Austrian Dialect Classification Using Machine Learning

Hanna Wagner, BA

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im Juni 2019

Declaration

Hagenberg, June 23, 2019

Hanna Wagner, BA

Contents

Abstract

Different speaker dialects are one of the main problems in automatic speech recognition (ASR). A possible solution to this issue is to have a separate classifier identify the dialect of a speaker and then load an appropriate speech recognition system. Because no attempts have yet been made in classifying Austrian dialects this thesis investigates in the classification of the two dialects spoken in Vorarlberg and Vienna. The choice of the selected classes is based on a linguistic perspective. Those two dialects have in consideration of history and characteristics the most differences.

The first part of this thesis deals with theoretical explanations of machine learning in general, the various algorithms that can be used, general challenges in machine learning and feature extraction methods. Due to the lack of an existing Austrian dialect database, several sources have been used to create a new one. Following a preprocessing pipeline has been established, which shows several paths of the data preparation from the raw data to the state where the data can be used as an input for the machine learning algorithms. After implementing some states, the best path and its model was chosen for evaluation. The classification was done using convolutional neural networks (CNNs). Support vector machines (SVMs) were also implemented but could not perform as good as the CNN algorithm.

Results showed that the best accuracy could have been achieved by splitting the files in 0.4 second smaller ones, applying a median filter and extracting the features using the mel frequency cepstral coefficient (MFCC). The CNN model reached the highest accuracy of 84.20% on the test set and 77.85% on the validation set.

Kurzfassung

Unterschiedliche Sprecherdialekte sind eines der Hauptprobleme bei der automatischen Spracherkennung. Eine mögliche Lösung besteht darin, dass ein separater Klassifizierungsalgorithmus den Dialekt eines Sprechers identifiziert und dann ein entsprechendes Spracherkennungssystem verwendet. Da noch keine Versuche zur Klassifizierung österreichischer Dialekte unternommen wurden, untersucht diese Arbeit die Klassifizierung der beiden Dialekte in Vorarlberg und Wien. Die Auswahl der Klassen basiert auf sprachwissenschaftlichen Erkenntnissen. Beide Dialekte weisen in Bezug auf Geschichte und Sprachmerkmalen die größten Unterschiede auf.

Der erste Teil dieser Arbeit befasst sich mit maschinellen Lernen im Allgemeinen, den verschiedenen verwendbaren Algorithmen, den Herausforderungen beim maschinellen Lernen und den Methoden zur Extrahierung der Features. Aufgrund des Fehlens einer vorhandenen österreichischen Dialektdatenbank wurden mehrere Quellen verwendet, um eine neue zu erstellen. Folgend wurde ein Vorbearbeitungsprozess für die Daten erstellt, der verschiedene Möglichkeiten darstellt, wie die Rohdaten bearbeitet werden können, damit sie als Eingabe für die Algorithmen verwendet werden können. Nach der Implementierung einiger dieser Schritte wurde die beste Möglichkeit und deren Modell ausgewählt. Die Klassifizierung erfolgte mithilfe von Convolutional Neural Networks (CNN). Support Vector Machines (SVM) wurden ebenfalls implementiert, konnten jedoch nicht so gute Resultate wie der CNN-Algorithmus erzielen.

Die Ergebnisse zeigten, dass die beste Genauigkeit erzielt werden konnte, wenn die Dateien in 0.4 Sekunden kleinere Dateien aufgeteilt, ein Medianfilter angewendet und die Features mit Mel-Frequenz-Cepstral-Koeffizienten (MFCC) extrahiert wurden. Das CNN-Modell erreichte die höchste Genauigkeit von 84.20% auf dem Testdatensatz und 77.85% auf dem Validierungsdatensatz.

Chapter 1

Introduction

This chapter provides a fundamental introduction to the topic of Austrian dialect classification using machine learning. First, background and motivation are presented. A lot of different challenges need to be overcome, in order to implement a successful project, which is described later on. Furthermore, some state of the art approaches and the delimitation to them are proposed. Finally, the outline is stated.

1.1 Background and Motivation

Although artificial intelligence (AI) has been introduced decades ago, the interest in machine learning has set off over the past years. The topic has made it not only in computer science programs and industry conferences but also to the daily local newspapers. The same goes with neural networks, which have also been around for 50 years. However, the lack of data and computing power weakened the interest over time. The enthusiasms extended with the publication of Geoffrey Hinton in 2006. In this paper[1], he showed how to train a deep neural network, which is capable of recognizing handwritten digits with a precision of over 98%. In the mid-1980s many significant architectural advancements were also made using neural networks. Now an immense amount of data is produced every day, which is an optimal foundation for this technology. Moreover, computational power has increased drastically over the past couple of years, so problems, which seemed to be impossible then can be solved now [12].

One of the earliest applications in this field, which are highly used until today, is the spam filter. This machine learning application was introduced in the 1990s. Although this program is no self-aware robot, it improves the lives of hundreds of million of people every day. One of the many questions concerning machine learning is, where does it start and where does it end? At what stage becomes a program intelligent? Machine learning is stated as the science of programming computers so they can learn from data. In the case of the spam filter, the program can learn to flag spam given examples of spam emails and regular emails. A more recent and impressive example is known as the Google DeepMind Challenge Match. In this application, a computer program, which was developed by Google DeepMind, could beat the 18-time world champion Lee Sedol in the abstract strategy board game Go. The third and most relevant application for

[1] http://www.cs.toronto.edu/~fritz/absps/ncfast.pdf

this thesis is using machine learning for recognizing speech [5].

Automatic speech recognition (ASR) has been around for the last 25 years. Although assistants such as Apple's Siri[2] or Amazon's Alexa[3] deliver good performance, there are still limitations of possible interactions due to the lack of natural communication with computers [6]. Human speech can be affected by several factors. There are age, gender, emotional state, and speaker-to-speaker variations, that have an impact on speech. Another reason is the difficulty of systems understanding speech by non-native speakers. Some efforts have been made on classifying dialects [20], but there are still some improvements in increasing the accuracy and adding new dialects to their models. Furthermore, the knowledge of the speaker's accent is not only important for the selection of specifically trained models, but it can also be significant for the results itself. For example, when a person from Vienna and a person from Vorarlberg are asking their phones the same questions using voice input, the program has two main challenges. The first is understanding and translating the speech to text, so the computer can work with it and execute the search. The second challenge is to deliver the best result possible. The quality of the outcome can increase by knowing from which part of Austria the speaker is from.

Recent individual studies [26, 13] have likewise shown that research has focused on building accent classification systems using machine learning. However, these approaches have some gaps, which need to be filled. Although both approaches have achieved promising results, some improvements are still possible. The first one is that their data was very well structured. The words or sentences were given, which made it easier to classify the accents because every speaker said the same. Therefore using one of these models in a real-life scenario could decrease their accuracy, because different words and word-combinations will highly likely be used. Several other approaches are introduced and discussed in Chapter 4. Although a lot of studies regarding English dialects classification exist, no attempts have yet been made in classifying Austrian dialects.

1.2 Problem and Solution

This section describes the main challenges that need to be covered and overcome during the implementation of the thesis project. The first challenge deals with a lot of questions concerning data. When classifying English dialects, databases like the Speech Accent Archive[4] or the Common Voice dataset by mozilla[5] could be easily used. In contrast, the first issue is that the gathering of Austrian dialects is difficult because there are no labeled audio files or databases in general available. Several approaches have been carried out. The first idea was to gather the data from the Austrian TV Station called ORF directly. Another method is to ask several students from the University of Applied Sciences Upper Austria to record themselves answering given questions (see Appendix B). The last option is to download the audio files directly from Vorarlberg

[2]https://www.apple.com/siri/
[3]https://alexa.amazon.com/
[4]http://accent.gmu.edu/
[5]https://voice.mozilla.org/en

press conferences[6], from the City of Vienna[7] and from the Literaturradio Vorarlberg[8].

The second issue is that the audio files do not have a common word sequence, which makes it most likely harder for the algorithm to differentiate. One approach could be to interview a bunch of people asking them to read out the same text passage. This would make it a lot easier for the algorithm to differentiate between the dialects. But this approach has two main downsides. The first one is that it is immensely time-consuming to gather that many interviews, so it can be used as a good source for training and testing the model. The second contrast is, that this model would not succeed in a real-life scenario, as mentioned before because different words and word-combinations are used.

The third problem is that the volume, the quality and the length of each audio file is different. So a lot of preprocessing and data cleaning steps will be inevitable. Also in some cases, background noises can be disturbing. Moreover, it needs to be noticed that an imbalance of female and male speakers can also worsen the results. The last issue concerns the preprocessing pipeline. Because there are a lot of different steps needed, such as audio normalization, defining the window size and choosing the right algorithm, hundreds of contrasting paths are possible. Furthermore, the order of those preprocessing steps is essential and needs to be evaluated. Therefore only a few of those possible solutions can be implemented and tested. One possible approach is to choose a baseline model as a reference and implement a few paths which are highly different from each other.

