - c_j)^2$, which is the alignment between points q_i and c_j . To find the best match between these two sequences, we retrieve a path through the matrix that minimizes the total cumulative distance between them. In particular, the optimal path is the path that minimizes the warping cost:

$$DTW(Q, C) = \sqrt{\sum_{k=1}^K w_k} \quad (2.3)$$

where w_k is the matrix element $(i, j)_k$ that also belongs to k^{th} element of a warping path W , a contiguous set of matrix elements that represent a mapping between Q and C . Methods of dynamic programming are used to fill in the values and find alignment of sequences. Thus a result of this algorithm is a mapping σ that can be used to construct sequence of pairs A containing members of both sequences C, Q in each time step. σ is constructed in such a way, that it holds:

$$A_i = (C_{\sigma(C,i)}, Q_{\sigma(Q,i)}); i = 1 \dots K \quad (2.4)$$

$$\max(|C|, |Q|) \leq K \quad (2.5)$$

$$\sum_{i=0}^K d(\sigma(C, i), \sigma(Q, i)) \text{ is minimal} \quad (2.6)$$

Where $d(x, y)$ is an arbitrary distance metric. In our application, we want to align two sequences of phonemes, hence we can treat it as strings and work with repsective distance. We choose Levensthein distance to be our metric.

2.6.2 Clustering

Cluster analysis or clustering ⁸ is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar to each other than to those in other clusters. The measure of similarity can be arbitrary distance measure. Usually, euclidean distance is used when considering points in space, however it is not a requirement. Clustering is inherently an unsupervised method, we often want to describe unknown data, that is find some structure or local similarities.

There exist different methods of clustering. *Hierarchical clustering* is based on the core idea of objects being more related to nearby objects than to objects

⁸https://en.wikipedia.org/wiki/Cluster_analysis

farther away. These algorithms connect object” to form clusters based on their distance. Hierarchical clustering is a bottom-up method, meaning that it forms clusters iteratively. The hierarchical clustering algorithms’ result can be represented as a dendrogram, which is a tree diagram representing relations among the explored data. An example of dendrogram is given in Figure 2.6.2. We can make a horizontal cut to derive respective clusters.

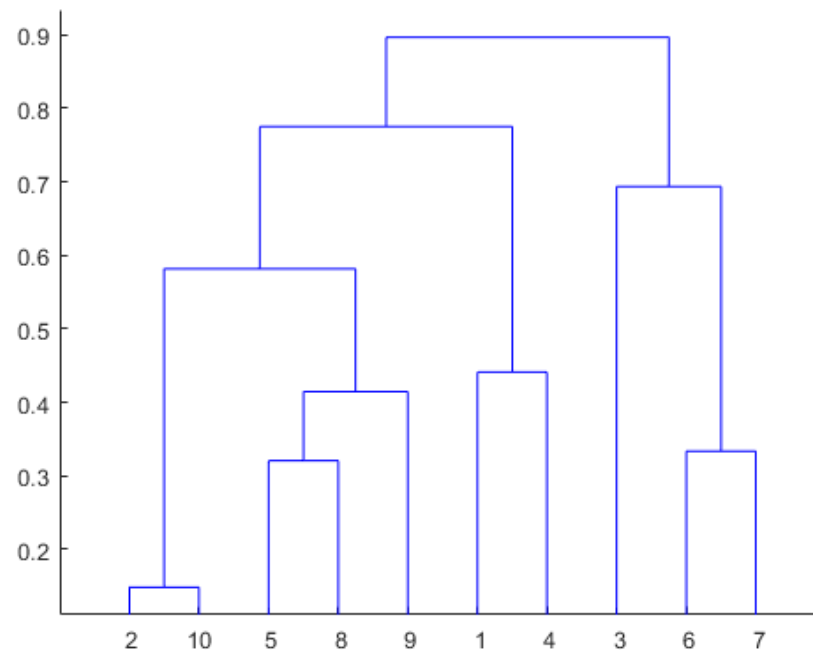


Figure 2.4: An example of dendrogram outputted by hierarchical clustering methods.

On the other hand in *Centroid-based clustering*, clusters are represented by a central vector, which may not necessarily be a member of the data set. If we fix the number of clusters to k , we can see the task as an optimization problem: find the k cluster centers and assign the objects to the nearest cluster center, such that the squared distances from the cluster are minimized. Typical algorithm from this group is k -means clustering⁹. which is an Expectation-Maximization (EM) algorithm.

Spectral clustering [15] The task of finding good clusters has been the focus of considerable research in machine learning and pattern recognition. Often, EM algorithms are used to learn a mixture density. Unfortunately, these approaches have several drawbacks. First, to use parametric density estimators, simplifying assumptions usually need to be made (e.g., that the density of each cluster is Gaussian). Second, the log likelihood can have many local minima and therefore multiple restarts are required to find a good solution using iterative algorithms. An alternative is to use spectral methods for clustering. Here, one uses the top eigenvectors of a matrix derived from the distance between points. Therefore, the mentioned problems are overcome.

⁹https://en.wikipedia.org/wiki/K-means_clustering

2.6.3 Basic notions in speech communication

Communication is a process during which some entities exchange information. There are different types of communication, we focus here on the communication in natural language. When we consider communication, we should define what we understand by a term form. This concept deserves some elaboration. A proper understanding of this part of communication is essential for our purposes, but thankfully we can rely on established frameworks.

The main issue when discussing form, or a message, is to distinguish the general nature of the form from any specific instantiation it may have. Imagine, someone wants to deliver an information by showing some color. If a red card is used, it isn't the card itself that is the important point, rather than the red color was used instead of different color. So for example, a different red card could be used successfully as it is not the particular red card itself which is the form, but rather that the card has the property "red" and it is this property and its contrasting values that is the basis of the communication system. We use the term signal to refer to the physical manifestation of the form. Here the term signal is again used in a specific technical sense: it can be thought of as the thing which is stored or the thing which is transmitted during communication. The idea of "red" and the fact that it contrasts with other colors is the form; the physical light waves that travel are the signal.

It is important to see that the relationship between meaning and form, and form and signal are quite separate. If we assigned meaning only to colors red, blue and green and an orange card would have been showed to us, we would not decode the meaning, although the signal-form translation worked well. This situation is similar to that when we read a new word - while are quite capable of using our eyes to read the physical pattern from the page and create an idea of the letters in our head, if we don't know the form-meaning mapping of the word, we can not understand it. Finally, we use the term channel or medium to refer to the means by which the message is converted and transmitted as a signal.

In this work, we deal mainly with the spoken channel and the written channel. Encoding is the process of creating a signal from a message. When dealing with speech, we talk of speech encoding and when dealing with writing we talk of writing encoding. Speech encoding by computer is more commonly known as speech synthesis, which is of course the topic of this book. The most significant aspect of speech encoding is that the nature of the two representations, the message and the speech signal, are dramatically different. A word (e.g. *HELLO*) may be composed of four phonemes (/H EH LOW/) but the speech signal is a continuously varying acoustic waveform, with no discrete components. In this work, we focus on derivation of the first representation and we let the used synthesis frameworks do the speech signal generation. So we simplify the speech representation to its phonetic transcription.

However, it is important to understand, that we do not change the meaning or even form of the communication, we just want to use different channel for the transmission. When we consider the verbal language, we may talk about the duality of form such that we have phonemes which combine to words which combine to form sentences. We define three terms that help us to describe the problematics:

1. **phonemes** are members of the relatively small set of units which can be combined to produce distinct word forms.
2. Term **phone** is used to describe a single speech sound, spoken with a single articulatory configuration.
3. **allophones** link the phonemes and phones: different ways of realising a single phoneme are called allophones.

The duality principle states that verbal language is not one system but two; the first system uses a small inventory of forms, phonemes, which sequentially combine to form a much larger inventory of words.

We have a similar dual structure in written communication, where words are made of **graphemes**. Graphemes are in many ways analogous to phonemes, but differ in that they vary much more from language to language. In alphabetic languages like English, graphemes can be thought of as letters or characters; in syllabic writing like Japanese hiragana, each grapheme represents a syllable or mora (unit that determines syllable weight, i.e. stress or accent) and in languages like Chinese each grapheme represents a full word or part of a word. We consider alphabetic languages in this work, encoded with the Latin alphabet¹⁰. However, it is possible to extend the used grapheme-to-phoneme conversion technique [11] to work with other types of languages.

To encode the phonemes, phonetic alphabets are used. The standard is an International Phonetic Alphabet (IPA)¹¹. IPA symbols are composed of one or more elements of two basic types, letters and diacritics. The general principle of the IPA is to provide one letter for each distinctive sound (speech segment), although this practice is not followed if the sound itself is complex. An example of IPA characters is given in Figure 2.6.3.

The problem is that some of the tools we work with use different phonetic alphabets with possibly different rules. However, the TTS engines work with IPA, so it is necessary to convert the transcriptions. To overcome this problem, we have derived one-to-one mappings between these sets and IPA. This process may introduce some errors, however the alphabets are usually based on similar principles, so it is not so hard to derive these mappings. Namely, we used the Arpabet¹², Timit¹³ and CGN¹⁴ phoneme sets.

As we have suggested, some speech synthesizers are able to accept input in the SSML format which we introduce in 2.5.2. However, not all the synthesizers have this ability. We use Cereproc to work with utterances written in SSML. An example of such utterance is given in Figure 2.6.3. The problem with providing transcriptions in this way is, that the Cereproc synthesizer ignores the characters defining stretch of the phoneme. Therefore, we omit these characters from the transcriptions by mapping the stretched phonemes to their base variants. This leads to worse quality of the pronunciation, however is sufficient to explore if the transcription contains sensible phonemes.

¹⁰https://en.wikipedia.org/wiki/Latin_alphabet

¹¹https://en.wikipedia.org/wiki/International_Phonetic_Alphabet

¹²<https://en.wikipedia.org/wiki/Arpabet>

¹³<https://catalog.ldc.upenn.edu/docs/LDC93S1/PHONCODE.TXT>

¹⁴<https://pdfs.semanticscholar.org/64d5/d612f9c639271a1340820a418fa5baf02770.pdf>

ɪ READ	ɪ SIT	ʊ BOOK	u: TOO	ɪə HERE	eɪ DAY	John & Sarah Free Materials 1996	
e MEN	ə AMERICA	ɜ: WORD	ɔ: SORT	ʊə TOUR	ɔɪ BOY	əʊ GO	
æ CAT	ʌ BUT	ɑ: PART	ɒ NOT	eə WEAR	aɪ MY	aʊ HOW	
p PIG	b BED	t TIME	d DO	tʃ CHURCH	dʒ JUDGE	k KILO	g GO
f FIVE	v VERY	θ THINK	ð THE	s SIX	z ZOO	ʃ SHORT	ʒ CASUAL
m MILK	n NO	ŋ SING	h HELLO	l LIVE	r READ	w WINDOW	j YES

Figure 2.5: Some letter of the IPA alphabet together with their occurrences in English words.

...
`<phoneme alphabet="ipa" ph="təmeɪrou"> tomato </phoneme>`
 ...

Figure 2.6: An example of the synthesizer's input in SSML, forcing it to use our phonetic transcription.

2.7 Related Work

2.7.1 Grapheme-to-phoneme conversion

An automatic grapheme-to-phoneme conversion was first considered in the context of TTS applications. The input text needs to be converted to a sequence of phonemes which is then fed into a speech synthesizer. It is common in TTS systems that they first try to find the desired word in the dictionary and if it doesn't find it, it employs the grapheme-to-phoneme (*g2p*) module. A trivial approach is to employ a *dictionary look-up*. However, it cannot handle context and inherently covers only finite set of combinations. To overcome this limitations, the rule-based conversion was developed. Kaplan and Kay [16] formulate these rules in terms of finite-state automata. This system allows to greatly improve coverage. However the process of designing sufficient set of rules is difficult, mainly since it must capture irregularities. Because of this, a *data-driven* approach based on machine learning has to be employed. Many such techniques were explored, starting with Sejnowski and Rosenberg [17]. The approaches can be divided into three groups.

Techniques based on local similarities

The techniques presuppose an alignment in the training data between letters and phonemes or create such an alignment in a separate preprocessing step. The alignment is typically construed so that each alignment item comprises exactly one letter. Each slot is then classified (using its context) and a correct phoneme is predicted. Neural networks and decision tree classifiers are commonly used for this task.

Pronunciation by analogy

This term is typically used for methods that could be described as nearest-neighbor-like. They search for local similarities in the training lexicon and the output pronunciation is chosen to be analogous to retrieved examples. In the work of Dedina and Nusbaum [18] the authors first identify words from the database that match the input string and then build a pronunciation lattice from them. Paths through the lattice then represent the derived pronunciations.

Probabilistic approaches

The problem can also be viewed from a probabilistic perspective. Pioneers in this area were Lucassen and Mercer [19]. They create 1 – to – n alignments of the training data using a context independent channel model. The prediction of the next phoneme is based on a symmetric window of letters and left-sided window of phonemes. Authors then construct regression tree, the leafs of which carry probability distribution over phonemes.

A popular approach based on probability modelling is to employ so called *joint sequence models* [11]. We use an open-source implementation of such a model in our work. This approach formalizes the task as follows:

$$\varphi(\mathbf{g}) = \operatorname{argmax}_{\varphi' \in \Phi^*} p(\mathbf{g}, \varphi') \quad (2.7)$$

where $*$ denotes a Kleene star. In other words, for a given ortographic form $\mathbf{g} \in G^*$ we want to find the most likely pronunciation $\varphi \in \Phi^*$, where G, Φ are ortographic and phonetic alphabets. The fundamental idea of joint-sequence models is that the relation of input and output sequences can be generated from a common sequence of joint units. These units carry both input and output symbols. Formally, the unit called *grapheme* is a pair $q = (\mathbf{g}, \varphi) \in Q \subset G * \times \Phi$ where \mathbf{g} and φ are letter and phoneme sequences which can be of different lengths. In the simplest case, each unit carries zero or one input and zero or one output symbol. This corresponds to the conventional definition of finite state transducers (FST) that are described in 2.3. The letter and the phoneme sequences are grouped into an equal number of segments. Such a grouping is called a joint segmentation. The joint probability is defined as:

$$p(\mathbf{g}, \varphi) = \sum_{\mathbf{q} \in S(\mathbf{g}, \varphi)} p(\mathbf{q}) \quad (2.8)$$

since there are many possible groupings of input sequences in general. The joint probability distribution $p(\mathbf{g}, \varphi)$ has thus been reduced to a probability distribution

$p(\mathbf{q})$ over grapheme sequences $\mathbf{q} = q_1, \dots, q_K$, which can be modeled using N -gram approximation:

$$p(q_1^K) \cong \prod_{j=1}^{K+1} p(q_j | q_{j-1}, \dots, q_{j-M+1}) \quad (2.9)$$

The probability distribution is then typically estimated by an Expectation-Maximization algorithm.

Generally, the problem of the wrong pronunciation in TTS is caused by a bad phonetic transcription. Traditional TTS systems are modular, one module's output is inputted into the next one. Because of this fact, the errors cumulate and thus the mistakes made by $g2p$ cannot be repaired. So if we want to improve the pronunciation, we can try to improve the $g2p$ as it is done in [20]. Authors in this work propose a method of exploiting a $g2p$ trained on a language with a high number of available resources to create a $g2p$ for language for which we do not have sufficient number of examples. This method relies on the existence of a conversion mapping between these two languages. Also it requires to do the conversion for every new language. In theory, a model can be created, that is able to transcribe grapheme sequences into an appropriate phonetic representation and can handle multiple languages. However, it needs to somehow obtain information, which language the input sequence comes from, which is not straightforwardly doable. This method has several drawbacks, because the language is not always known and the set of known languages is limited, so it does not really solve the OOV problem. Also, it potentially requires a lot of training data. Moreover, if we want to learn a new pronunciation of just one word, it's more convenient to do it in a different way.

2.7.2 Learning pronunciation from spoken examples

. This group of methodologies aim to derive phonetic transcriptions directly from audio input. They are built on the theory of Automatic Speech Recognition which we discuss in 2.4. Authors of [21] introduce method of deriving correct pronunciation for a word in order to enlarge the recognizer's dictionary. They propose a data-driven approach to automatically add new words and their variants, respectively. The authors argue that in the spontaneous speech, the most frequent pronunciation does not need to be the one that is marked as correct and is used in the training phase. Thus the overall performance of the recognizer may be degraded since the phonetic units are bound with inadequate acoustics. The method is proposed, that relies on the use of both phoneme and word-level speech recognizer. The phoneme recognizer is constructed using smoothed bigram Language model. We discuss the phoneme recognizers in more detail in 2.4. The algorithm collects all occurrences of words in the database and creates phonetic transcriptions using the recognizer. It then sorts the variants, rejects some of them and creates an $n - best$ list which is added to the dictionary. The recognizer can then be retrained, allowing multiple pronunciations for each word. Thus the recognition performance can be improved by an automatic procedure without the need for using the phonological rules.

In the work of McGraw et al. [22], the concept of Pronunciation Mixture Models

is introduced. The authors use a special kind of speech recognizer, the search space of which has four primary hierarchical components: a language model G , a phoneme lexicon L , phonological rules P that expand the phoneme pronunciations to their phone variations, and a mapping from phone sequences to context-dependent model labels C . All of the components mentioned can be with advantage represented as FSTs and thus the full decoder network can be represented as a composition of these components: $R = C \circ P \circ L \circ G$. A probabilistic lexicon is considered, meaning, that several pronunciations are allowed for each word and there is no hard limitation that would force the recognizer to choose one. Instead, a kind of soft voting is considered, meaning, that each transcription is used with certain probability. Also, joint-sequence modelling is considered, as we introduced in 2.7.1, so each transcription is considered together with its orthographic form. That means, that we can describe the log-likelihood of M utterances $D = \{\mathbf{u}_i, \mathbf{W}_i\}$ where \mathbf{u}_i are speech data and \mathbf{W}_i their transcriptions as follows:

$$\mathcal{L}(\Theta|D) = \sum_{i=1}^M \log \sum_{\mathbf{B} \in \mathcal{B}} P(\mathbf{u}_i, \mathbf{B}, \mathbf{W}_i; \Theta) \quad (2.10)$$

where \mathbf{B} are respective phone sequences (i.e. pronunciations) and Θ represents the model parameters. Then, we derive using a chain rule:

$$P(\mathbf{u}_i, \mathbf{B}, \mathbf{W}_i; \Theta) = P(\mathbf{u}_i|\mathbf{B})P(\mathbf{B}|\mathbf{W}_i; \Theta)P(\mathbf{W}_i) \quad (2.11)$$

If we further assume, that pronunciation sub-units $\mathbf{b}_i \in \mathbf{B}$ are context independent, we can transcribe the above expression:

$$P(\mathbf{u}_i|\mathbf{B})P(\mathbf{B}|\mathbf{W}_i; \Theta)P(\mathbf{W}_i) = P(\mathbf{u}_i|\mathbf{B})\left(\prod_{j=1}^{k_i} P(\mathbf{b}_j|\mathbf{w}_j^i; \Theta)\right)P(\mathbf{W}_i) \quad (2.12)$$

The model parameters are then estimated using the EM-algorithm. Parameters related to the language model, can be initialized with use of grapheme language model. Several technical issues has to be dealt with, however the Pronunciation Mixture Models can be trained on the same data as traditional ASR engines and can be used to obtain phonetic transcriptions from audio data.

Similar approach is considered in the work of Reddy and Gouvea [23], except they do not have access to acoustic models or phone lattices. They use only the mistakes done by the recognizer as their source of information. An OOV word is passed through an ASR decoder giving an n -best word recognition output. Since the words are OOVs, every hypothesis will be a recognition mistake. These mistakes are then exploited, assuming that the following generative story of the recognition output for a word \mathbf{w} holds:

1. A pronunciation baseform \mathbf{b} is drawn from the distribution Θ .
2. A phonetic confusion function from the word \mathbf{w} and the selected baseform \mathbf{b} is applied in order to generate a phoneme sequence \mathbf{p} with probability $P(\mathbf{p}|\mathbf{b}, \mathbf{w})$

3. A word sequence \mathbf{e} with probability $P(\mathbf{e}|\mathbf{p}, \mathbf{b}, \mathbf{w}) = 1$ is generated using the pronunciation lexicon.

The authors model the joint probability of hypothesis and reference word $P(\mathbf{e}, \mathbf{w}) = P(\mathbf{w} \sum_b f_{e,b,w})$, where $f_{e,b,w} = P(\mathbf{e}|\mathbf{b}, \mathbf{w})$ is the phonetic confusion function and it is used to estimate the distribution $P(\mathbf{b}|\mathbf{w}, \mathbf{e})$. Thus they are able to derive pronunciations without access to ASR lattices, i.e. it only considers the recognizer as a black-box.

2.7.3 Conclusion

Many approaches were introduced that are able to convert an utterance in orthographic or audio form to its phonetic representation. *G2P* converts the grapheme transcriptions, using only the text input. It is a well explored field of study with many different variants of realization. Although it achieves very good results nowadays, it suffers from the fact that same groups of letters may have different pronunciations in different languages. We typically don't have access to the information which language is considered and it may be difficult to get access to sufficient number of datasets. The latter problem can be partially solved by transfer learning as proposed in [20].

Alternatively, one can derive pronunciations directly from an audio signal. This approach has been also explored by some authors, however it usually requires quite low-level modifications of the speech recognizer. Also, the authors used the derived pronunciations to enlarge the phonetic dictionary of the recognizer, not to improve the Text-To-Speech systems. We explore methods of merging the mentioned approaches to combine both textual and acoustic information and its usability when confronted with human judgments.

3. Proposed approaches

3.1 Basic overview

We aim to improve the pronunciation, that means, we need to derive phonetic transcriptions of a good quality. The straightforward way of achieving this is to use a grapheme-to-phoneme (g2p) module. As we discuss in 2.5.2 and 2.7, this approach has some limitations, mainly because the pronunciation of some tokens in different languages may vary. However, grapheme-to-phoneme conversion is a well explored area and its output may serve as a sufficiently good baseline. In section 2.7 we also discuss methods that are able to derive phonetic transcriptions directly from the speech signal, using a speech recognition framework. These methods are somewhat complex and they are typically specific for a particular recognizer. Rather than improving any of these methods, we aim to explore a possibility to combine them. That is, we want to exploit both textual and acoustic information to derive phonetic transcriptions of the desired words.

In the desired settings, the acoustic data with user’s recordings are obtained online. We mean by this a situation, when the system asks the user for a correct pronunciation directly. It can be difficult to make user say words we are interested in, so the ultimate goal is to develop an algorithm able to extract those words from a dialogue history or ask user to say them in a not irritating way. The Crucial point of this approach is, that the user should not feel bored by the process. For the purpose of this work, we assume that we have already obtained recordings with correct pronunciations and work with it. In the real setting we would have to employ a dialogue policy to be able to gather our own recordings or process the history and try to identify respective words.

3.2 Used techniques

3.2.1 Phoneme recognition

To improve the pronunciation, it is essential to obtain information from the user’s recording. We need to work on the phonetic level, so we have to gather phonetic transcription of the recording. In theory, we could use the standard output of the speech recognizer, i.e. a sequence of words and transcribe it phonetically. However, this is not desirable, because such procedure adds new source of potential errors, because we would have to use some mechanism, *g2p* module for instance, that would create the transcription. Furthermore, since we aim to add new words to the TTS vocabulary, it is likely, that the word is missing also in the vocabulary of the speech recognition system. This fact means that the desired word will not appear in the recognizer’s output and thus we cannot obtain any relevant transcription. Instead, we can modify the decoder used in the speech recognizer in such way, that it outputs sequences of phonemes rather than words. This can be done in several ways, some of which we introduce in the following text.

On the Automatic Speech Recognition (ASR) decoding process

We use the Kaldi framework¹ for the ASR task and follow the procedure suggested in the official documentation. Kaldi constructs a Finite State Transducer for purposes of the decoding process². This transducer is created as a composition of four other FSTs, $HCLG = H \circ C \circ L \circ G$. The meanings of the parts are as follows:

- G is an acceptor (i.e. its input and output symbols are the same) that encodes the grammar or language model.
- L is the lexicon; its output symbols are words and its input symbols are phones.
- C represents the context-dependency: its output symbols are phones and its input symbols represent context-dependent phones, i.e. windows of N phones
- H contains the acoustic model's definitions; its output symbols represent context-dependent phones and its input symbols reference the model.

As we can see, the G FST brings information contained in the language model to the decoding graph. We provide an example in Figure 3.2.1 which shows, that the L transducer determines the output elements. Hence, by modification of G and L we can change the output set of the decoded tokens.

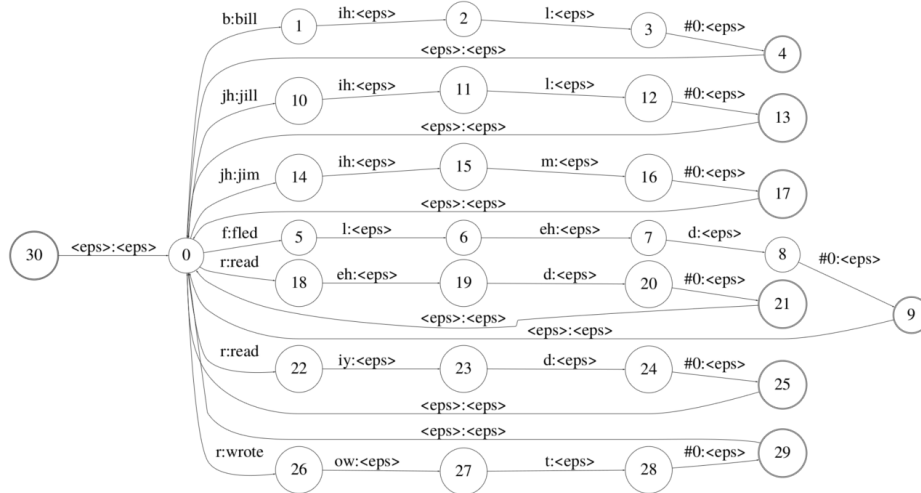


Figure 3.1: An example of the finite state transducer representing the lexicon[24]

Evaluation

In ASR systems, the most common phone recognition evaluation measures are Phone Error Rate (PER), or the related performance metric, phone accuracy rate. The latter is defined by the following expression:

$$Accuracy = \frac{N_T - S - D - I}{N_T} \times 100\% \quad (3.1)$$

¹<http://kaldi-asr.org/>

²<http://kaldi-asr.org/doc/graph.html>

where N_T is the total number of phonemes in the gold utterance and S , D and I are the substitution, deletion and insertion errors, respectively. From this perspective, we can see it as related to Levensthein distance of the gold and test sequences. Then, we can define $PER = 100\% - Accuracy$.

Another measure is correctness, which is similar to accuracy, but does not consider insertion errors. The number of insertion, deletion and substitution errors is computed using the best alignment between two token sequences: the manually aligned (gold) and the recognized (test). An alignment resulting from search strategies based on dynamic programming is normally used successfully for a large number of speech recognition tasks [25]. We use a *sclite* utility³ to measure PER.

Modifying the Language Model

Language Model (LM) is an essential part of every ASR engine. It provides probabilities that a particular word occur in the considered context. For example we can imagine a model that assigns probability to every possible n-gram. The easiest approach is to use a so called 0-gram LM. It simply distributes the probability mass uniformly among the words.

Obviously, this model is not very good. If we want to employ prior knowledge about domain, we can use collected textual data and estimate probabilities of n -grams. If we want to decode phonemes, we can construct a language model that describes probabilities of phoneme occurrences instead of words.

We can then modify L and G transducers to change the output of the recognizer to phonemes. Ideally, we would use a set of name transcriptions to train the language model. However, it is difficult to get such corpus that would be also large enough to robustly estimate the LM parameters. So we create an artificial training dataset by transcribing written text with a *g2p* module. Alternatively, we could use phonetic alignments that are created during the decoding process and can be extracted from the recognizer. However, we did not choose this option, since the transcriptions that are obtained this way are not accurate enough, mainly because of the word recognition errors.

Creation of the decoding graph works with vocabulary and phonetic dictionary. The vocabulary contains a list of known words and the dictionary respective phonetic transcriptions. To change the transducers in the above mentioned way, we reduce the vocabulary to list of phonemes and setting the dictionary mapping to be identity. This way we force the decoder to output phonemes.

Timit dataset

The above approach has a disadvantage lying in the fact that it is difficult to obtain a language model of good quality without a proper corpus of transcriptions. This corpus may be found as a part of the timit dataset [26]. In fact, the Kaldi framework contains scripts, that train a model using this dataset. The timit dataset has got transcriptions labeled with granularity of individual phonemes. This means we can train the phonetic recognizer directly. Problem is that different set of phonemes is used but we can solve it by mapping the phonemes to IPA.

³<http://www1.icsi.berkeley.edu/Speech/docs/sctk-1.2/sclite.htm>

Worse is the fact, that the phone error rate measured on different data gives poor results. This can be caused by the fact that the dataset is not very big and thus the model can have problems accomodating to different acoustic conditions.