The aim of this thesis is to not only find out which algorithms perform best in this scenario but also to find out by experiments which data preparation steps are most suitable for the classification of Austrian dialects using machine learning. The implementation is written in Python using the Scikit-Learn[9] library for the support vector machine classifier and the Keras[10] library for the neural network classifier. A full account of packages is listed in Appendix C.

1.3 Limitations

This thesis focuses on data preparation, data cleaning and data wrangling. This is partly motivated by the fact that preparing the audio signal for the machine learning algorithms is a sophisticated and significant task. Furthermore, the main part of these classification experiments deals with dialect classification based on spontaneous speech. This is, in contrast to given utterances of single words or given sentences, which is reasonably the more common variant of raw data, more difficult but more promising. The embodiment of dialect identification into automatic speech recognition, which leads possibly to a performance increase, is not tested. As no examples of Austrian dialect classification with defined dialect regions have been found, two dialects are presented and used here for the first time.

[6]http://presseaudio.vorarlberg.at/
[7]https://www.wien.gv.at/video/Wir-und-Wien
[8]http://www.literaturradio.at/kategorien/mundart/
[9]https://scikit-learn.org
[10]https://keras.io/

1.4 Outline

Chapter 2 introduces important definitions and provides detailed information about algorithms, the feature extracting methods and other technical concepts. In Chapter 3 essential background information and insights into the differences between the dialects of Austria are provided. Following, Chapter 4 presents various approaches to classifying dialects for different languages. In Chapter 5 the data, which is later used for the experiments, is introduced. Moreover, technical strategy and the data preparation pipeline are defined in Chapter 6. Chapter 7 explains the details of the experiments and a reflected discussion of these results. Lastly, Chapter 8 contains conclusions and reflections on the work.

Chapter 2

Technical Background

Machine learning is a relevant and interesting field and with new algorithms and frameworks developed and improved constantly, it is more relevant than ever. In order to understand the following chapters, this chapter states necessary definitions and provides detailed information about algorithms, the feature extracting methods and other technical concepts.

2.1 Introduction to Machine Learning

This section gives an introduction into the field of machine learning. The following reference gives a good and understandable definition [30]:

> "Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves."

2.1.1 Types of Machine Learning Systems

Figure 2.1 shows the relation between artificial intelligence and machine learning and what other fields are dependent. There are several types of learning algorithms:

- supervised learning,
- unsupervised learning,
- reinforcement learning,
- deep learning and neural networks.

Using *supervised learning*, an algorithm can be trained with a dataset (input) and a solution (output). At the beginning of the training, there is data with a labeled tag, which means that each element of the dataset comes with a class. The algorithm remembers that this item belongs to this special group and tries to extract some distinctive features, which should be unique for this group. When a new input without a label is being put into the algorithm, it compares the unique features of the new data with the familiar groups that are already known. The new data obtains the label in which the

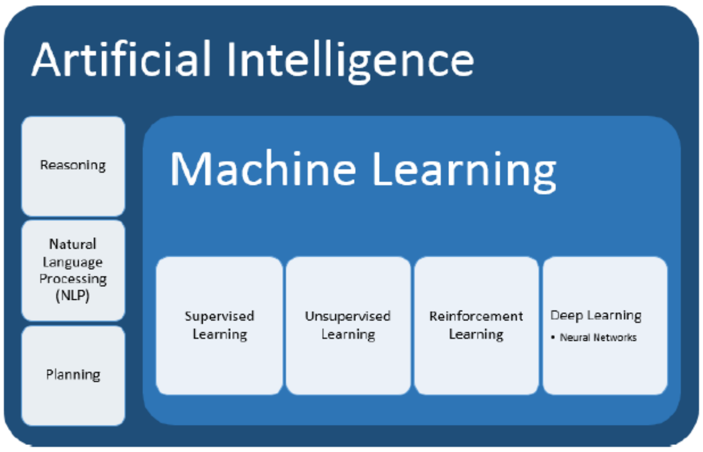


Figure 2.1: Related Fields [10].

highest accordance is. An exemplary task would be the classification of text categories or to find out if an email is spam [5].

With the use of *unsupervised learning* algorithms, there is also a training dataset (input) but does not provide a tag marker. A typical task would be to design a clustering algorithm that tries to detect e.g., the groups of similar visitors on your website. Because there are no labels involved here, the program needs to find out all the characteristics and comes up with the number of groups by itself. There is also the possibility to have hierarchical clustering, which means that you can subdivide your groups into smaller ones. A possible task for these kinds of algorithms would be the anomaly detection. That means that the program could detect unusual credit card transactions [5].

Reinforcement learning is a little bit different compared to supervised and unsupervised learning. The learning algorithm is named an agent, which discovers its environment by a trial-and-error principle. After every move, the system gets a reward or a penalty and it must learn by itself which the best strategy is to maximize the rewards. The last concepts, deep learning, and neural networks are explained in Section 2.2. In this case, the concept of supervised learning has been applied [5].

2.1.2 Classification vs Regression

Within this master thesis, the task is to distinguish between Austrian dialects. In the machine learning environment, this is called a *classification task*, which is a typical supervised learning task. An algorithm tries to create a model based on the input data, which is able to tell for every new, unseen data, to which category (i.e., Vorarlberg) it belongs. This prediction could either be 0 or 1, if there are only two possible classes or can be a floating-point number i.e., 0.9, which means that the new data belongs to 90% to this class [12].

Another typical task is called *regression*. The goal is not to predict a certain class or category, but a numeric value, such as the price of an apartment. In order to achieve this task, the data needs to consists of labels and a set of features (e.g., price, equipment, and condition of the building) [5].

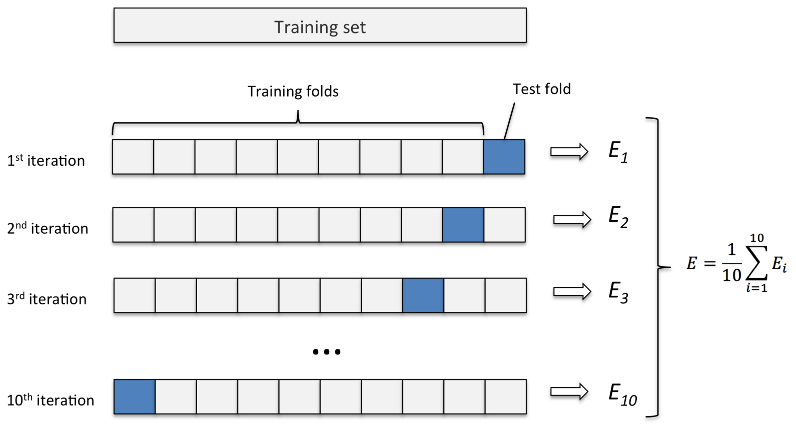


Figure 2.2: K-fold cross-validation [27].

2.1.3 Training a Machine Learning Model

After choosing one of the machine learning algorithms, which are later described in Section 2.2 and Section 2.3, the training set is now ready for its training. In the first step, the data is split into training and test. The chosen algorithm looks only at the training set and tries to figure out characteristics that divide each class from another. This knowledge is now tested against the second unseen dataset. If the algorithm has chosen good characteristics it is now able to differentiate between the classes. Often the first approach is not always the best. Therefore some improvements need to be done to the algorithm, this is called *hyperparameter tuning* [5].

One possibility is to fine-tune all the hyperparameters manually, which is time-consuming and lots of possibilities need to be explored. A better solution is, in the case of Support Vector Machines, which are described in Section 2.3, to use *grid search*. In this case, a list will be created with all the hyperparameters, that need to be tuned, and its values. The grid search runs through all combinations and delivers those with the best performance. Inside of this search *cross-validation* is used [5].

When using the cross-validation, the training data is split into multiple subsets. Each model is trained with one of the subsets and validated through the remaining ones. This technique is used to avoid needing too much training data in the validation sets. Figure 2.2 displays the split of the training set into the different training folds. By running through the comparison, a better estimate can be generated by the model's performance [27].

2.1.4 Measurement of Performance

Without any tools how to measure the performance there would be no comparison possible, how well the selected algorithms perform. The following validation possibilities can be used:

- confusion matrix,
- classification report.

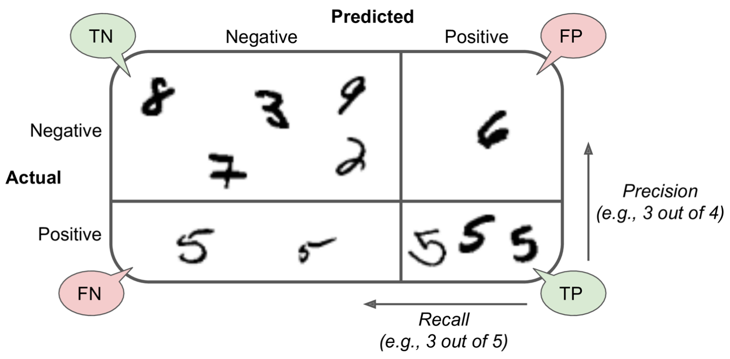


Figure 2.3: Confusion Matrix [5].

The general idea of a *confusion matrix* is to count the number of how often an instance of class A has been classified as class A, B or C. So you can see how often the classifier confused the class with another class. For this matrix, you need the test data as well as the target names. Figure 2.3 displays an illustrated confusion matrix, where hand-written digits are classified. Two different methods exist on how to calculate the success of this result. The first option is to calculate the precision [5] as

$$\text{precision} = \frac{TP}{TP + FP}. \tag{2.1}$$

The precision is a result of dividing the *true positives* (TP) by the sum of the TP and the *false positive* (FP). In other words, it is the accuracy of positive predictions. In this example, three out of four were classified correctly. The second option is to calculate the recall [5] as

$$\text{recall} = \frac{TP}{TP + FN}. \tag{2.2}$$

The recall is a result of dividing the TP by the sum of the TP and the *false negatives* (FN). It is also called sensitivity or true positive rate (TPR) and is the ratio of positive instances that are correctly detected. Using the recall as a measurement, three out of five digits were successfully recognized [5].

The *classification report* gives you a detailed view of the precision, recall, F1-score and support for each class. The F1-score, also known as balanced F-score or F-measure, can be interpreted as a weighted average of the precision and recall. The last parameter, the support, indicates the number of occurrences of each class. Table 2.1 displays an example of this classification report showing the precision etc. from 20 different classes and also the overall average [5].

2.1.5 Sequential Decision Making

Usually, the whole data is processed before the result can be calculated. This approach is called *dynamic decision making. Sequential decision making* on the other hand describes a situation in which successive observations of the process are made before a final decision is calculated. This results in being more concerned with controlling the system over time [4].

Category	Precision	Recall	F1-score	Support
alt.atheism	0.82	0.76	0.79	319
comp.graphics	0.78	0.80	0.79	389
comp.os.ms-windows.misc	0.77	0.76	0.77	394
comp.sys.ibm.pc.hardware	0.74	0.76	0.75	392
comp.sys.mac.hardware	0.85	0.85	0.85	385
comp.windows.x	0.87	0.80	0.83	395
misc.forsale	0.83	0.89	0.86	390
rec.autos	0.91	0.90	0.90	396
rec.motorcycles	0.95	0.96	0.95	398
rec.sport.baseball	0.91	0.96	0.94	397
rec.sport.hockey	0.96	0.98	0.97	399
sci.crypt	0.92	0.95	0.94	396
sci.electronics	0.86	0.75	0.80	393
sci.med	0.89	0.88	0.89	396
sci.space	0.87	0.94	0.90	394
soc.religion.christian	0.83	0.93	0.88	398
talk.politics.guns	0.76	0.93	0.83	364
talk.politics.mideast	0.96	0.92	0.94	376
talk.politics.misc	0.83	0.65	0.73	310
talk.religion.misc	0.78	0.59	0.67	251
avg / total	0.86	0.86	0.85	7532

Table 2.1: Example of a classification report.

Formally expressed this problem can take observations X1, X2, X3 and so on, one at a time. After each observation, a decision can be made which leads to the termination of the process and calculates the final result. The algorithm could also decide to continue taking more observations into consideration. This procedure, when to decide that the observations should be terminated is called the *stopping rule*. The goal is to find a rule that optimizes the decision in terms of minimizing losses or maximizing gains. In this case, the best stopping rule would look at as little as possible new instances, but as much as necessary items, in order to achieve a fast and good result [4].

2.1.6 Challenges in Machine Learning

Like in every field there are some challenges that need to be overcome in order to generate a satisfying output [5]:

- quantity of training data,
- nonrepresentative training data,
- poor-quality data,
- irrelevant features,
- overfitting,
- underfitting.

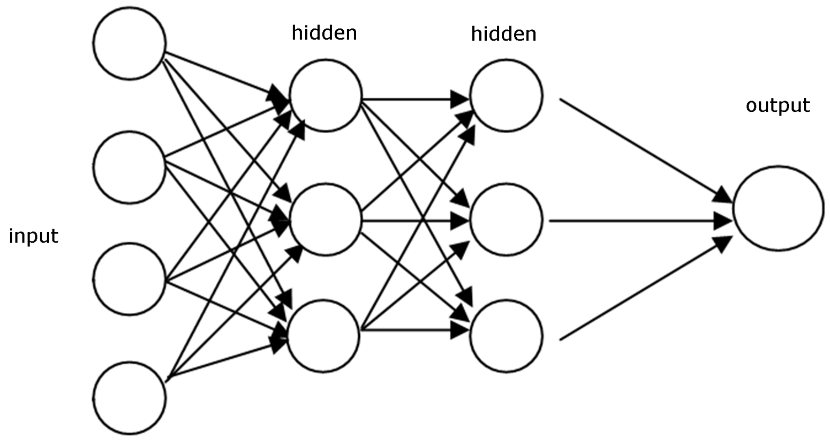


Figure 2.4: Feed-forward and feedback networks [12].

One of the hardest parts of machine learning is the acquisition of huge amounts of training data. Unfortunately even a very simple problem would need hundreds or better thousands of examples with which the model can be trained in order to make a good prediction. For more complex programs even millions of examples would be necessary. On top of that, it is also important that the input data is of high quality and represents the scenario as best as possible. The last two points are dealing with the problems of over- and underfitting. Overfitting or overgeneralizing principally means the generalization from one small example to something for a larger group. This means that the model or program only works with the training data, but not for any new data, so it is not suitable for bigger unknown data. The opposite would be underfitting. When this problem occurs the model is too elementary and has not extracted the features properly, so it did not learn the underlying structure of the data well enough [5].

2.2 Neural Networks

Neural networks are systems or approaches inspired by the biological neural networks that constitute animal brains. Figure 2.4 shows the layers in such a system:

- input layer,
- hidden layer,
- output layer.

The input layer is the first layer and introduces the initial data into the system. In the hidden layers, which are in contrast to the first layer multiple ones, artificial neurons take in a set of weighted inputs and produce an output through an activation function. The last layer is the output layer, which produces given outputs for the program [12].

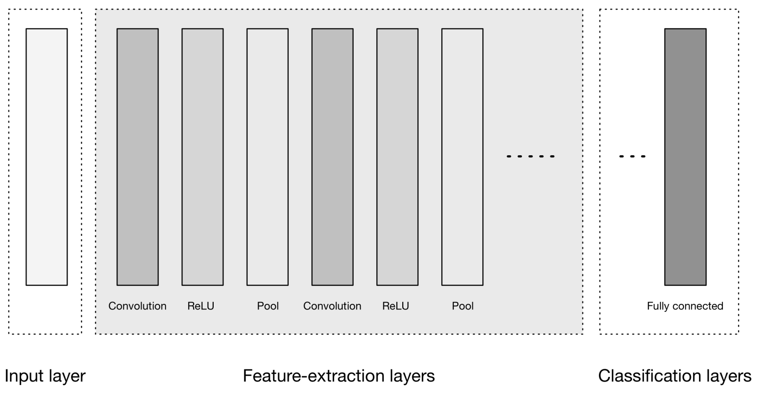


Figure 2.5: Basic CNN architecture [12].

2.2.1 Convolutional Neural Network

The aim of the *convolutional neural network* (CNN) is to learn features via convolutions. One of their major applications is image classification. CNNs are competent at extracting position invariant features from the data. Similar to neural networks, this architecture also transforms the input by connecting layers into class predictions (output layer). Different variations of the architecture exist, but all of them are based on the pattern shown in Figure 2.5. The first group accepts three-dimensional input, the height and width from the picture and the RGB color channel as a third option. The second group, the feature-extraction layers, consist of a repeating pattern [12]:

- convolution layer,
- pooling layer.

These two layers find a set of different features and calculate higher-order features. The last layers, which are called *classification layers*, have one or more fully connected layers. They get the higher-order features as an input and produce the class probabilities [12].

2.3 Support Vector Machines

Although these algorithms were invented by Vladimir N. Vapnik and Alexey Ya. Chervonenkis in the 1970s, they are still powerful machine learning models according to Aurélien Géron [5]. On top of that, he even says that it is one of the most popular models in machine learning and very well suited for classification.