Modeling phoneme bigrams

The decoding graph is context dependent. It means that the decoder takes into account both the right and the left context of the phonemes. In order to model the pronunciation on various positions in the word, each phoneme is represented in several variants, depending in which part of the word it is located (*beginning, end, inside*). One option is to model each phoneme as a singleton as we have done in previous. Alternatively, we can create a decoding graph, that models bigrams of phonemes. This way the information about the context is preserved. Moreover, we can model the input more accurately and distinguish pauses between words from space between phonemes. This approach proved to yield the best results.

Exploring the speech recognizer

The decoding process in Automatic Speech Recognition yields a list of hypotheses, ordered by their confidence scores, so called n -best list. Usually, the best hypothesis is chosen and used since it is the most relevant result. However, interesting information may be found deeper of the list. We explore this theory in the following paragraphs.

In section 3.2.1 we have introduced Phone Error Rate (PER). We extend the term here and define **Oracle Phone Error Rate**:

$$OPER = \min_{h \in hypotheses} PER(h, gold) \quad (3.2)$$

We can see that Oracle Phone Error Rate is PER of the best hypothesis (if we consider the hypothesis with the lowest PER as the best one.) In real setup we obviously does not have such oracle that would tell us the best hypothesis, so we have no other option but looking at the top of the n -best list, computing PER per each hypothesis and choosing the closest one. Hence we approximate the true Oracle PER, since if we explore "sufficiently many" best alternatives, the probability that we did not see the best one is very low. We give an overview of our results in Figure 3.2.1.

Using Oracle Phone Error Rate to evaluate the recognizer is basically cheating, since in real decoding process, we are not given the gold transcription which we could use to pick the best hypothesis. Nevertheless, it indicates some properties of the decoder, namely that the hypothesis located at the top is not always the best one. We explore this phenomenon further by looking at the particular position of the closest (best) hypothesis. In Figure 3.2.1 we plot the histogram showing, number of times the closest hypothesis appeared in particular depth. We can see, that relevant hypotheses can be found in bigger depth.

The observations we have made in the previous text indicates that there is a substantial piece of information contained in the phoneme recognizer's n -best list. The problem of the n -best list is, that many pairs of hypothesis differs only in a few positions. That is because of the nature of the procedures used

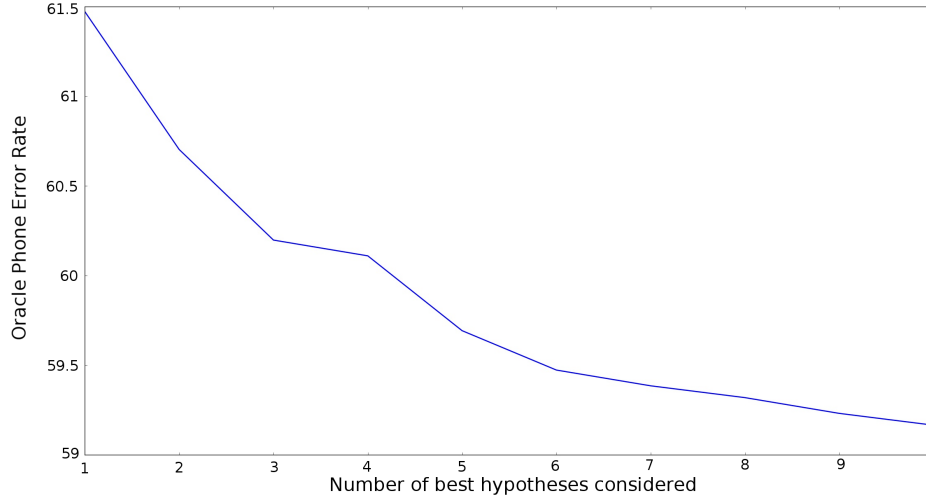


Figure 3.2: A plot of Oracle Phone Error Rate dependency on the depth of the n -best list we explore. We can see, that the Oracle PER decreases with the depth, so it makes sense to try to exploit the hypotheses located deeper.

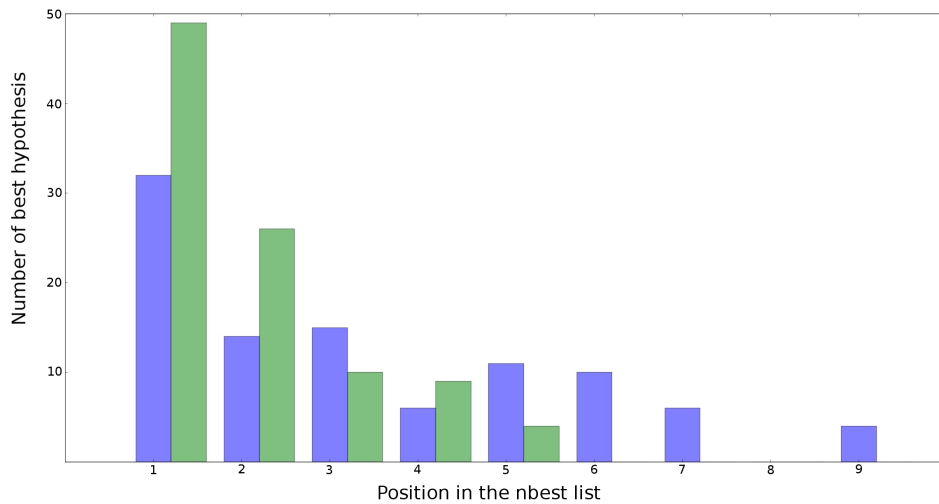


Figure 3.3: Histogram that visualizes the distribution of the hypothesis, that is the closest to the gold reference utterance in terms of Phone Error Rate. The bars show the number of times the best hypothesis was found on the particular position. Two settings are included in this figure: the green bars describe situation, when first five positions were used, the blue ones were measured using nine positions

for the decoding. It is quite intuitive, that hypothesis that differ only in one position have similar probabilities. We would like to use the information but we are not interested in many hypotheses that are nearly the same. So we propose to clusterize the hypotheses contained in the n -best list. The goal is to obtain a more compact version of the list with high variability among items. We adress this issue again in section TODO: ref, giving results and examples.

3.3 Text-to-speech systems' evaluation

The TTS evaluation[27] is a desired task since we often want to choose between two or more engines. However the task introduces several challenges some of which we mention in this section. We need to define a metric in order to evaluate the TTS. Such a metric will be dependend on particular task's setting and may consist of several variables. We can describe a taxonomy of different evaluation methods, depending from which point of view we look at it we distinguish between:

- *Black box vs. Glass box evaluation* - whether we evaluate the TTS system as a whole or look at particular modules of it.
- *Human vs. automated* There are two fundamentally distinct ways of evaluating speech synthesizers: one is to use human subjects; the other to automate the evaluation process. Nowadays, it is necessary to employ humans in the evaluation process, mainly to evaluate the quality of integration and functionality of the system.
- *Global vs. analytic* approach tell us whether we rate global aspects such as naturalness and overall quality or more analytic aspects like vowel and consonant clarity, tempo or appropriateness of stresses and accents.

A critical measurement of a TTS system is whether or not human listeners can understand the text read by the system. So called *intelligibility* tests were developed to measure this aspect. For example Diagnostic Rhyme Test is used that uses set composed of pairs of similair words. Those are synthesized with a TTS engine and the listeners try to correctly distinguish between them. The intelligibility is the evaluated and compared to listening human speakers. An example set of words that may be used for such test is given in Figure 3.3.

Voicing		Nasality		Sustenation		Sibilation		Graveness		Compactness	
veal	feel	meat	beat	vee	bee	zee	thee	weed	reed	yield	wield
bean	peen	need	deed	sheet	cheat	cheep	keep	peak	teak	key	tea
gin	chin	mitt	bit	vill	bill	jilt	gilt	bid	did	hit	fit
dint	tint	nip	dip	thick	tick	sing	thing	fin	thin	gill	dill
zoo	sue	moot	boot	foo	pooh	juice	goose	moon	noon	coop	poop
dune	tune	news	dues	shoes	choose	chew	coo	pool	tool	you	rue
vole	foal	moan	bone	those	doze	joe	go	bowl	dole	ghost	boast
goat	coat	note	dote	though	dough	sole	thole	fore	thor	show	so
zed	said	mend	bend	then	den	jest	guest	met	net	keg	peg
dense	tense	neck	deck	fence	pence	chair	care	pent	tent	yen	wren
vast	fast	mad	bad	than	dan	jab	gab	bank	dank	gat	bat
gaff	calf	nab	dab	shad	chad	sank	thank	fad	thad	shag	sag
vault	fault	moss	boss	thong	tong	jaws	gauze	fought	thought	yawl	wall
daunt	taunt	gnaw	daw	shaw	chaw	saw	thaw	bong	dong	caught	thought
jock	chock	mom	bomb	von	bon	jot	got	wad	rod	hop	fop
bond	pond	knock	dock	vox	box	chop	cop	pot	tot	got	dot

Figure 3.4: An example set of words used to perform the Diagnostic Rhyme Test

While a TTS system has to be intelligible, this does not guarantee user acceptance, because its quality may be far from that of a human speaker. That is the

reason, why methods like Mean Opinion Score (MOS) are frequently used. MOS is human-subject judgment testing that asks the subjects to rate several samples on the scale of 1 to 5 (1 = Bad, 2 = Poor, 3 = Fair, 4 = Good, 5 = Excellent). The scores are averaged, resulting in an overall MOS rating. We have used this method of labeling to annotate our sample dataset.

Normalized MOS scores for different TTS systems can be obtained without any preference judgments. Nevertheless, sometimes comparisons are desired, especially for systems that are informally judged to be fairly close in quality. So called Category Rating (CCR) method, may be used. In this method, listeners are presented with a pair of speech samples on each trial. The order of the system A system B samples is chosen at random for each trial, providing, that half of the trials, the system A sample is followed by the system B and the order is reversed in the remaining trials. Listeners rate the pair with labels meaning, that the second utterance is:

- 3 - much better
- 2 - better
- 1 slightly better
- 0 about the same
- -1 slightly worse
- -2 worse
- -3 much worse

Sometimes the granularity can be reduced as much as simply "prefer A/prefer B". We use this method to tell which transcription is better.

3.4 Evaluating classifiers

Basic measure that is used to evaluate classifiers' performance is *accuracy*. It is a fraction of correctly classified examples over the total number of examples. However, more descriptive measures are need sometimes. In this work, we use the *F-score*, which is computed using terms *precision* and *recall* [?]. To explain this terms, we first need to introduce some others. Suppose, we have binary classification task, that is we want to identify positive and negative examples. In this settings, four types of situations may occur:

- *True Positive(TP)* - classifier identified positive example as positive
- *False Positive(FP)* - classifier identified negative example as positive
- *True Negative(TN)* - classifier identified negative example as negative
- *False Negative(FN)* - classifier identified positive examples as negative

With this terminology, we can define:

$$precision = \frac{TP}{TP + FP} \quad (3.3)$$

$$recall = \frac{TP}{TP + FN} \quad (3.4)$$

In other words, *precision* describes how many selected examples are relevant, while *recall* tells us how many relevant examples were actually selected. Finally, we can define:

$$F - score = 2 \times \frac{precision \times recall}{precision + recall} \quad (3.5)$$

That is, the *F-score*, sometimes called *F₁ score* is a harmonic mean of *precision* and *recall*.

4. Experiments

4.1 Used Datasets

We are interested in datasets that would contain a lot of Out-of-vocabulary words. This is desirable for us, because it means the dataset contains a lot of words, that are not present in a vocabulary of the speech recognition and text-to-speech (TTS) that we have used. When it comes to pronunciation of such words, text-to-speech usually performs poorly. They should be difficult to pronounce for the TTS engine and thus there is space for an improvement. We created two datasets ourselves, we call them *artificial* in the following text.

4.1.1 Dataset D_1 - artificial

Dataset D_1 contains 100 randomly chosen Dutch Names. We take the orthographic transcription of these names and create two phonetic transcriptions. The first is obtained from our *g2p* module, the other is created with the procedure described in 4.3.3. Both transcriptions were then synthesized using the *Cereproc* engine and rated by judges using the CCR method introduced in 3.3.

4.1.2 Dataset D_2 - artificial

D_2 is also manually labeled set of 110 words. 70 words are English, remaining 40 are Czech. Czech words were chosen to contain lot of diacritic hence to be difficult to pronounce for TTS trained on English. Each word was synthesized by four TTS engines, namely *Cereproc*, *MaryTTS*, *gTTS* and *Pico*. Each recording was then manually annotated.

Labeling . Labeling of D_2 was performed by three annotators. Each of them had to listen every word four times, each time synthesized by different engine. He then labeled each of the recording with a discrete number ranging from one to five. The recordings labels were averaged. Thus we obtain labels per each recording and we have four labels in total for each word. We average these four labels and obtain a score between one and five, that describes overall quality of the word's pronunciation. We thus obtain data that are suitable for our purpose, meaning, that we can explore methods of the label prediction and find out, if they correspond to the raters' opinion.

4.1.3 Dataset D_3 - Autonomata Spoken Name Corpus

The Autonomata Spoken Names Corpus [28] is a database of approximately 5.000 read-aloud first names, surnames, street names, city names and control words. It was published by Dutch-Flemish HLT Agency¹. The corpus consists of a Dutch part and a Flemish part. Besides the speech data, the corpus also comprises phonetic transcriptions, speaker information and lists containing the orthography of the read names. Spoken utterances of 240 speakers living in the Netherlands

¹<http://tst-centrale.org/nl/>

Annotators	Rec. 1	Rec. 2	Rec. 3	Rec. 4	mean
A_1 vs A_2	0.85	0.77	0.85	0.63	0.78
A_1 vs A_3	0.84	0.82	0.83	0.67	0.79
A_2 vs A_3	0.92	0.89	0.93	0.88	0.91
mean	0.87	0.83	0.87	0.73	

Table 4.1: The table contains correlations that were observed among the raters.

(NL) or in Flanders (FL) are included in the corpus. Since we have both orthographic and phonetic transcriptions available in addition to the audio recording, the dataset is ideal for our purposes, because we can evaluate the quality of our transcriptions.

4.1.4 Annotation procedure

We have collected some datasets (D_1 , D_2) ourselves and we had to obtain ratings for it. Hence, we asked three annotators to label the recordings. They were given instructions to judge the recordings with respect to quality of the pronunciation, not the sound quality of the recording. However, the criteria were defined quite vaguely. The raters were not told which aspects of pronunciation they should focus on. Therefore, the ratings may be subjective according to respective annotators' preferences. Because of that, we have explored how individual annotators' ratings correlate. We sum up the results for the D_1 dataset in TODO and for the D_2 in Table 4.1.4.