Figure 2.6 shows the difference between a linear classification algorithm on the left side and a *support vector machines* (SVMs) on the right side. The colored lines in the left picture are showing the decision boundaries. One of them (the green one) does not work at all. The red and the purple can clearly separate the two classes (the yellow circles from the blue cubes). This choice will only perform well on this training data, but will probably not perform as well on new instances as possible. In comparison, the decision boundary of the SVM classifier, shown on the right, separates the two classes and tries to stay as far away from the closest training instance. Because it looks like one the dotted lines are also called a *street*.

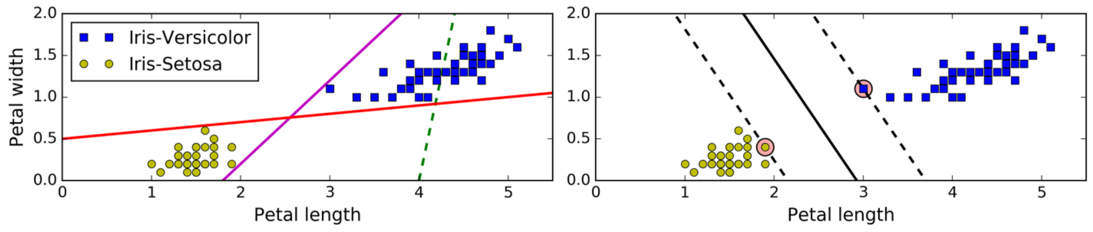


Figure 2.6: Large margin classification [5].

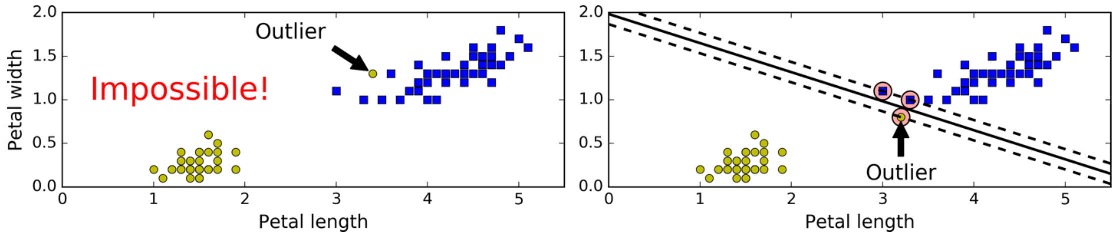


Figure 2.7: Hard margin sensitivity to outliers [5].

Naturally, not all datasets can be 100% separable. If we would assume that all instances being off the street and at the same time on the right side, this is called *hard margin classification*. Not only can some datasets not be separated, but this classification is also very sensitive to outliers, which is shown in Figure 2.7. To avoid this problem, the solution is defined as following [5]:

> "The objective is to find a good balance between keeping the street as large as possible and limiting the margin violations (i.e., instances that end up in the middle of the street or even on the wrong side)."

Figure 2.8 shows the attempt to balance the margin violations. On the right side, fewer violations have been made, by keeping the street small. The right-hand displays a wider street at the cost of having more violations. In all sorts of machine learning algorithms implementing polynomial features are simple and can work great, but with support vector machines high polynomial functions are creating an immense number of features resulting in a too slow model. Using the mathematical technique called *kernel trick* makes it possible to get the desired result from many polynomial features, without adding them [5].

2.4 Mel Frequency Cepstral Coefficient

In contrast to text classification, in which words are counted and vectorized, extracting features from audio data is a bit more complicated. One way is to use the *mel frequency cepstral coefficient* (MFCC), which has been introduced in the 1980s by Davis and Mermelstein [2] and has been one of the most popular approaches ever since. James Lyons has divided the process in six steps [25]:

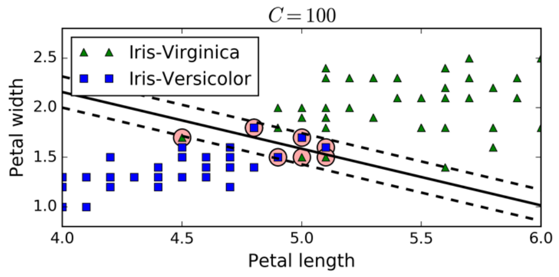


Figure 2.8: Fewer margin violations versus large margin [5].

1. Split the signal into short sections.
2. Calculate the peridogram estimate of the power spectrum for each section.
3. Apply the mel filter bank and sum the energy in each filter.
4. Take the logarithm of all filter bank energies.
5. Apply the *discrete cosine transform* (DCT) to the energies.
6. Only keep the coefficients 2–13 and remove the rest.

For simplification, we assume that for a certain short period of time, the signal from our audio data is not changing. Therefore the signal is divided into short frames (20-40ms) to get a reliable spectral estimate. If the period would be shorter, too little information could be extracted. If the time would be much longer, the assumption that the signal does not change over time could not be satisfied. Next, the power spectrum of each frame is calculated to be able to identify which frequencies are present in each specific frame.

Following the mel filter banks are used because the current signal still contains too much information. In detail, the human cochlea cannot differentiate between two frequencies if they are too close, which gets worse the higher the frequencies are. Using the filter banks the signal gets divided into multiple chunks, each representing a specific frequency range. The higher the frequency gets, the bigger the filter bank range is, which is displayed in Figure 2.9. With these filter banks, an estimate of the energy, which occurs at each spot, can be retrieved. The calculation of the filter banks can be retrieved by applying triangular filters on a Mel-scale to the power spectrum to extract frequency bands. The conversion between Hertz (f) and Mel (m) is the following [25],

$$f = 700(10^{m/2595} - 1), \tag{2.3}$$

$$m = 2595 \log_{10}(1 + \frac{f}{700}). \tag{2.4}$$

Next up is taking the logarithm of the filter bank energies. This compression operation makes features match more closely to what humans actually hear. In this step, the features are compressed using the logarithm. The cepstral mean subtraction, which is a channel normalization technique, can be applied afterward. Because the filter banks are all overlapping, the filter bank energies are having a strong correlation. Using the DCT, this correlation will be removed. Lastly only 12 out of the 25 DCT coefficients are used

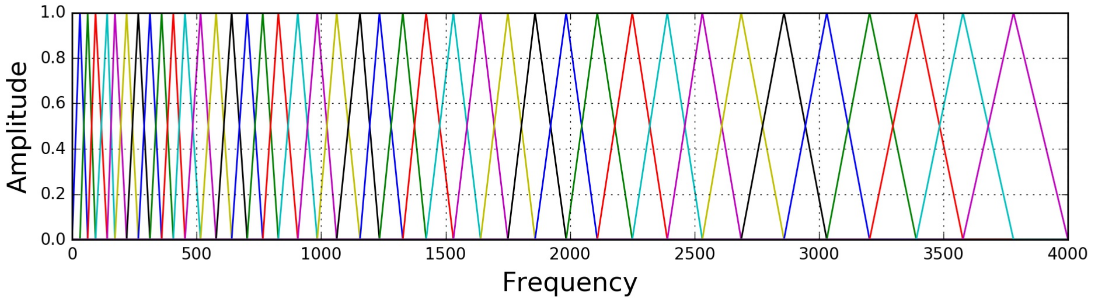


Figure 2.9: Filter bank on a Mel-scale [23].

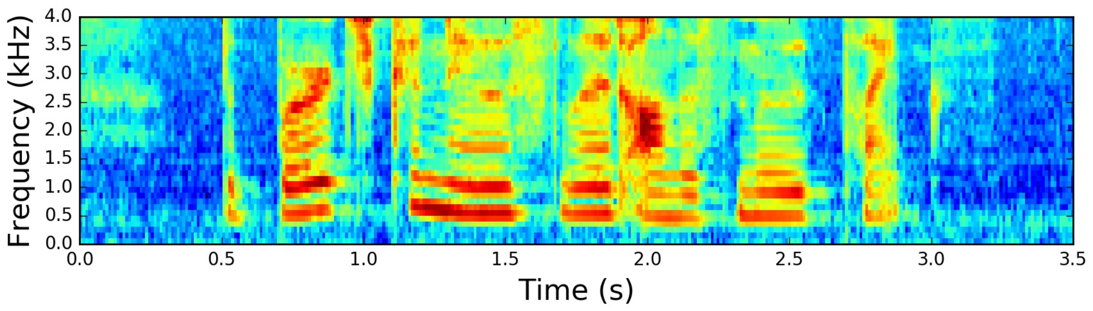


Figure 2.10: Spectrogram of the signal [23].