4.2 Identifying difficult words

If we want to improve pronunciation of the synthesized recordings, we should first identify words that are mispronounced. Obviously, we could let the TTS system pronounce each word in a best effort style and the user would identify mistakes and correct them. This approach has several drawbacks. First, the communication with the user is rather complicated, since the incorrectly pronounced word has to be isolated prior to obtaining the correct pronunciation. Second, it may be unpleasant for the user if he or she has to undergo this process and hear the incorrect pronunciation. Thus, it would be better if we could recognize the possibly difficult words somehow. It would mean that we can ask the user directly for the correct pronunciation of the word. We explore this issue in the rest of this section.

4.2.1 Measuring the difficulty

In order to estimate the difficulty of each word's pronunciation, we propose three measures in this section. The key idea is that we obtain values for each of this measures and then combine them in one feature vector that represents the recording. Then we can train a classifier that learns to predict quality of the pronunciation.

M_1 measure - averaged Mel Cepstral Distortion

We discuss Mel Cepstral Distortion in detail in 2.2. We use it here to define the M_1 measure. We can describe it as follows:

$$M_1(k) = \frac{1}{N} \sum_{(i,j)} MCD(r_{ki}, r_{kj}) \quad (4.1)$$

Where $N = \binom{3}{2}$, (i, j) stands for every combination of synthesizers, r_{ki} is name k synthesized by engine i and MCD is the Mel Cepstral Distortion. The key idea motivating this measure is that if there is a problem with a pronunciation of some part of the word, every synthesizer has to deal with it somehow. It is likely, that each of them will do it in slightly different way. This implies that the pairwise MCD will increase.

Nevertheless, the process of computing the value of M_1 introduces some problems. The troubles stem from the fact, that there is a high variability in the data obtained from the synthesizers. That is, one synthesizer's output may differ greatly from the others, presumably if there is a big flaw in this synthesizer's pronunciation. This influence the resulting measure heavily. We can see the effect in Figure 4.2.1. A solution to this problem may be, that we detect the outlying values and get rid of them. However, it is unclear, whether this is always desired. It can be the case, that the big difference between synthesizers' outputs occurs because the pronunciation is really difficult.

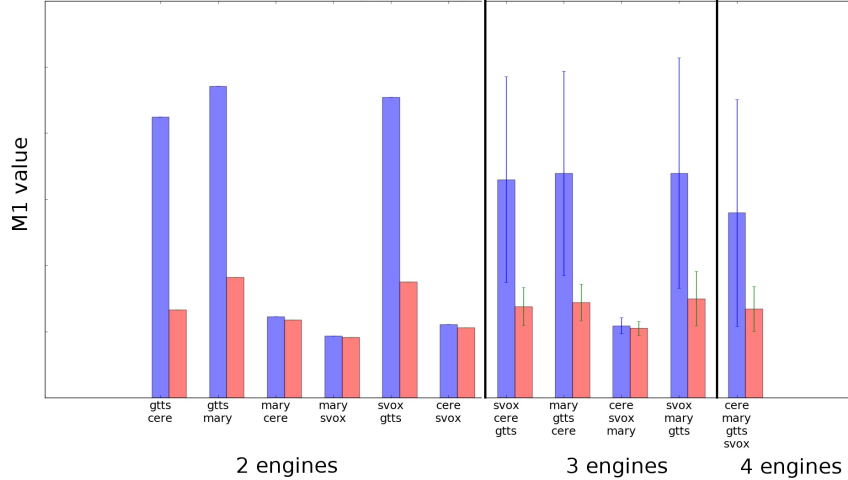


Figure 4.1: An example demonstrating, how the measured M_1 between two recordings can vary. Red and blue bars represent respective recordings, their heights correspond to the measured M_1 . On the left, there are measurements using only two synthesizers ($\sim MCD$), in the between, there are triples and the most right column corresponds to four synthesizers. One outlying example can negatively influence the relevancy of the measure. While the recording represented by the red columns has quite similar MCD between every pair, the other one has outliers and thus the resulting value has got low confidence.

We tested, how good is the correlation of M_1 measure and human judgment.

First, we synthesized the words by different synthesizers. Each recording created this way was then labeled by a human. Set of recordings for each word was then used to compute M_1 value and this was compared with the mean of the respective human ratings. Results on our sample dataset with the use of the 0-th coefficient and without it are shown in Figure 4.2.1. Based on these results, it may seem that the M_1 measure is not very good, the best correlation was 0.471. However we further combine it with other measures. In section 4.2.2 we show, that it adds a useful piece of information to the feature vector.

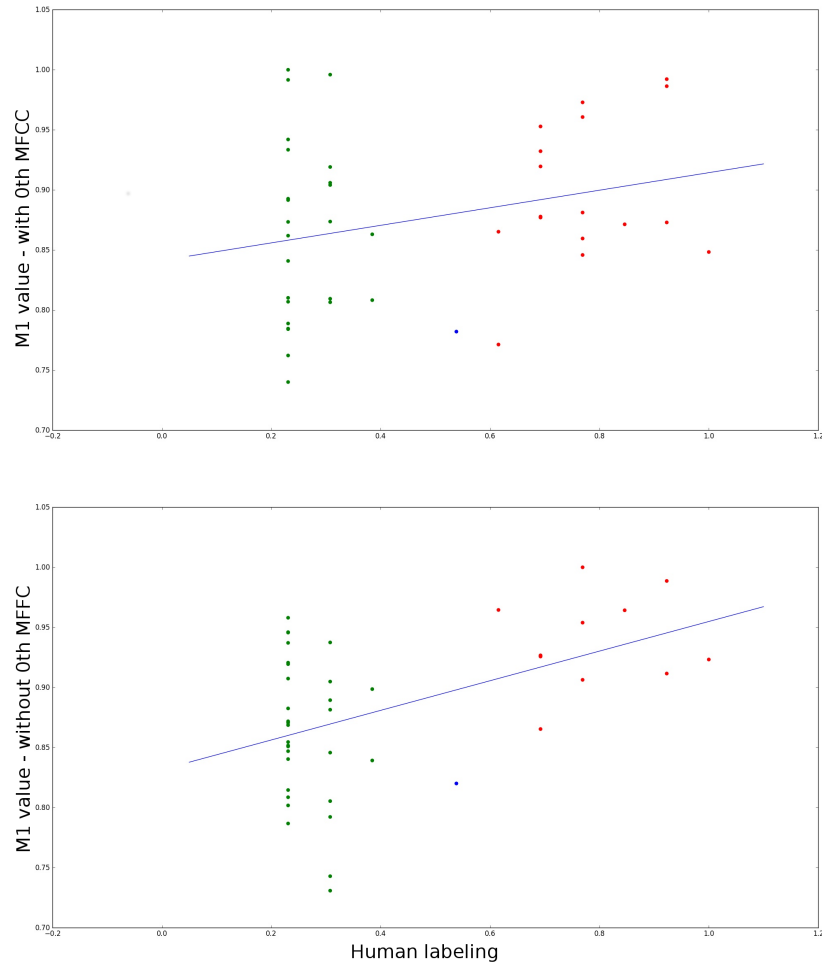


Figure 4.2: Plot of the M_1 measure correlation with human judgments. Points are horizontally spaced and colored according to gold labels. The vertical axis shows M_1 value. The blue lines represents estimate yielded by least square fit. The top figure includes the 0-th mel cepstral coefficient, while the figure on the bottom does not.

The *MCD* has one property that can be looked at as a disadvantage: It does not weight its coefficients, more accurately it weights all with the same weights. We propose to change this - for example a linear model could be trained, that finds the combination of MFCC's that corresponds the best. The coefficients derived by this model would be difficult to interpret without further research. However, we can train it and try to use it instead of typical *MCD*.

We want to supply the computation of M_1 by a linear model prediction. To train such a model, we must first prepare the data. We use the D_1 dataset described in section 4.1.

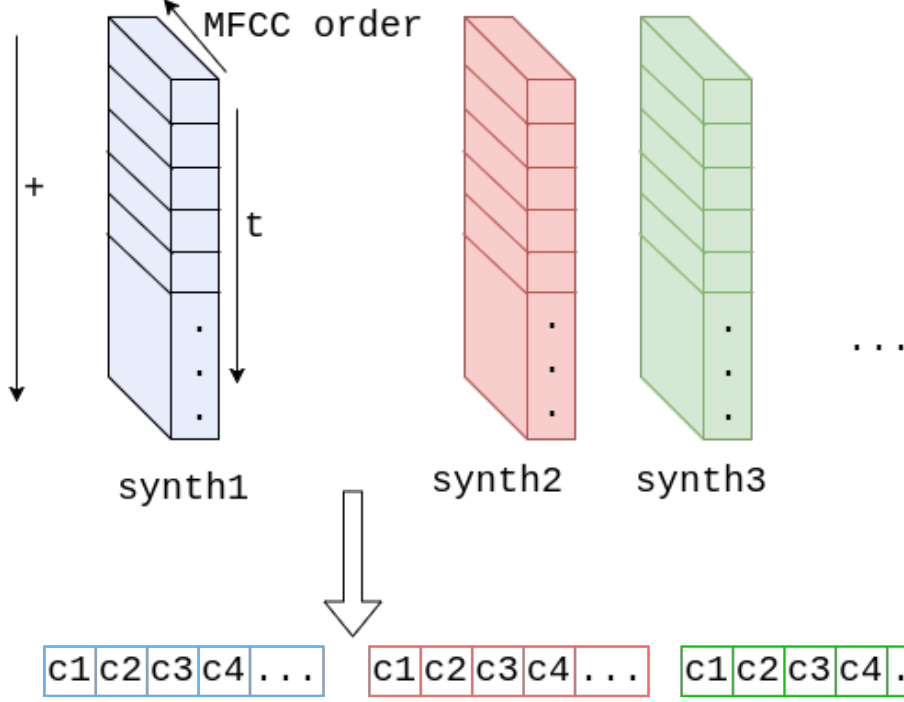


Figure 4.3: The process of aggregating the values obtained from mel cepstral analysis. Each recording is represented by sequence of M -sized vectors. These sequences are summed, thus the representation is flattened into one vector of length M .

Since we want to use a simple linear model, we need to work with fixed size vectors. Nevertheless, the input data are variable-length sequences of vectors. We first transform each synthesized recording by summing all its mel cepstral coefficients. Thus we get N M -sized vectors, where N is the number of synthesizers used and M is the order of mel cepstral analysis. This step is visualized in Figure 4.2.1. We then compute differences in each position between every pair of recognizers and sum those to obtain one resulting vector of length M . We use labels obtained from annotators normalized to interval $[0, 1]$ as a target variable. We then evaluate the model using 5-fold cross-validation. The results are shown in Figure 4.2.1. The correlations and R^2 for respective folds are shown in Table 4.2.1.

M_2 measure - averaged phonetic distance

Another measure we can possibly use is based on phonetic transcriptions. We first recognize the recordings with a phoneme recognizer, thus we obtain a sequence of characters per each recording. We can compute pairwise distances of these transcriptions, using Levenshtein [29] distance which we normalize by the length of the word. The motivation is similar to the M_1 measure. Assuming the difficult words have positions that are problematic for the TTS engine, the recognized

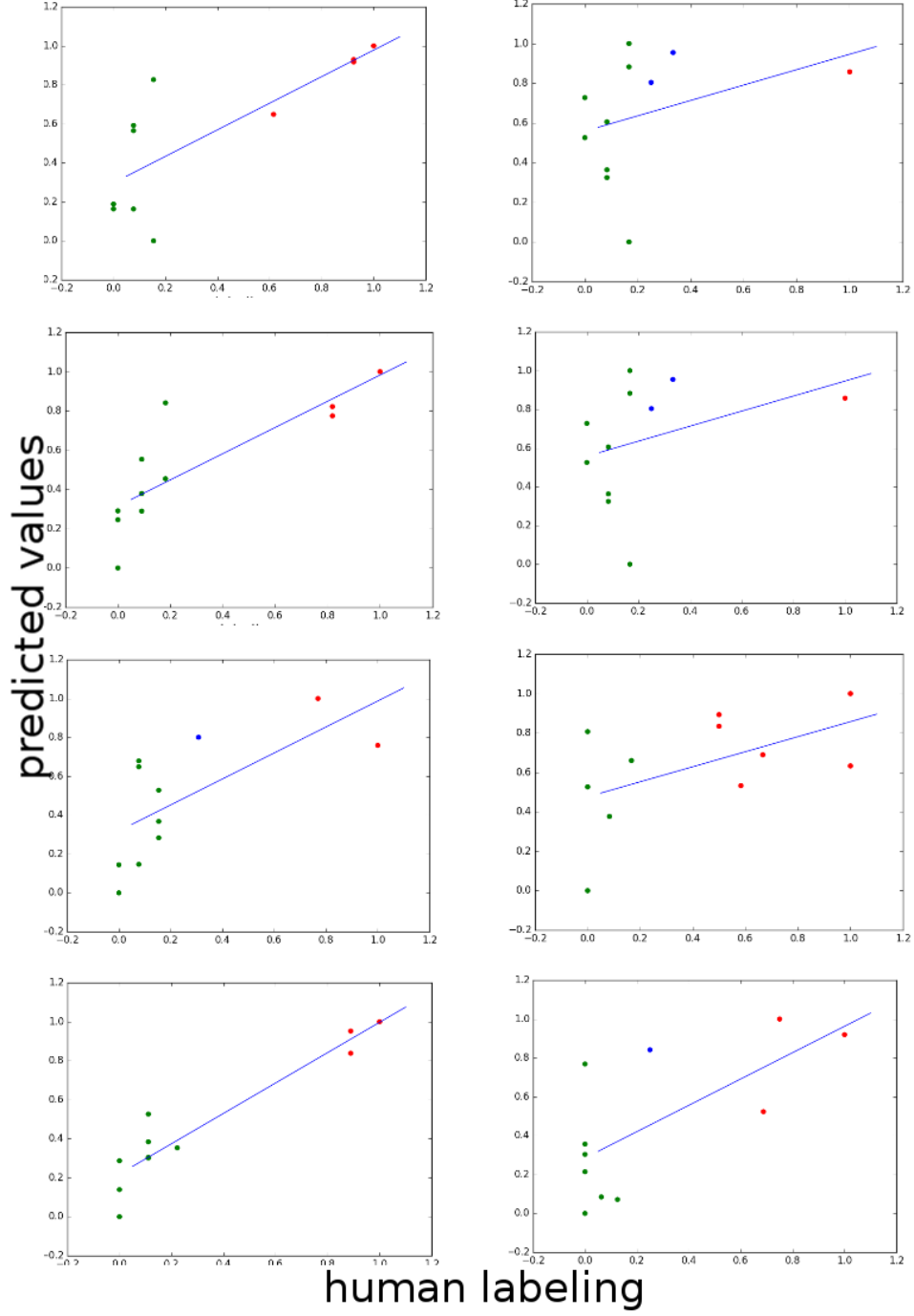


Figure 4.4: Plots of the correlation of the measure computed by model in cross-validation folds. The predicted values are on the vertical axis, the horizontal position represents the human judgments. The blue lines shows least-squares fit. Although the training data are rather small, it shows, that the model can output meaningful values.

phonemes on these positions should differ and thus the distance between transcriptions should increase. For our purposes, we have used Levenshtein distance as a metric.