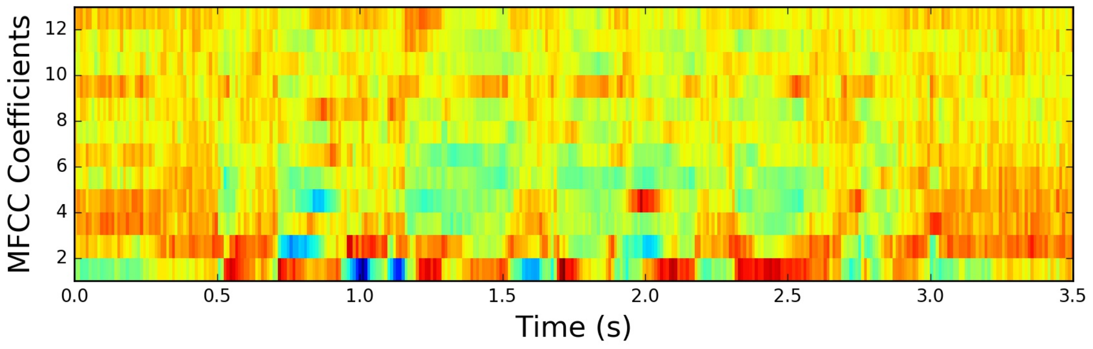


Figure 2.11: MFCCs [23].

further. This is because in the past better results have been achieved by dropping the other energies. The results of this step are shown in Figure 2.10.

The result of the whole process is displayed in Figure 2.11, showing the 12 features, which are called the mel frequency cepstral coefficients [25]. This picture shows a 3.5-second frame. The dark blue spots representing little energy, in contrast to the dark red, which are representing a massive amount of energy in this section. Some steps along this path were motivated by the nature of the speech signal and the human perception of such signals. The last steps, which are necessary to compute the MFCC filter out of the

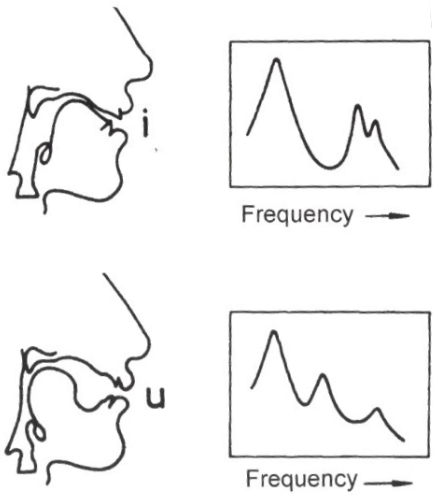


Figure 2.12: Vocal tract for the vowels i and u and the according frequency spectra [14].

filter banks, were motivated due to the limitation of some machine learning algorithms. Fayek [23] questions that if, with the usage of deep learning in speech systems, MFCCs are still the optimal choice, given that deep neural networks are less susceptible to highly correlated input and therefore the DCT is no longer a necessary step.

2.5 Formants

Similar to the MFCC, which have been described in Section 2.4, *formants* can also be used in order to extract features from audio data. In acoustics and phonetics, formants refer to the concentration of acoustic energy in a fixed (invariable) frequency range (Hz), independent of the frequency of the generated signal. These frequency ranges are amplified or attenuated over others, leaving the formants as energy spikes. Each human vowel outputs unique frequency spectra, which is illustrated in Figure 2.12. Generally speaking, a formant is one of the characteristic partials of a sound and multiple formants form a whole vowel [14].

Given the first two formants, each vowel can be identified. Theoretically, there are more than four formants, but even the 5th can often not be measured. Furthermore, those additional formants do not contribute to any new features. Therefore only the first four are commonly used. The first two formants F1 and F2 are important for the intelligibility of the vowels. Their position characterizes the spoken vowel, the third and fourth formants F3 and F4 are no longer essential for understanding speech. They characterize rather the anatomy of the speaker and its articulation characteristics as well as the timbre of his language and vary depending on the speaker [1].

Using the software Praat[1], formants can be extracted for every 0.01 seconds from an audio file. Figure 2.13 shows the calculated formants from a 0.5-second audio snippet, which are displayed as red dots throughout the timeline. The red dotted horizontal line is used as a reference for the Hz range. In this case, the range for the formants is

[1]http://www.fon.hum.uva.nl/praat/

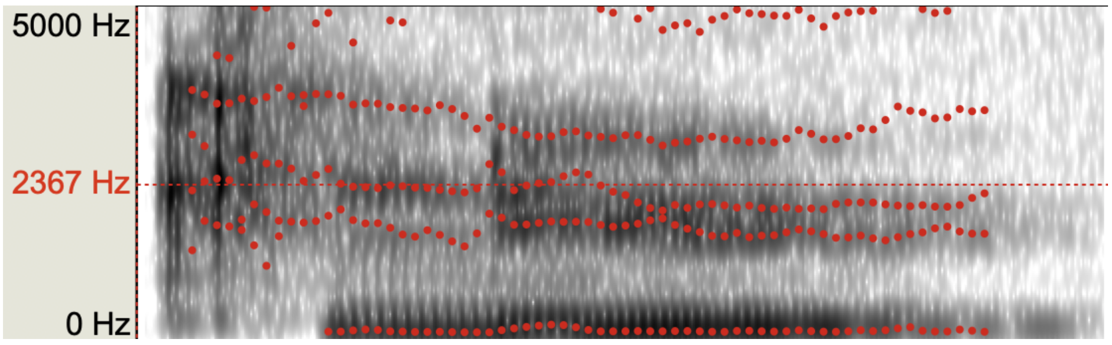


Figure 2.13: Extracted formants using the Praat software.

between 0 Hz and 5000 Hz but could be increased by manual settings. In this scenario, the value is the default and only used for exemplary matters.

Because the third and the fourth formants contain the characteristic of the sound, the assumption could be made, that those two vowel formants can be used to classify the dialect of the speaker. Furthermore, C. Themistocleous [16] provided a classification model of two Greek dialects using information from the formants dynamics of F1, F2, F3, F4, and vowel duration yielding good results.

2.6 Noise Reduction in Sound

ASR or in this scenario *dialect detection* describes a technology in which machines are able to derive human spoken words. Ideally, the speech signals, which are used for training, are recorded in a clean and noise-free environment. But in a real setting, speech gets influenced by the microphone, the distance, the room, and the background noises, which leads to worse quality of the audio. Noise can be defined as the following [29]:

> "Noise refers to any external and unwanted information that interferes with a transmission signal. Noise can diminish transmission strength and disturb overall communication efficiency. In communications, noise can be created by radio waves, power lines, lightning and bad connections."

For better quality and a better accuracy, the goal is to reduce the noise as much as possible without losing too many details of the signal. In this sections two different filters are described:

- median filter,
- kalman filter.

2.6.1 Median filter

One solution is to use a *median filter* for the removal of the noise from the signal. This filter is a non-linear filter. The first step is to sort the item to be changed and its neighbors, according to their size. This filter has its popularity due to the simplicity of its calculation and effectiveness [8].

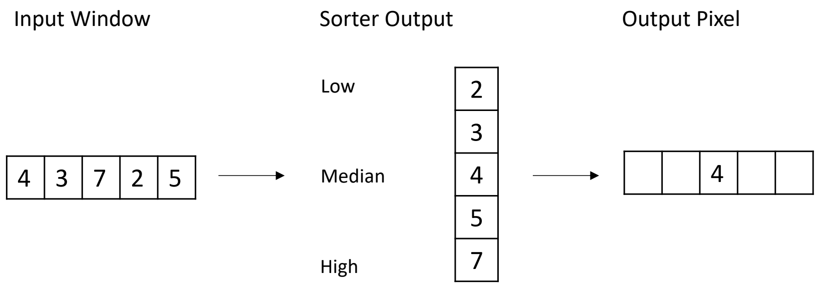


Figure 2.14: Steps of the median filter.

In the example in Figure 2.14 the values are arranged from 2 to 7. The value in the middle (the median), in this case, 7, is representative and will be used to replace the values, which need to be filtered. With this technique, outliners can be removed. So the main concept is to go through the data entry by entry and replace each entry with the median of the neighboring entries. In this example, the value 7 has been replaced by 4. The closest two neighbors have been used for calculations, but more values could be used as an input.

Figure 2.15 shows the result of using this filter on a speech signal. The first 0.6 seconds and the last 0.6 seconds do not contain any human voice, but background sounds. The word is spoken between 0.6 and 1.5 seconds. The most obvious results have been achieved between 0.5 and 0.6 seconds. In this time period, the noise has been reduced drastically. Therefore it can be assumed that the noise during the spoken word has also reduced, even though it can not be seen as clearly on the MFCC pictures.

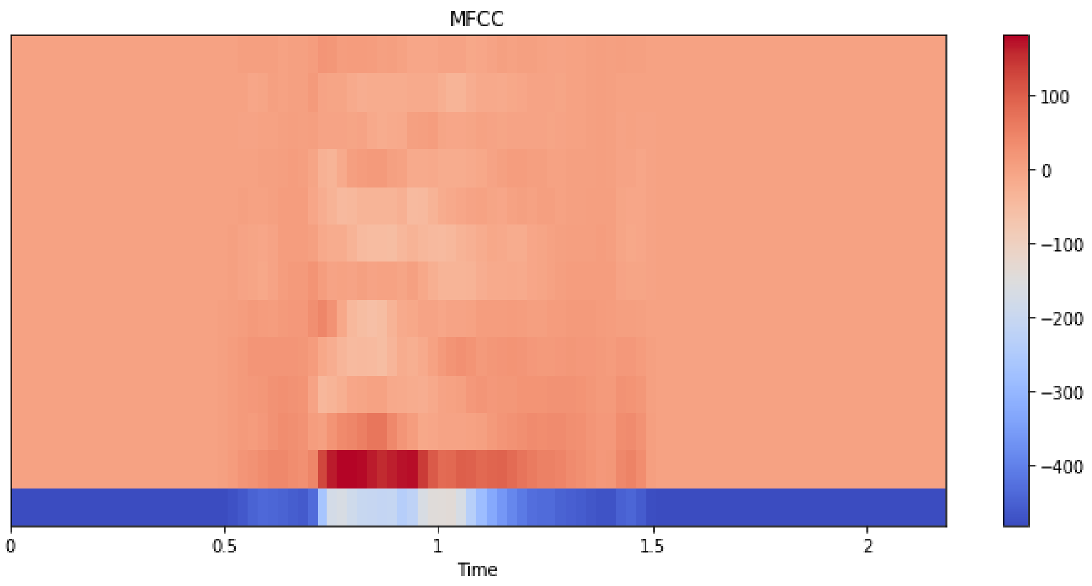
2.6.2 Kalman Filter

Another possibility to reduce the noise in sound is to use a *Kalman filter*. This filter was invented in the 1960s by Rudolf E. Kalman. Originally it was used for discrete-time linear systems. The huge advantage is the iterative aspect, which is especially useful for real-time applications. The goal is to predict the values as well as possible over time. At first, an initial estimate has been chosen. Then the first measured value will be looked at. This measurement contains some errors. Based on the measurement an estimate for the true value will be calculated. These steps will be repeated in an iterative way. So first the value will be measured, then the estimate will be produced [9].

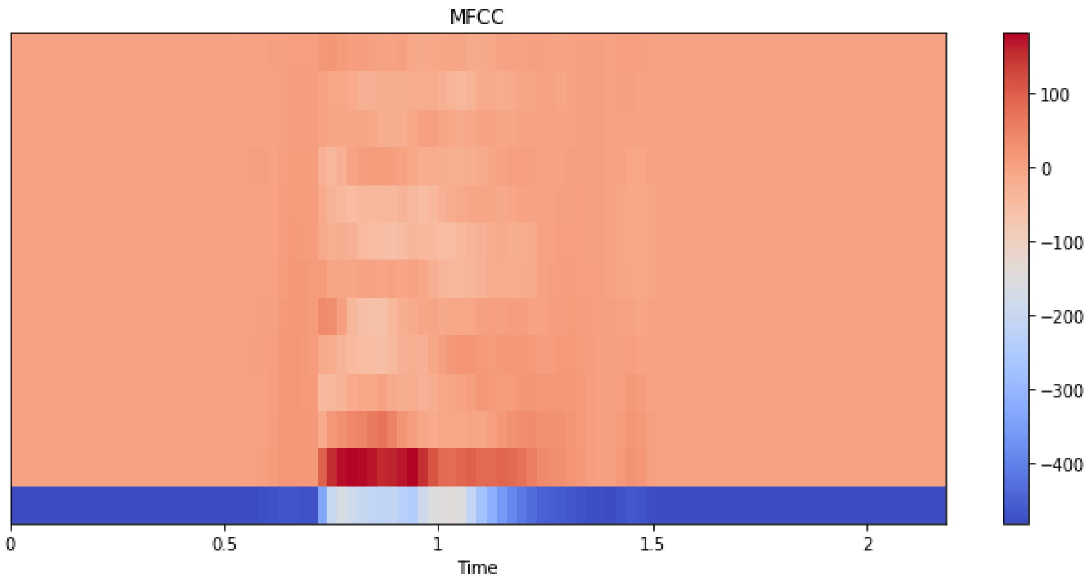
Nair et al. [11] propose that using this filter in combination with MFCC provides considerable results in speech recognition. The approach proposes that the measured values, which are not 100% reliable, contain background noises, which needs to be filtered. The estimate values are describing the desired signal without the noise.

2.7 Audio Normalization

Audio normalization is an important step in data cleaning. Because different kinds of audio files with various background noises and volume are used, the aim is to match



(a)



(b)

Figure 2.15: Effects of median filter on audio. *Original audio* (a), *Audio after using median filter* (b).

them all as close as possible to the same volume. Therefore three different methods can be used:

- peak normalization,
- root mean square(RMS)-based normalization,
- European broadcasting union(EBU) R128 standard.

2.7.1 Peak Normalization

Given the assumption that all signals lie between -1 and $+1$, the goal within *peak normalization* is to convert the maximum value to either one of those extremes. The maximum magnitude in this example is 1. A signal's magnitude can be normalized to 1 *full scale (FS)*. To create a normalized output is [18]

$$\text{out} = \frac{\text{in}}{\text{max}(\text{abs}(\text{in}))}. \tag{2.5}$$

The signal's peak magnitude will be normalized to the value of 1 FS. In other words, peak normalization results in making the audio as loud as possible. This method is based on the instantaneous level of a signal [18].

2.7.2 RMS-based Normalization

In comparisont to the peak normalization, where the basis value is the highest value, using the *RMS-based normalization* the average signal strength across the signal is being used. The process can be described as the following [19],

$$x_{\text{rms}} = \sqrt{\frac{1}{n}(x_1^2 + x_2^2 + ... + x_n^2)}. \tag{2.6}$$

In this case, the process does not simply measure the arithmetic mean of the signal. If the arithmetic mean of the sine wave was taken, the values would cancel each other out. Therefore, for the calculation of the value, it does not matter if the signal has positive or negative values. The average is calculated by squaring each value (resulting in only positive numbers), then the signal average can be calculated afterward. For implementing the RMS-based normalization the first step is to convert the RMS amplitude on the decibel (dB) scale to the linear scale. The process can be described as the following [19]:

> "In this case, we will multiply a scaling factor, a, by the sample values in our signal to change the amplitude such that the result has the desired RMS level, R."

Following, the whole signal will be normalized according to the previously calculated value.

2.7.3 EBU R128 Standard

As the first two methods, this standard is also using dB as a reference unit. The *EBU R128 standard*, which contains a *loudness normalization and permitted maximum level of audio signals* recommendation, uses a subjective loudness level measured in $LUFS$ (Loudness Unit Full Scale). This standard suggests using the average loudness for the normalization of audio signals. Furthermore, the standard implies that peak normalization has led to considerable loudness differences between programs and between broadcast channels. The audio files should be normalized to a target level -23.0 $LUFS$. A permitted deviation from this level should not exceed ± 0.5 $LUFS$ [17].

Chapter 3

Dialects in Austria

This chapter provides background information and insights into the differences between the dialects of Austria. In this case, the federal states Vienna and Vorarlberg are especially interesting. Generally speaking, Austrian German is a variety of Standard German, with differences in grammar, vocabulary, and pronunciation.

In this context, the distinction between accent and dialect is necessary. Hasa [24] defines that accent is a way of pronouncing words that appear among people in a certain region or country, whereas dialect is a variety of a language spoken in a particular geographical area or by a specific group of people. For example, a person who is born and raised in Vienna speaks Austrian German, more specific, the Austrian dialect called Viennese, which will be elaborated later. Let us assume that the same person can also speak English. In contrast to the Austrian dialect, this person then speaks English with a German or Austrian accent. So native speaker can probably guess where you are from based on how you pronounce specific words. Figure 3.1 displays that a lot of different influences have characterized the Austrian language. Now following dialects exist:

- West-Central Bavarian,
- East-Central Bavarian,
- Viennese,
- South-central Bavarian,
- Southern Bavarian,
- Slovenian,
- Swabian,
- High Alemannic,
- Highest Alemannic,
- Alemannic influenced Southern Bavarian.