Fold	Correlation	R^2
1	0.77	0.74
2	0.03	0.75
3	0.82	0.76
4	0.35	0.77
5	0.69	0.74
6	0.53	0.72
7	0.94	0.75
8	0.68	0.73
Mean	0.60	0.75

Table 4.2: The table contains results of linear regression model training on aggregated MFCC vectors. Correlation with human judgment and R^2 measure for respective cross-validation folds are shown.

The M_2 measure is described by equation:

$$M_2(k) = \frac{1}{N} \sum_{(i,j)} \frac{LD(Hyp(r_{ki}), Hyp(r_{kj}))}{\max(len(r_{ki}), len(r_{kj}))} \quad (4.2)$$

Where $N = \binom{3}{2}$, (i, j) stands for every combination of synthesizers and r_{ki} is name $\{k\}$ synthesized by engine $\{i\}$. LD means Levenshtein Distance and Hyp represents the best hypothesis from the phonetic recognizer. Note, that we normalize the distance by length of the longer transcription.

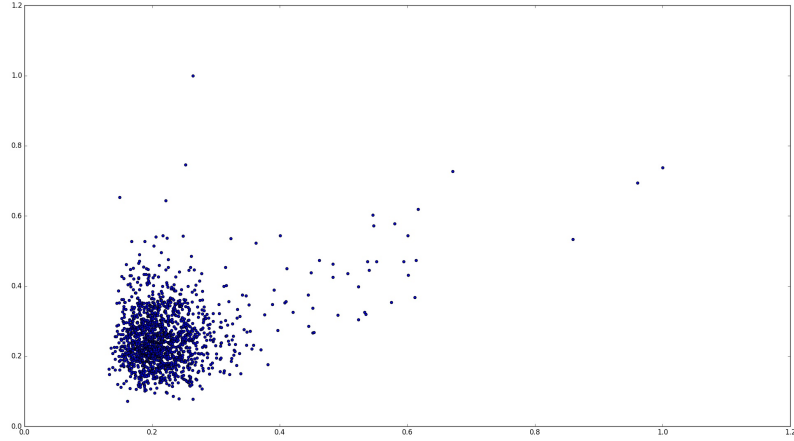


Figure 4.5: Scatterplot showing relation between the M_1 and M_2 measures. We can see, that the measures are rather uncorrelated. However, they can still be complementary.

M_3 measure - occurrence of confusable bigrams

The M_3 measure is based on a different approach. We want to learn the typical mistakes of a grapheme-to-phoneme converter, more specifically we want to

explore which groups of graphemes are difficult to transcribe for the *g2p*. We suppose that words with many occurrences of such groups are difficult to pronounce. To compute the measure, we need two corpuses (C_1 and C_2) in different languages and a *grapheme – to – phoneme* (*g2p*) training algorithm. The corpuses should consist of pairs (word, transcription). We can then prepare for computation of M_3 in three stages:

1. Train *g2p* model G_1 on corpus C_1 in language L_1 .
2. Use trained G_1 to transcribe words contained in corpus C_2 .
3. Identify problematic parts.

Stage 3 needs further description.

We obtained list of triples, i^{th} of which is structured:

(word w_i from C_2 , original transcription t_i^o , transcription t_i^h derived by G_1)

We can then use the Dynamic Time Warping algorithm to obtain pairwise alignments of these sequences, illustrated on Figure 4.2.1 In fact, we first transform the phoneme sequences into sequences of bigrams. This is because graphemes and phonemes are not in one-to-one correspondence and bigrams capture the relations better. Once we have the alignments, we can identify positions, in which

h	ε	l	ə	ʊ
h	ə	l	o	ʊ

Figure 4.6: Sample alignment of two phonetic sequences

the original transcriptions from C_2 differ from the hypothesis derived in Stage 2. Respective bigrams from the original word can then be identified. For each bigram, we count number of times it was marked as difficult. Each word can then be scored according to number of bad bigrams it contains.

To evaluate the M_3 measure, we have to synthesize the words, have them rated and compare the ratings with the measure. MaryTTS framework was used, because we can extract phonetic transcriptions from it and thus use its *g2p* module. This means that we are able to derive the transcriptions with exactly the same module that is used in the synthesizer. We have also used the Cereproc engine. The words were chosen as a subset of the Autonomata Corpus.

Because we do not have access to the *g2p* that Cereproc uses, we trained our own model on the CMU dictionary². We use a subsets of Autonomata corpus as the dataset of foreign worlds. The data was then processed and the confusion matrix was created, containing number of times, respective bigrams are confused with each other. The matrix turns out to be quite sparse, which is not surprising, since the majority of bigram pairs are interchanged with probability nearly zero.

If we restrict the matrix only to values greater than a certain threshold, its dimensions decrease dramatically and we can visualize it; the result can be seen in

²<http://www.speech.cs.cmu.edu/cgi-bin/cmudict>

Figure 4.2.1 The scoring procedure counts for each word a number of occurrences of bigrams that were confused and the number of confusions is summed and divided by the length of the word. We plot the scores in Figure 4.2.1. The shape of the curve corresponds to the fact that many bigrams do not occur a lot, or are not problematic.

The correlation of this measure and annotations was 0.20 for the MaryTTS and 0.31 for the Cereproc, respectively. Intuitively, if we have access to the *g2p* module, which is the MaryTTS case, it is expectable, that the results would be better. However, this turns out not to be true. It may be caused by the fact, that the recordings created by Cereproc are of better quality in general, so the ratings are better as well.

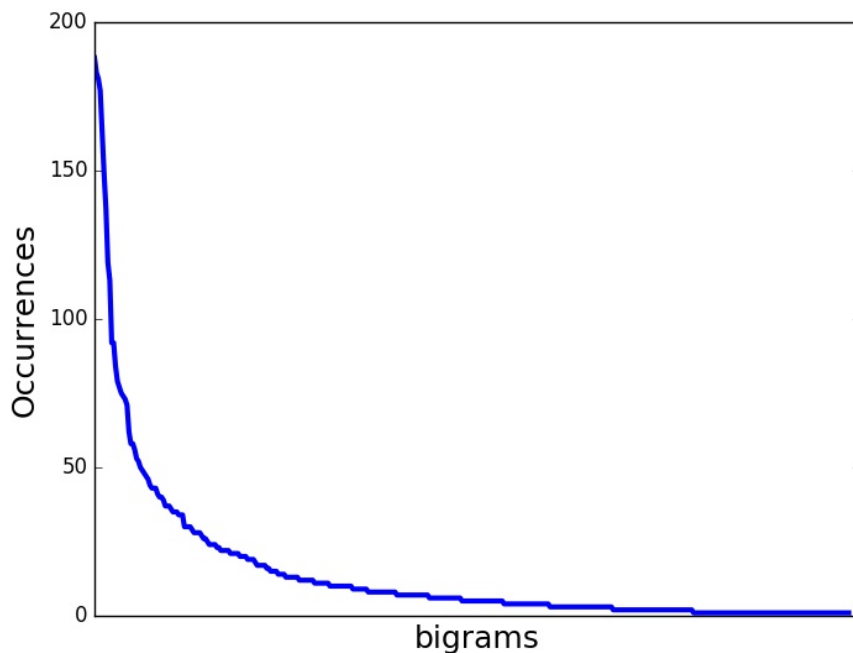


Figure 4.7: Frequencies of confusing respective bigrams, i.e. how many times it occurs in some confused pair. We can see, that the majority of bigrams has very low frequencies.

4.2.2 Measure combination

We have seen that neither of the measures correlates well with the annotators' labeling (Figure 4.2.1). Despite this fact, it may be interesting to explore, whether the combination of the measures can work better. As described in the introduction to this section, we combine the measured values to one feature vector that represents the word. However, it does not make much sense to include the M_3 measure, since it is language-dependent. Namely, it captures only the specific differences between English and Dutch. Since our dataset contains OOV words from Czech, we do not include M_3 to the feature vector. We use a *Ridge regression* [?] model, which is a regularized version of the linear regression. It adds to the objective function quadratic norm of the weight vector. We train the model using

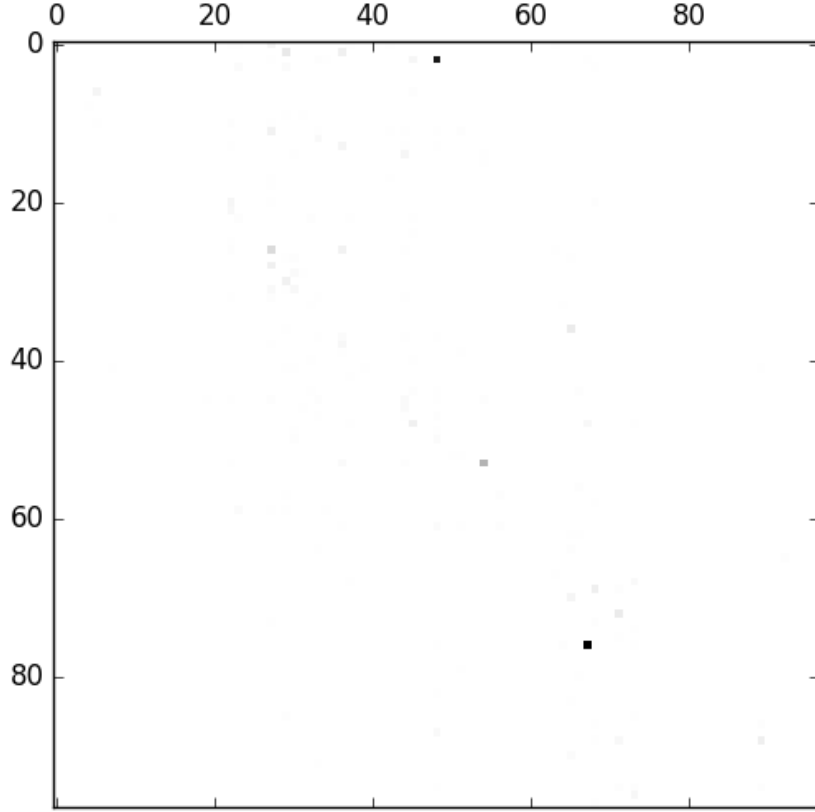


Figure 4.8: Visualization of the confusion matrix computed from the results.

averaged human labels as a target variable. We have used 5 fold cross validation and measured the R^2 and $MeanSquaredError(MSE)$, which we averaged over all the folds. The measured MSE was 0.39 and R^2 was 0.58. We remind, that the human labels are discrete values in the interval $[1, 5]$, so the MSE value is good, saying that we differ from the human label by 0.6 in average. We plot the differences from true labels in Figure 4.2.2.

We are interested not as much in the label prediction, as in the identification of words that are difficult to pronounce. To achieve this, we can choose a threshold and claim that if the model's predicted value is greater than the threshold, the respective word is hard to pronounce and we should try to improve the pronunciation. We could empirically determine a threshold separating difficult words from the easier ones. If we plot the results, as in Figure 4.2.2, we can see, that setting the threshold to value 2.5, which is exactly in the middle of the scale, is reasonable choice. Thus we can reformulate the problem as a binary classification task which gives us another insight. Simply, we have to decide if a given example will be pronounced correctly or not. We aim to identify incorrectly pronounced words and we refer to these words as positive examples.

Thus we can compute the *precision and recall* and obtain the F_1 score 0.86. Based on these experiments, our method has reasonable results and may be used

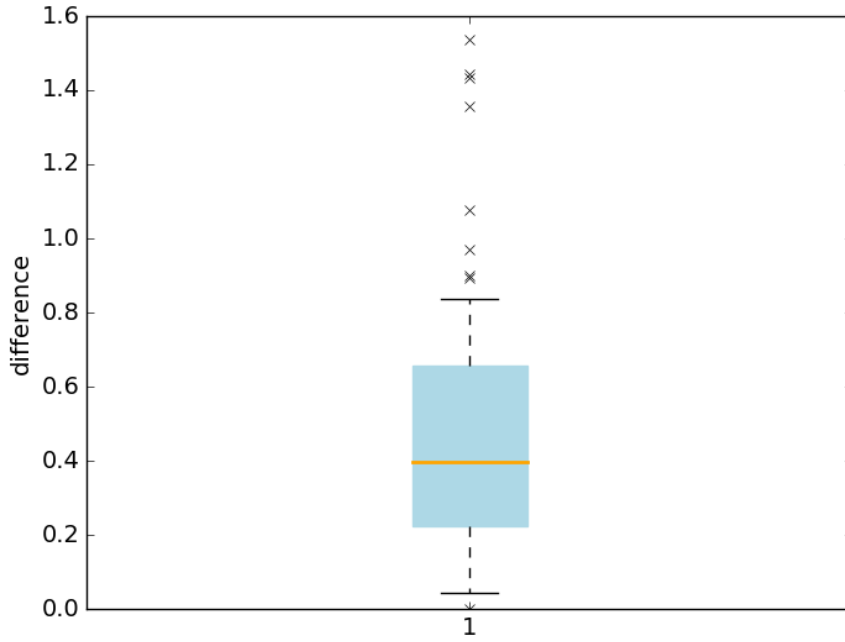


Figure 4.9: A boxplot showing difference of the predicted labels from human ratings.

to identify difficult recordings confidently.

Since we have approached the problem as a binary classification task, we can also train a classifier. We train a Logistic Regression classifier and determine a classification threshold by plotting the Receiver Operating Characteristics line - Figure 4.2.2. Since we want to identify the bad transcriptions, we focus on *Recall* more. We choose threshold as plotted and obtain F_1 score 0.85. This is slightly worse than the F_1 obtained from the linear regression model. Based on this fact, the linear regression model is at least comparable to the binary classifier in the classification task. In addition, it gives us additional information, that is a quantification of pronunciation difficulty.