In contrast to Upper Austria and Lower Austria, which share some characteristics (East-Central Bavarian), Vienna has almost its own characteristic language (Viennese). Also, Vorarlberg does not share much of its specifics with other federal states such as High Alemannic and Highest Alemannic. In contrast to all other federal states, Vorarlberg is the only one that does not have any Bavarian influences. Therefore the assumption can be made, that Vienna and Vorarlberg should be differentiated quite well, not only

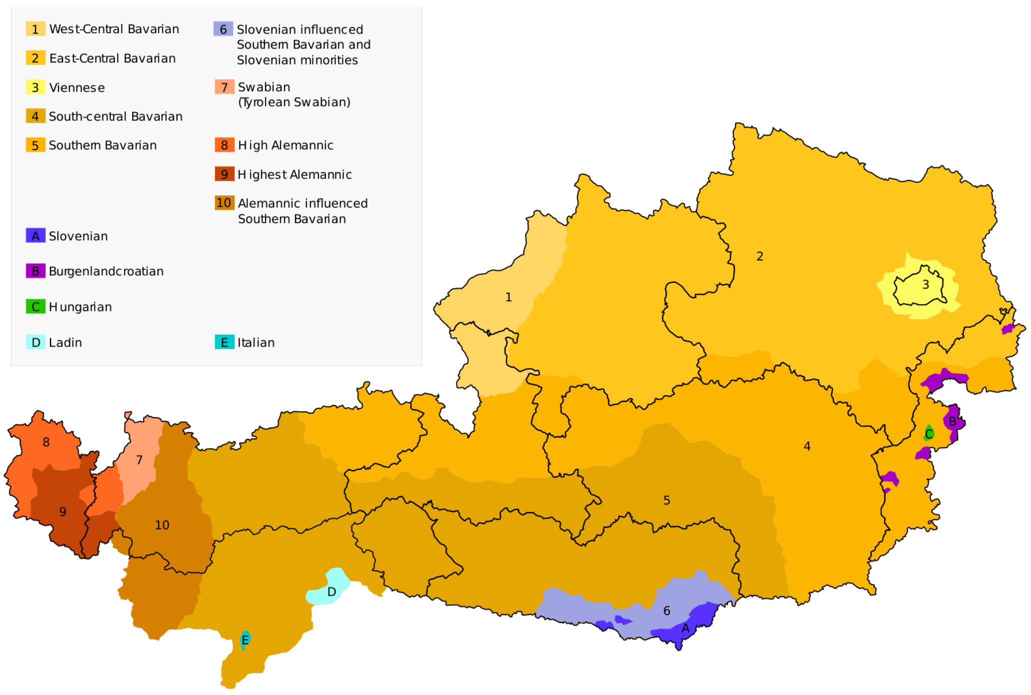


Figure 3.1: Austrian dialects [28].

because of the geographical distance but also due to the language features and influences mentioned before.

3.1 History and Characteristics of Dialects in Austria

The Austrian dialects have historically developed from Middle High German. Even in the High Middle Ages, there were distinct pronounced dialects in Austria. These dialects could be found in the so-called *dialect seals*, which formation was related to the secular and ecclesiastical dominions. The Middle Bavarian Language includes the dialects of Vienna, Lower Austria, Burgenland, Upper Austria, most of Salzburg and a small part of Styria. The characteristic features of these Austrian dialects of the Danube countries and plains are [22]:

- consonant-weakening,
- vocalization of the l and r,
- disappearance of side-tone vowels,
- preservation of ancient usury sounds.

In the case of consonant-weakening are fortis consonants replaced with its lenis consonants e.g, “p” changes to “b”, “t” to “d” and so on. Vocalization is the change of pronunciation, where a consonant becomes a vowel. In this instance, the word “lernen” is pronounced as “leanen”, so the “r” becomes an “a”. The third example deals with the disappearance of side-tone vowels. Here the vowels disappear, e.g., the prefix “-ge” and the “-e” at the end. The word “gelernt” is pronounced as “glernt”. In the case of usury sounds, pronunciation is facilitated by changing the syllable structure. This is done in this case by syllable-forming insertion of a vowel. The Austrian dialects, which belong to Southern Bavaria, are spoken in the mountainous regions in Styria, Carinthia, Tyrol, in parts of Salzburg and in the south of Burgenland. Those dialects are closer to the written language than the Middle Bavarian influenced dialects [22].

The dialects in Burgenland are regarded as relatively antiquated and singing. Carinthian is also considered to be archaic, because of its melodious and smooth sound, which is contingent by Slovenes influences. One trait of the Carinthian Dialect is the diminutive “-le” and the filler word “lai”, which means “only”. In contrast, the Lower Austrian language is already a modern type of dialect, especially in the areas around Vienna. With the exception of the eastern part, the idiom in Upper Austria is traditional rural dialects. The Salzburg Language is a mountain dialect and manifests bizarre forms in the Flachgau region. In Styria, a distinction is made between the Highland or Upper Styria and Middle Styria. In the northeast, its territory extends into Lower Austria. The most conservative language is spoken in Tyrol, with noticeable features like [22]:

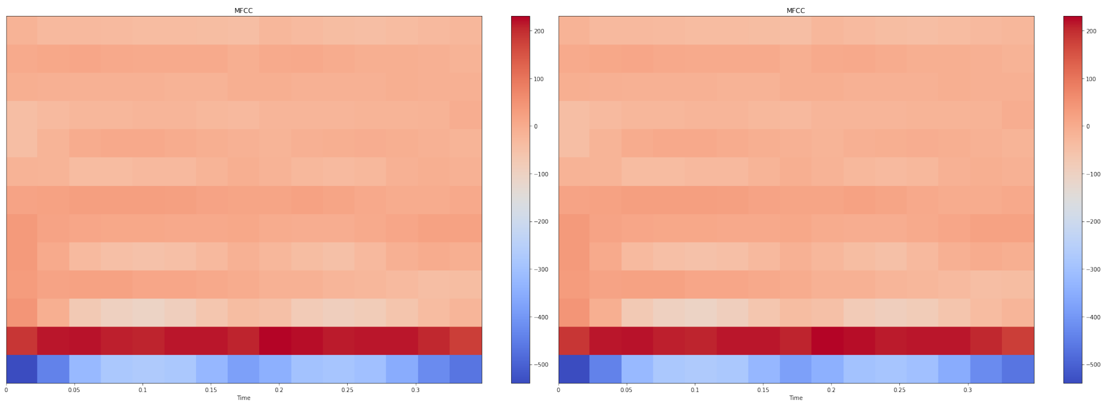
- “sch”-like pronunciation of the “s”,
- the affirmed “k” as “kch”.

Even more conservative are the numerous language island dialects emanating from Austria in the Middle Ages. The Old Viennese, which was spoken at the court of Maria Theresia, has changed noticeably since then and has long been retreating. New forms have developed like the new Viennese since 1918 and the young Viennese since 1945. By contrast, a colloquial and linguistic language developed between the old dialect and the high-level language. From Vienna, this language influences large parts of Austria, especially the provincial and provincial towns, and increasingly shapes the rural dialects. Vorarlbergian is a collective name for in the Austrian province of Vorarlberg widespread dialects of the language group of Alemannic dialects. Within the country, there is a wealth of peculiarities and in-house developments [22].

3.2 Comparison of Words

This section displays the differences between the pronunciation from one man from Vorarlberg and one man from Vienna. All recordings have been made using the same microphone and were taped in the same room. Therefore several words have been recorded and the MFCC representation has been performed:

- Mama,
- Gabel,
- Samstag,
- Herbst,
- Knie,
- Stein,
- Freitag,



(a) (b)

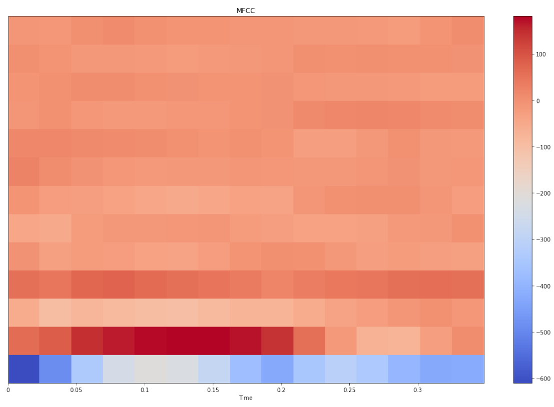
Figure 3.2: Similar pronunciation of the word Mama. *Vorarlberg Mama* (a), *Vienna Mama* (b).

- Nebel,
- Bock.