4.2.3 Discussion

We proposed and explored three measures for a purpose of recognition of words that are difficult to pronounce for a TTS system. Our goal is to develop a method that would identify such words before they are introduced to the user. Exploration of these measures showed, that they do not correlate well with the human judgments. Nevertheless, the experiments indicate, that the measures are complementary in the sense, that if we combine them in a feature vector, we can train a model that predicts reasonable values. If we determine a threshold, we can use this model for classification purposes. Although the results are promising, the computation process that precedes the creation of the feature vector is not very convenient. We have to have access to speech engines, non-trivial phonetic corpora (M_3) and phoneme recognizer (M_2). We also have trained a model that substitutes the MCD computation. It yields meaningful results, however the

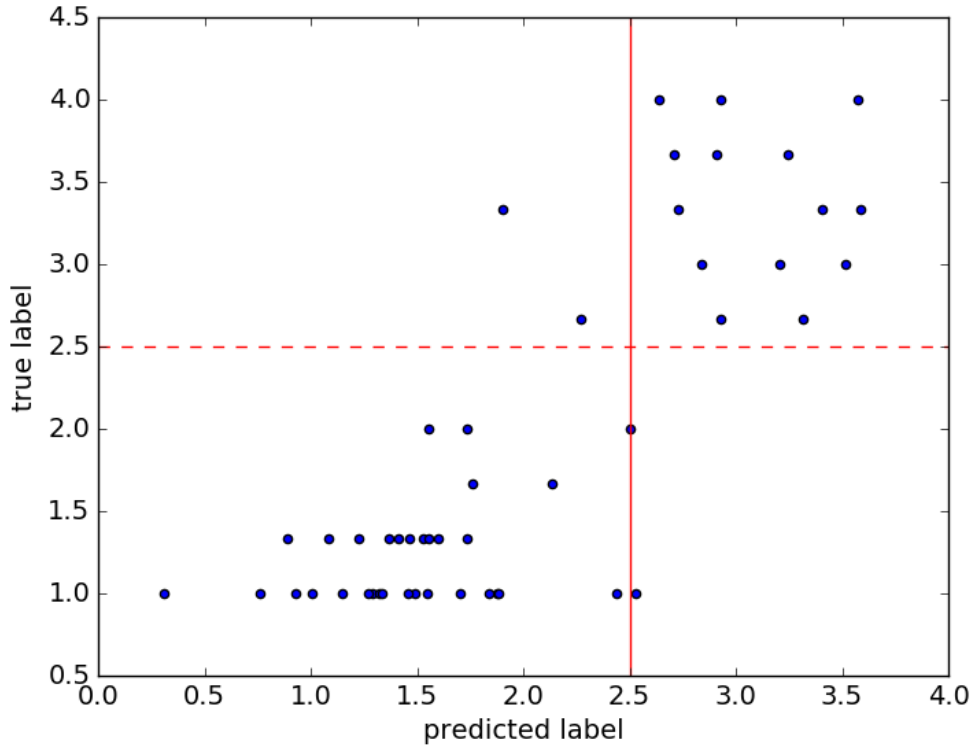


Figure 4.10: Plot of the labels given by humans (vertical axis) and predicted values (horizontal axis). The vertical line represents the threshold with respect to the predictions, the horizontal dashed line illustrates true division.

disadvantage of this approach is that we need labeled data, while the previous method works in an unsupervised way.

As for the M_3 , its disadvantage is that it relies quite heavily on the alignment process, which is imperfect. Because of that, it can sometimes mark as confused pair of bigrams that is not correctly aligned. Also if occurrences of some bigrams are very sparse, then its scores may be estimated poorly. Another possible problem is, that it relies on the corpora with phonetic transcriptions and then it is specific to certain pair of languages. It did not prove to be very useful.

4.3 Improving the pronunciation

4.3.1 Overview

We now briefly remind the task we consider. It may occur, that the TTS system pronounce certain words incorrectly. This is caused by the fact, that these words are not present in the vocabulary, that is, we do not have access to respective phonetic transcriptions. Thus we need to derive these transcriptions somehow. We can use a *g2p* module which we introduce in section 2.5.2. However as we discuss there, the *g2p* is not able to handle pronunciations of groups of graphemes that should be pronounced differently than in the language the TTS is trained on. Also, we do not have access to information, which language we should use to get

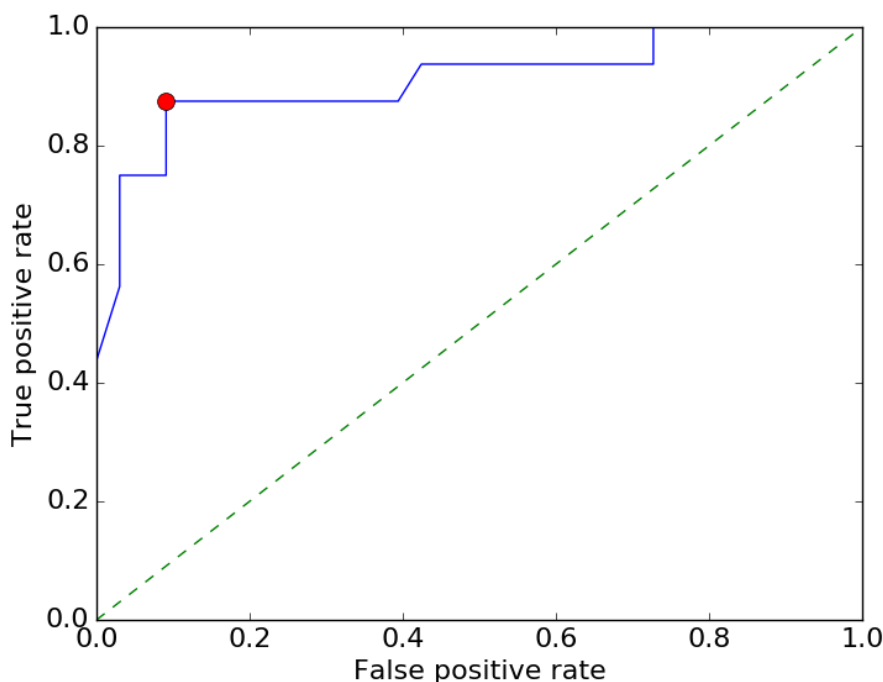


Figure 4.11: A Receiver Operating Characteristics line of Logistic Regression classifier. The green dashed line indicates a random choice, the red point corresponds to the chosen threshold.

the correct pronunciations. Therefore, the use of *g2p* is not possible or at least very problematic. We have also discussed methods (section 2.7.2), that derive pronunciation from spoken examples directly. However, such methods usually require quite substantial modifications to the speech recognizer.

4.3.2 Exploiting the speech recognizer

We try to exploit information obtained from the speech recognizer to improve the pronunciations. The very first step in the derivation process is to use the *g2p* module and obtain baseline phonetic transcriptions from it. Next, we need to get the phonetic transcription. Hence, we feed the spoken version of the utterance into a speech recognizer and decode output as we describe in section 3.2.1. In 3.2.1 we notice, that an interesting piece of information is contained in the recognizer's *n*-best list. We propose to exploit the information by clustering hypotheses in *n*-best list to a fixed number of clusters. Thus we obtain more compact yet sufficiently variable version of it. We then want to use this compressed list to choose the best transcription as we discuss in detail later.

Getting the recognizer

First of all, we have to get our speech recognizer. We use Kaldi³ speech recognition toolkit for the model training and subsequent decoding. In section 3.2.1 we describe, how we can modify the decoding graph to output phonemes. Thus

³<http://kaldi-asr.org/>

Decoder type	Phoneme Error Rate	Oracle PER (3best)	Oracle PER (5best)
DecoderVocab	90.1	-	-
DecoderBi	66.3	-	-
DecoderBi - chain	61.47	60.20	59.69
Timit	72.1	-	-

Table 4.3: Overview of results in terms of Phone Error Rate (PER) or Oracle PER, respectively. We measured the Oracle PER only on the most succesful decoder. The best results are showed in bold.

we are not limited by the choice of acoustic model, we simply do the necessary modifications to the decoding graph.

We have explored, how well different methods perform. In Table 4.3.2 we provide short summary of the results. Figure 4.3.2 shows dependency of the Oracle Phone Error Rate on the depth of the nbest list. The problem with evaluation is, that the reference must contain phonetic transcriptions. We have used our *g2p* module to obtain them. Thus our transcriptions are not ground truth and our evaluation is biased. However, the conditions are equal for each setting so it can help us to choose. We now present shortly the mentioned methods, details can be found in section 3.2.1:

- *DecoderVocab* was created by changing the vocabulary that is used for decoding. Thus, there does not exist a mapping from phoneme sequences to words as usual, each phoneme is mapped to itself instead. The acoustics were modeled using Gaussian Mixture Models⁴.
- *DecoderBi* refers to decoder created to model bigrams of phonemes, that is, it includes contextual information. The acoustics were modeled using Gaussian Mixture Models⁵.
- *DecoderBi – chain* has same type of decoding graph, but the acoustic part of the decoder was obtained from the Kaldi chain model⁶.
- *Timit* decoder was trained on the Timit dataset⁷ directly. This dataset has a vocabulary that contains phonemes, its language model is built on phonemes too. That means the decoder outputs phonetic representation of words.

For training, a subset of Librispeech corpus⁸ or Timit^{https://catalog.ldc.upenn.edu/ldc93s1} were used. The evaluation was done on the TODO:ptien-ny. Based on the results, we choose the decoder acoustic part of which is modelled by the chain model and the decoding graph models phones as bigrams.

⁴<http://kaldi-asr.org/doc/model.html>

⁵<http://kaldi-asr.org/doc/model.html>

⁶<http://kaldi-asr.org/doc/chain.html>

⁷<https://catalog.ldc.upenn.edu/ldc93s1>

⁸<http://www.openslr.org/12/>

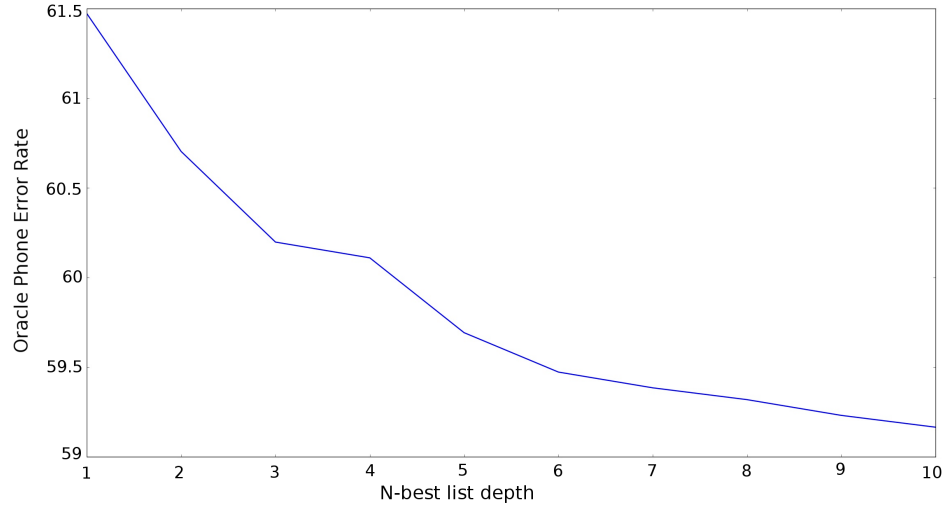


Figure 4.12: Dependency of decoder’s Oracle Phone Error Rate on the depth of the n -best list we explore. The bigger part of the list is exploited, the lower is the error rate.

Compressing the n -best list

To cluster the items, we first need to choose the clustering method. We explore two methods here, namely *kmeans* and *spectral clustering*. We now briefly discuss pros and cons of each.

- *Kmeans clustering* works with vectors in euclidean space and thus understandably uses *euclidean distance*. We thus transform the hypotheses into *count vectors*. *Count vector* is a vector that represents strings as euclidean vectors. Each position in it represents one character from the alphabet and the number on that position is the number of occurrences of this character in the string. This approach does not capture the context of the occurrences, but it can be extended to capture the occurrences of *bigrams* (i.e. character pairs) in the same way.
- In case of *spectral clustering*, we have to decide, which distance metric we will use. Since the hypotheses that we want to work with are sequences of phonemes (i.e. strings), either *Hamming*⁹ or *Levenshtein distance*¹⁰ come into consideration. However, the *Hamming* distance is not usable, because it counts number of different positions in two strings of the same length. Unfortunately, this is not the case of the hypotheses gathered from the decoder. Thus we use *Levenshtein* distance to be our metric.

For illustration, we give examples of the resulting clustering in Figure 4.3.2 or Figure 4.3.2, respectively. It is difficult to decide, which method is more appropriate. Based on results and observations, we choose the *kmeans clustering* to be our clustering method. The clusters derived by *kmeans* appears to be more similar and coherent, especially if we use longer list of hypotheses. We do not show longer lists here for readability purposes.

⁹https://en.wikipedia.org/wiki/Hamming_distance

¹⁰https://en.wikipedia.org/wiki/Levenshtein_distance

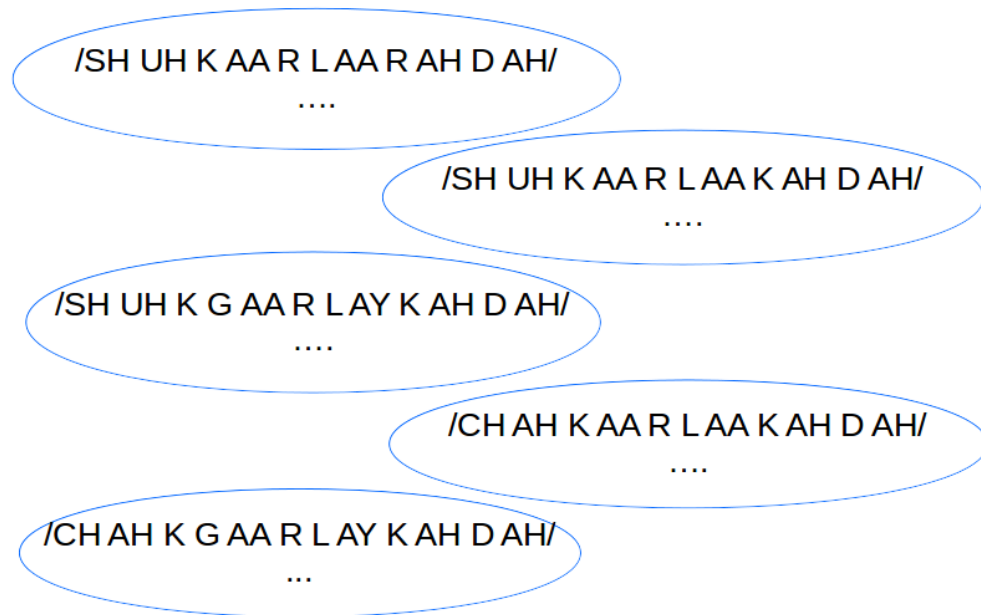


Figure 4.13: An example of clustering hypotheses of the phone recognizer with spectral clustering algorithm. The input was spoken czech word "Čokoláda".

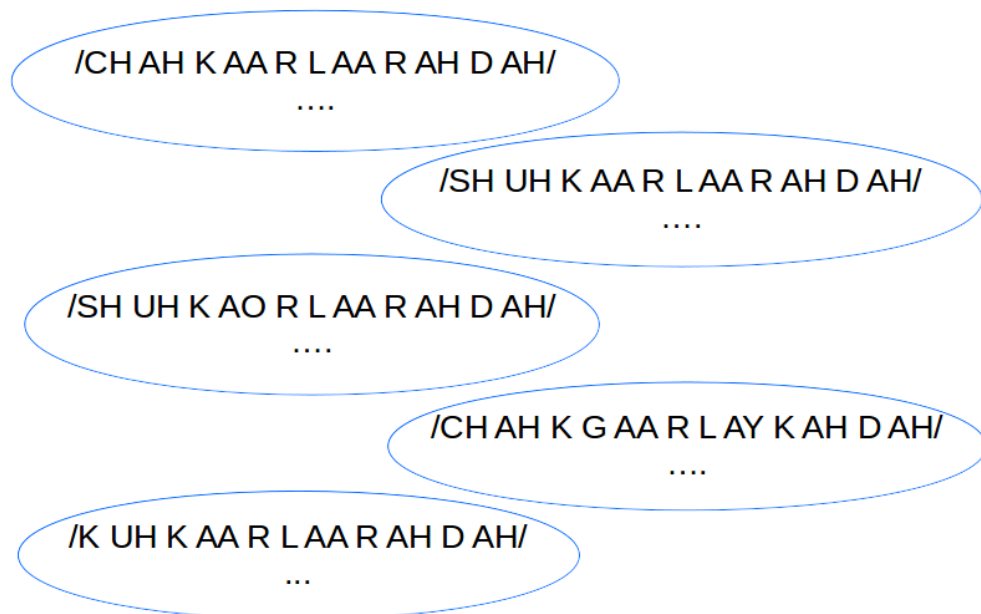


Figure 4.14: An example of clustering hypotheses of the phone recognizer with kmeans clustering algorithm. The input was spoken czech word "Čokoláda".

Choosing the best transcription

We are now in the situation that we have a fixed-length list of hypotheses and we want to choose the most appropriate one. Therefore, we formulated the task as a problem of classification. How can we determine the best hypothesis? We want

to choose one that is the closest to the reference phonetic transcription in terms of *Levenshtein* distance.

We want to approach the problem as a machine learning task - train a model, that classifies a list of N items (hypotheses) to class $1 \dots N$, provided that the list belongs to class k if the hypothesis on the k -th position is the best. This is just different way of saying, that we want to choose the best hypothesis among others. Nevertheless, the list of hypotheses alone does not provide sufficient information for the classifier. Hence, we base our decision not only on the list of hypotheses but also on the orthographic form of the word.

In this experiment, we fix the length of the list to five. The data for this experiment comes from the D_3 dataset. Each spoken name was recognized with our recognizer, 20-best list was clusterized and 5 representatives were chosen. In order to explore the model behavior we actually created two datasets. The first one was created according to the procedure we have described previously. In the second one however, the reference phonetic transcription has been mixed among the hypotheses, replacing one of the alternatives. The latter setting should be much easier to deal with, because the reference transcription is different from the rest. Ideally, the reference should be chosen every time. We include the second version of the dataset to find out, if the model has an ability to choose correctly using our proposed input.

The sequences can vary in length, but the models need the inputs to be of fixed length. Thus we have to encode the data somehow to achieve this. One option is to use the Bag of N -grams (BoN) technique. BoN is similar to Bag of Words¹¹. It is used to transform arbitrary-length finite strings coming from a finite alphabet into fixed length vectors. First, the number of occurrences of respective n -grams in the sequence is counted. Since the alphabet is finite, the number of n -grams is limited and we can construct an integer vector and assign each n -gram fixed position in it (i.e. enumerate the n -grams). The desired semantics is, that the number on the i^{th} position expresses the number of occurrences of the i^{th} n -gram in the string. We provide example of the decoding in Figure 4.3.2.

In general, the n can be chosen arbitrarily, nevertheless, the length of the vector increases exponentially with respect to n , because number of n -grams over an alphabet of size m is m^n . High dimensional vectors are not very suitable for most of machine learning techniques because of the data sparsity and many model parameters. Thus we choose $n \leq 3$. In our experiments we include all shorter n -grams in the feature vector, i.e. if we choose $n = 3$ we track not only number of trigrams but also bigrams and unigrams.

After transformation to the fixed-length vectors, the encoded hypotheses are shuffled and concatenated, so a feature vector is formed. The items are shuffled to prevent the model learning to give meaning to the ordering. Rather it should learn to choose the best transcription according to its relationship with other transcriptions and the reference. Supposing that we have used l hypotheses, the feature vector looks like:

$$h_{\pi(1)} \cdot h_{\pi(2)} \cdot \dots \cdot h_{\pi(l)}$$

Where $\pi : \{1 \dots l\} \rightarrow \{1 \dots l\}$ is a random permutation and the dots between hypotheses represents a concatenation operation, no separator characters are used.

¹¹https://en.wikipedia.org/wiki/Bag-of-words_model

unigram	order	word: <i>moon</i>			
m	1	Encoding:			
n	2				
o	3				
p	4				
		1 (m)	2 (n)	3 (o)	4 (p)
		1	1	2	0

Figure 4.15: An example of encoding word “moon” using BoN technique with $n = 1$. The reduced Latin alphabet was used.

Classifier	Acc. with gold	Acc. no gold
Most Frequent Class	0.26	0.43
Linear regression	0.987	0.53
Random Forest	0.99	0.40
SVC linear	0.98	0.48
SVC rbf	0.97	0.41
SVC polynomial	0.26	0.41
Multi Layer Perceptron	0.99	0.45

Table 4.4: Test set accuracies of various models both with and without the reference transcription included. The baseline is set by trivial classifier, which assigns each example the most frequent label. Red values indicates, that the model does not outperform the baseline, bold values are the best achieved results.

Thus, if the size of alphabet is K , we choose a maximal n -gram and use l hypotheses, the length of each feature vector is $L = (K + K^2 + \dots + K^n) \times (l + 1)$.

Next, we have to add target labels to the feature vectors. Levenshtein distance of each hypothesis and reference phonetic transcription is computed and the hypothesis with the smallest distance is marked as the best. The index of this hypothesis is then used as a target label. It is correct to use a reference transcription in the data preparation phase to derive a label, it does not mean that we use it in the training phase.

To solve the classification task, machine learning theory gives us a large portfolio of methods. We try several classifiers, namely Support Vector Machines, Logistic Regression, Random Forest and Multi Layer Perceptron. In the experiment settings, we have used hypotheses lists of length 4. Datasets of different sizes were used, all created from a subset of the D_3 dataset. We can see the dependency of accuracy on the dataset size in the Figure 4.3.2. The data were split to training, validation and test sets in ratio 7 : 2 : 1. Overview of the respective accuracies is given in Table 4.3.2.

Discussion We can see, that the classifiers perform much better when the reference transcription is included in the data. This can be caused by the fact,

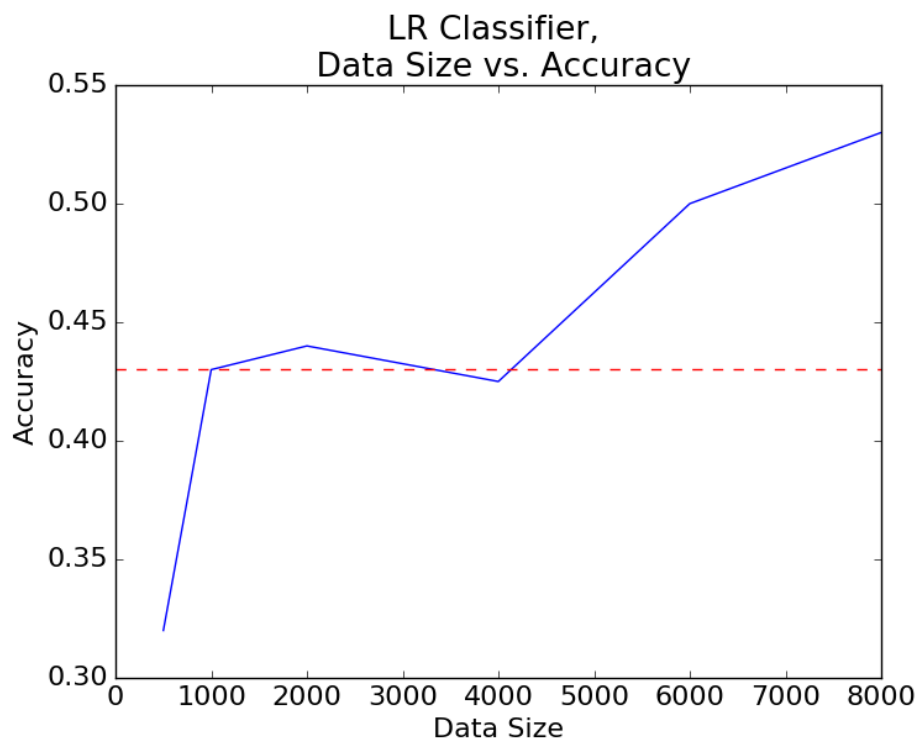


Figure 4.16: The dependency of the Linear Regression model accuracy on the size of the data it is trained on. The red dashed line represents the performance of trivial classifier. We can see, that with sufficient data, the model outperforms the baseline.

tha the reference transcription should be quite different from the rest, so it is supposed to be easily recognizable. Also, the distribution of classes is nearly uniform when the reference is included. This is in contrast with the real settings, because there the distribution is biased, even after shuffling as we can see in Figure 4.3.2.

Some of the models used did not even outperform the trivial classifier that chooses always the most frequent class. Nevertheless, some of them did, namely the Logistic Regression, Multi Layer Perceptron and Support Vector Classifier with linear kernel. Therefore it makes sense to include the process of selection of the best hypotheses from the n -best list.

4.3.3 Combining acoustic and textual information

In the previous, we have managed to derive phonetic transcriptions and pick the most appropriate one. Problem is that even the best transcriptions are too different from the desired ones. They cannot be used directly as input for the TTS. However, they capture pronunciations of some of the desired parts and thus can bring in some essential information. On the other hand, $g2p$'s output, is close to the correct transcription, just with some mistakes. As we discussed in section 2.5.2, it is difficult to correct these mistakes only with $g2p$, but there is a possibility to employ the phonetic transcriptions. The idea is to identify positions, where the $g2p$ is unsure and replace these with the respective counterparts in the speech recognizer's output.

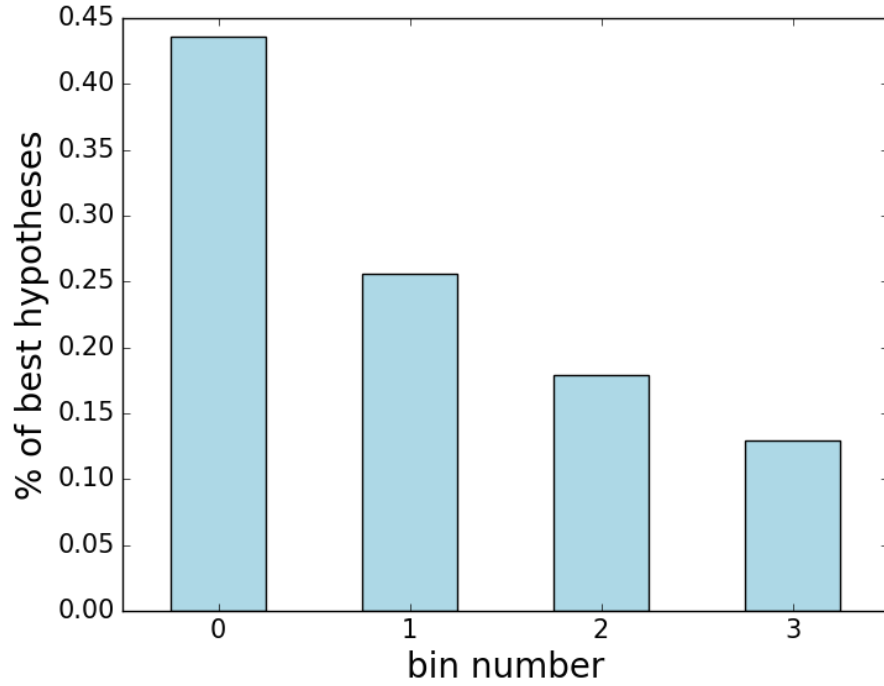


Figure 4.17: Distribution of target classes in the dataset. Although the hypotheses lists have been shuffled, the distribution is biased, making the task more difficult.

The question also arises, how to evaluate the acquired transcriptions once we have them. In section 3.3 we introduce the A-B testing. This is suitable for our purpose, however, we first have to choose, to what we want to compare. Clearly we cannot compare to the original transcription, since it is supposed to be the correct one. The baseline is actually set by the *g2p* output - we want to outperform it. Therefore we synthesize transcriptions from the *g2p* as well as the improved ones. And we let the raters choose which one is better.

One option how to identify such positions is to let the *g2p* give more alternatives. We then align those alternatives using Dynamic Time Warping Algorithm and pick positions, where many alternatives are possible. We assume, that confidence of such positions is low. Then we align the *g2p* outputs with the ASR transcription and do the replacement on respective positions.

We propose an alternative to the previous approach which is more exact. After some modifications to the *g2p* module, we can make it output posterior probabilities of the individual characters. Similarly, we can obtain posterior probabilities from the ASR decoder's output. That means, if we have transcription $\mathbf{W} = w_1 \dots w_n$, we obtain probabilities $\mathbf{P}(w_i | w_1 \dots w_{i-1})$. We can obtain an n -best lists from both the decoder and the *g2p* module. Thus we can combine the information and create Finite State Transducer the represents all the possible paths as illustrated in the Figure 4.3.3.

The idea is, that if the *g2p* has low confidence at some position i , we replace the position by the respective part of decoder's output, if it is more confident here. However, the problem is, that the transcriptions may be of different length. Thus we have to employ Dynamic Time Warping (DTW) algorithm again to align

the transcriptions.

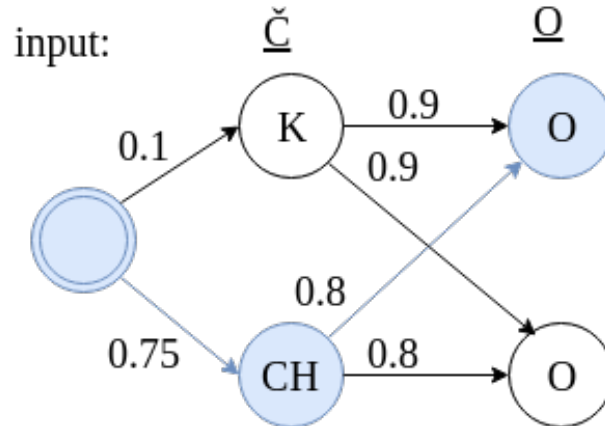


Figure 4.18: An illustrative example of the FST combined from the *g2p* (top) and speech decoder (bottom). A part of input is chosen corresponding to the syllable "ČO" and the chosen path is highlighted. It can be seen, that the path goes through both the decoder's and *g2p*'s hypotheses.

It is not straightforward how to use the DTW. Another problem is, that although we obtained posterior probabilities, the *g2p* and the decoder treat their graphs differently, namely the decoder assigns greater probabilities to the individual phonemes, thus restricting number of alternatives. Therefore, we have to weight the decoder's probabilities if we want to combine the results. We could for example try to train a model that would learn the weights. We left this research for future work. Nevertheless, we ran some preliminary experiments showing, that it is promising way to go, as showed in Figure 4.3.3.

original	<i>buxaʊtsʌlɛɪ</i>
<i>g2p</i>	boʊɛkaʊtsʌli
combined	boʊhaʊtsɛlɛɪ

Figure 4.19: An example of the preliminary result derived from combined FST. We can see, that the *g2p* transcription is rather incorrect. We highlight groups of phonemes, where the transcription is changed so it is more similar to the original. However we can see that some other phonemes were replaced incorrectly and the result is far from perfect yet. The transcriptions are given in IPA.

Discussion We have proposed two methods of combining the acoustic and textual information. Both methods suffer from the fact that the transcriptions are of different length and misaligned. The use of Dynamic Time Warping algorithm may not be good enough. The resulting alignments are not always desired since

it may pair two phonemes that should not be aligned. Also, the method that combines the FST's is not exact yet, mainly we did not derive the weights for the speech decoder's hypotheses. However, looking at the ratings and preliminary results, this is a promising way to go.

Conclusion

4.4 Summary and discussion

The problem of missing phonetic transcriptions (pronunciations) of words is a not solved issue yet. So called Out-of-vocabulary words still occur, causing incorrect pronunciation of the Text-to-speech systems. There are possibilities how to derive correct pronunciations of unknown words, using either textual (*g2p*) or acoustic information. We discuss it in section 2.7.

In this work, we try to come up with slightly different approach. First we explore the issue of identifying words that are difficult to pronounce in 4.2. This task is justified, because if the system can identify difficult words itself, it can initialize the process of obtaining the correct transcription and thus provide better user experience. We measure the difficulty using different synthesizer’s engines and exploit differences in their outputs. We also employ the knowledge of languages we can deal with to find out which confusions in the pronunciation are the most common. Using this data, we create a feature vector per each word, from which we learn to predict the difficulty. Some of the features are computed in the unsupervised way, which is important, because we thus do not need any labeled data.

Although the individual features does not correlate much with the human ratings, their combination give good results. However, the process of computing the features is quite demanding, because we need to synthesize the words and the process it. Also, the M_3 measure which relies on the language information suffers from the fact, that we need to know, which languages are compared, which is not true in general.

Second, we propose a method to combine the textual information from the *g2p* module with the acoustic information obtained from the phoneme recognizer. In 3.2.1 we notice, that the phoneme recognizer’s best hypothesis is not always the best in terms of phone error rate. We come up with procedure to extract substantial information from the recognizer’s n -best list and learn to choose the most appropriate hypothesis (4.3.2).

We then explore basic approach to the combination (4.3.3) and evaluate the results. We also propose further improvements of this method and discuss the drawbacks. The disadvantage of these approaches is in the fact that they rely on the alignment process quite heavily. We use Dynamic Time Warping, which sometimes creates alignments with mistakes thus represents a weak spot.

Another weak spot of our experiments are sizes of the datasets. We created the data ourselves and it had to be rated. Because of the difficulties of the procedure, the datasets are small. Thus the presented outcomes are rather proof of concept than strong results.

4.5 Future Work

Data If we want to confirm our results and make it more valuable, we have to obtain larger datasets. This process would involve employment of some crowd-sourcing platform and careful processing of the results.

Predicting Difficulty We should focus on simplification of the method proposed for the prediction of difficulty of the pronunciation. Methods of Machine Learning could be explored to substitute the current process. If it was successful, we would not need different synthesis engines. Also, regarding the M_3 computation process, it could be extended to consider multiple languages and thus become more universal.

Combining the text and acoustics In section 4.3.3 we have mentioned the future work in this area. We should explore, how the weighting factors should be derived when combining Finite State Transducers obtained from the $g2p$ and the decoder. Also, we should focus on the alignment process and use maybe use some more sophisticated method than Dynamic Time Warping.

Bibliography

- [1] Richard W Sproat. *Multilingual text-to-speech synthesis*. KLUWER academic publishers, 1997.
- [2] Xavier Anguera Miro, Peter Veprek, and Jean-claude Junqua. Multilingual text-to-speech system with limited resources, September 29 2009. US Patent 7,596,499.
- [3] Arpabet overview. <https://nlp.stanford.edu/courses/lisa352/arpabet.html>. Accessed: 2017-04-16.
- [4] P. Taylor. *Text-to-Speech Synthesis*. Cambridge University Press, 2009.
- [5] R Kubichek. Mel-cepstral distance measure for objective speech quality assessment. In *Communications, Computers and Signal Processing, 1993., IEEE Pacific Rim Conference on*, volume 1, pages 125–128. IEEE, 1993.
- [6] Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational linguistics*, 23(2):269–311, 1997.
- [7] Mehryar Mohri. Weighted automata algorithms. In *Handbook of weighted automata*, pages 213–254. Springer, 2009.
- [8] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR abs/1609.03499*, 2016.
- [9] Yuxuan Wang, RJ Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, et al. Tacotron: A fully end-to-end text-to-speech synthesis model. *arXiv preprint arXiv:1703.10135*, 2017.
- [10] Kaisheng Yao and Geoffrey Zweig. Sequence-to-sequence neural net models for grapheme-to-phoneme conversion. *arXiv preprint arXiv:1506.00196*, 2015.
- [11] Maximilian Bisani and Hermann Ney. Joint-sequence models for grapheme-to-phoneme conversion. *Speech Communication*, 50(5):434–451, 2008.
- [12] Tim Schlippe, Sebastian Ochs, and Tanja Schultz. Grapheme-to-phoneme model generation for indo-european languages. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 4801–4804. IEEE, 2012.
- [13] Paul Taylor and Amy Isard. Ssml: A speech synthesis markup language. *Speech communication*, 21(1-2):123–133, 1997.
- [14] Chotirat Ann Ratanamahatana and Eamonn Keogh. Everything you know about dynamic time warping is wrong. In *Third Workshop on Mining Temporal and Sequential Data*. Citeseer, 2004.

- [15] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856, 2002.
- [16] Ronald M Kaplan and Martin Kay. Regular models of phonological rule systems. *Computational linguistics*, 20(3):331–378, 1994.
- [17] Terrence J Sejnowski and Charles R Rosenberg. *NETtalk: A parallel network that learns to read aloud*. MIT Press, 1988.
- [18] Michael J Dedina and Howard C Nusbaum. Pronounce: a program for pronunciation by analogy. *Computer speech & language*, 5(1):55–64, 1991.
- [19] John Lucassen and Robert Mercer. An information theoretic approach to the automatic determination of phonemic baseforms. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP’84.*, volume 9, pages 304–307. IEEE, 1984.
- [20] Aliya Deri and Kevin Knight. Grapheme-to-phoneme models for (almost) any language. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 399–408, 2016.
- [21] Tilo Sloboda and Alex Waibel. Dictionary learning for spontaneous speech recognition. In *Spoken Language, 1996. ICSLP 96. Proceedings., Fourth International Conference on*, volume 4, pages 2328–2331. IEEE, 1996.
- [22] Ian McGraw, Ibrahim Badr, and James R Glass. Learning lexicons from speech using a pronunciation mixture model. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(2):357–366, 2013.
- [23] Sravana Reddy and Evandro B Gouvêa. Learning from mistakes: Expanding pronunciation lexicons using word recognition errors. In *INTERSPEECH*, pages 533–536, 2011.
- [24] Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88, 2002.
- [25] Hermann Ney and Stefan Ortmanns. Progress in dynamic programming search for lvcsr. *Proceedings of the IEEE*, 88(8):1224–1240, 2000.
- [26] Carla Lopes and Fernando Perdigao. Phoneme recognition on the timit database. In *Speech Technologies*. InTech, 2011.
- [27] X. Huang, A. Acero, and H.W. Hon. *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Prentice Hall PTR, 2001.
- [28] Henk Van Den Heuvel, Jean-Pierre Martens, Bart D’hoore, Kristof D’hanens, and Nanneke Konings. The autonomata spoken names corpus. In *LREC*, 2008.
- [29] Gonzalo Navarro. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):31–88, 2001.

List of Figures

1.1	A high level architecture of spoken dialogue system.	3
1.2	Sample dialogue illustrating how the pronunciation correction process may look like. The dialogue system pronounces user's name incorrectly. User then correct it and the system learns. The transcriptions of the user's name are given in ARPABET.	4
2.1	Example of Finite State Transducers and their composition	8
2.2	Overview of a typical statistical Text-to-speech system architecture. The text is first analysed and translated to phonetic transcription. Then, the acoustic parameters are estimated. Finally, the speech is synthesized.	11
2.3	Example of SSML code	12
2.4	An example of dendrogram outputted by hierarchical clustering methods.	14
2.5	Some letter of the IPA alphabet together with their occurrences in English words.	17
2.6	An example of the synthesizer's input in SSML, forcing it to use our phonetic transcription.	17
3.1	An example of the finite state transducer representing the lexicon[24]	23
3.2	A plot of Oracle Phone Error Rate dependency on the depth of the n -best list we explore. We can see, that the Oracle PER decreases with the depth, so it makes sense to try to exploit the hypotheses located deeper.	26
3.3	Histogram that visualizes the distribution of the hypothesis, that is the closest to the gold reference utterance in terms of Phone Error Rate. The bars show the number of times the best hypothesis was found on the particular position. Two settings are included in this figure: the green bars describe situation, when first five positions were used, the blue ones were measured using nine positions . . .	26
3.4	An example set of words used to perform the Diagnostic Rhyme Test	27
4.1	An example demonstrating, how the measured M_1 between two recordings can vary. Red and blue bars represent respective recordings, their heights correspond to the measured M_1 . On the left, there are measurings using only two synthesizers ($\sim MCD$), in the between, there are triples and the most right column corresponds to four synthesizers. One outlying example can negatively influence the relevancy of the measure. While the recording represented by the red columns has quite similar MCD between every pair, the other one has outliers and thus the resulting value has got low confidence.	32

4.2	Plot of the M_1 measure correlation with human judgments. Points are horizontally spaced and colored according to gold labels. The vertical axis shows M_1 value. The blue lines represents estimate yielded by least square fit. The top figure includes the 0-th mel cepstral coefficient, while the figure on the bottom does not. . . .	33
4.3	The process of aggregating the values obtained from mel cepstral analysis. Each recording is represented by sequence of M -sized vectors. These sequences are summed, thus the representation is flattened into one vector of length M	34
4.4	Plots of the correlation of the measure computed by model in cross-validation folds. The predicted values are on the vertical axis, the horizontal position represents the human judgments. The blue lines shows least-squares fit. Although the training data are rather small, it shows, that the model can output meaningful values. . .	35
4.5	Scatterplot showing relation between the M_1 and M_2 measures. We can see, that the measures are rather uncorrelated. However, they can still be complementary.	36
4.6	Sample alignment of two phonetic sequences	37
4.7	Frequencies of confusing respective bigrams, i.e. how many times it occurs in some confused pair. We can see, that the majority of bigrams has very low frequencies.	38
4.8	Visualization of the confusion matrix computed from the results. .	39
4.9	A boxplot showing difference of the predicted labels from human ratings.	40
4.10	Plot of the labels given by humans (vertical axis) and predicted values (horizontal axis). The vertical line represents the threshold with respect to the predictions, the horizontal dashed line illustrates true division.	41
4.11	A Receiver Operating Characteristics line of Logistic Regression classifier. The green dashed line indicates a random choice, the red point corresponds to the chosen threshold.	42
4.12	Dependency of decoder's Oracle Phone Error Rate on the depth of the n -best list we explore. The bigger part of the list is exploited, the lower is the error rate.	44
4.13	An example of clustering hypotheses of the phone recognizer with spectral clustering algorithm. The input was spoken czech word "Čokoláda".	45
4.14	An example of clustering hypotheses of the phone recognizer with kmeans clustering algorithm. The input was spoken czech word "Čokoláda".	46
4.15	An example of encoding word "moon" using BoN technique with $n = 1$. The reduced Latin alphabet was used.	47
4.16	The dependency of the Linear Regression model accuracy on the size of the data it is trained on. The red dashed line represents the performance of trivial classifier. We can see, that with sufficient data, the model outperforms the baseline.	48

4.17	Distribution of target classes in the dataset. Although the hypotheses lists have been shuffled, the distribution is biased, making the task more difficult.	49
4.18	An illustrative example of the FST combined from the <i>g2p</i> (top) and speech decoder (bottom). A part of input is chosen corresponding to the syllable "ČO" and the chosen path is highlighted. It can be seen, that the path goes through both the decoder's and <i>g2p</i> 's hypotheses.	50
4.19	An example of the preliminary result derived from combined FST. We can see, that the <i>g2p</i> transcription is rather incorrect. We highlight groups of phonemes, where the transcription is changed so it is more similar to the original. However we can see that some other phonemes were replaced incorrectly and the result is far from perfect yet. The transcriptions are given in IPA.	51

List of Tables

4.1	The table contains correlations that were observed among the raters.	31
4.2	The table contains results of linear regression model training on aggregated MFCC vectors. Correlation with human judgment and R^2 measure for respective cross-validation folds are shown.	36
4.3	Overview of results in terms of Phone Error Rate (PER) or Oracle PER, respectively. We measured the Oracle PER only on the most succesful decoder. The best results are showed in bold.	44
4.4	Test set accuracies of various models both with and without the reference transcription included. The baseline is set by trivial classifier, which assigns each example the most frequent label. Red values indicates, that the model does not outperform the baseline, bold values are the beast achieved results.	48

List of Abbreviations

Attachments