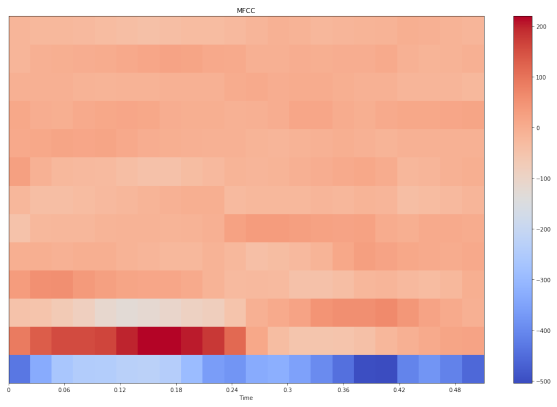
The representation of these words can be found in Appendix A. Those words have been chosen, because some of them are pronounced certainly different and result presumably in completely different MFCC representations. The other half are pronounced almost identical and result most likely in a similar image.

Figure 3.3 represents three examples in which the articulation of the words is different and presumably the representation can reflect this difference. The s in the word "Herbst" is pronounced as an "sch" in the Vorarlberg example, in contrast to the Viennese example, were it remains a "s". Therefore the upper lines have a much higher energy dense in the Vorarlberg picture, than in the Viennese picture. In the second illustration, the word "Knie" is spoken as it is written in Vienna. In Vorarlberg, on the other hand, the "ie" sounds more like an "ü". In the last word, "Stein", the second part has been pronounced something close to "-ua" and not "-ein" in the Vorarlberg dialect. All those differences appear on various levels and forms in the MFCC representation.

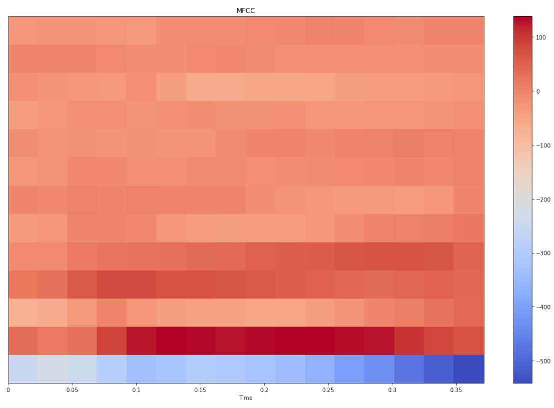
Figure 3.2 displays the examples in which the exact opposite has occurred. It shows the contrast between the spoken words "Mama" and "Gabel". In both dialects, those words have similar pronunciation. Therefore the graphical representation of the signal is also almost identical. There are some minor differences due to the fact that two different people have spoken those words, but it has nothing to do with their origin.



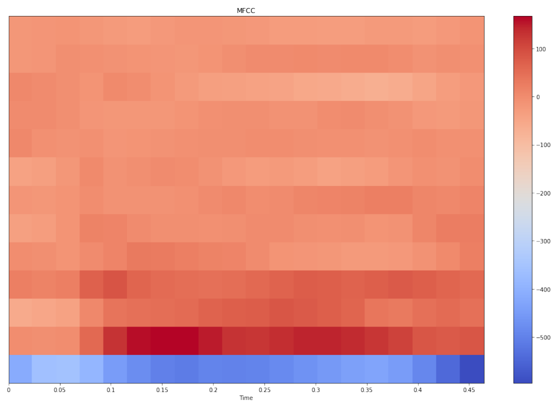
(a)



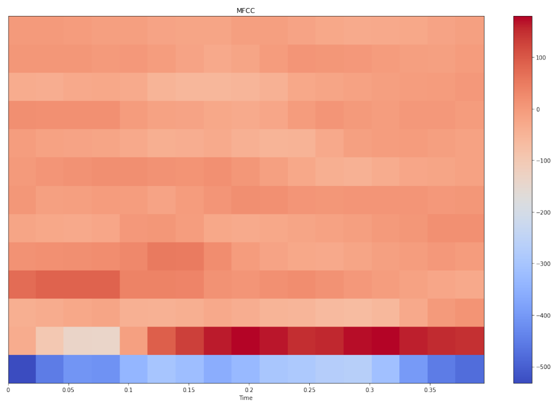
(b)



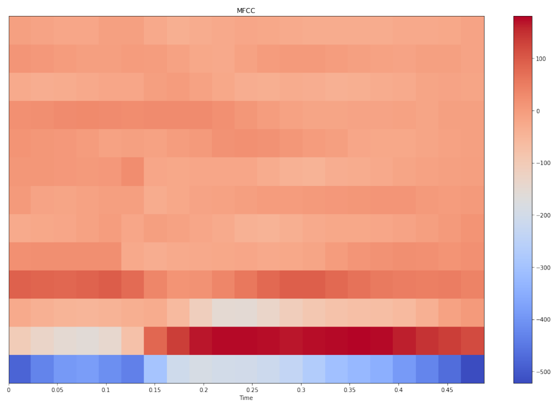
(c)



(d)



(e)



(f)

Figure 3.3: Different pronunciation of words. *Vorarlberg Herbst* (a), *Vienna Herbst* (b), *Vorarlberg Knie* (c), *Vienna Knie* (d), *Vorarlberg Stein* (e), *Vienna Stein* (f).

Chapter 4

State of the Art

This chapter will present various approaches to classifying dialects for different languages. The methods will be analyzed with regard to input data, feature extraction methods, data preparation, and implemented algorithms. The following sections are distinguished by the type of input data used. In the first part, all approaches have the same list of words or sentences for each speaker, whereas in the second part free speech or continuous speech has been used as an input.

4.1 Scripted List of Words

Sheng and Edmund [26] claim that gender and accent are still an issue in ASR systems. Therefore they built an accent classification machine learning model, which distinguishes the English from Chinese, English, and Korean as native language speakers. This model could be used as an input for an ASR pipeline. As an input, they used an utterance of a word by each speaker. The data used is called Wildcat Corpus of Native and Foreign-Accented English [1], which contains a scripted list of words and the according audio files. Using a peak detection library Sheng and Edmund were able to divide each audio file into multiple segments each containing one word. After several experiments, they found out that a window length of 0.18 seconds, where the energy density was more than 4.8% of the average energy density of the entire audio signal in the file, was the optimal value. A MFCC filter was used in order to extract the features from the signal. Following the normalization of the samples by subtracting the mean and dividing by the standard deviation has been carried out. The result of the preprocessing is a multidimensional tensor. Table 4.1 shows the result of the used machine learning methods. The first two methods were implemented as a baseline performance because the assumption was made that the neural networks will outperform the traditional machine learning methods. Furthermore, they discussed that those samples, which have been misclassified, had noticeable background noises. Those parts were incorrectly extracted because they were loud enough to be detected by the extraction script.

Another approach has been introduced by Rizwan and Anderson [13]. The algorithm was built using the TIMIT [2] dataset, which was developed by Texas Instruments (TI)

[1] http://groups.linguistics.northwestern.edu/speech_comm_group/wildcat/

[2] https://catalog.ldc.upenn.edu/LDC93S1

Model	Test Accuracy
Gradient Boosting	69.1
Random Forrest	69.1
MLP	80.0
CNN	88.0

Table 4.1: Results achieved by Sheng and Edmund [26].

and Massachusetts Institute of Technology (MIT). The data is particularly interesting because it contains various dialects from the United States such as New York City, New England and Northern. In this, two sentences by each speaker saying the same sentence have been recorded. Rizwan and Anderson implemented a weighted accent classification using multiple words. As an input, multiple words per speaker were used. Following this, the samples were normalized between -1 and 1 and 12 MFCCs were extracted and normalized. Furthermore, a Hamming window and a triangular filter bank for the MFCCs were used. After applying the preprocessing steps, 21 *extreme learning machines* (ELMs) were used to distinguish between two accents. They claim that ELMs are highly suitable for accent classification because they can be quickly trained. Additionally, they also provide a good generalization capability for small amounts of training data. Their experiments were conducted in seven classes. As a comparison, Rizwan and Anderson also implemented support vector machines as classifiers. The differences are that [13]:

> "Both ELMs and SVMs converge to a single global optimum solution. ELMs optimize sum of squared errors, while SVMs contruct a hyperplane that maximizes the separation between the data classes."

As they have increased the number of words to five for every speaker, the weighted accent classification algorithm using ELMs and SVMs results in an accuracy of 77.88% and 60.58%, which is displayed in Figure 4.1. Both introduced approaches have achieved promising results, but there are still some improvements possible. First, the data used was very well structured. The words or sentences were given, which made it easier to classify the accents because every speaker said the same. Using one of these models in a real-life scenario could decrease their accuracy because different words and word-combinations will highly likely be used.

A different approach has been introduced by Deshpande et. al [3] in 2005. Their main motivation does not primarily derive from the development from ASR systems, rather than improving voice biometrics as an authentication technique. The goal is to determine the role that features plays in discriminating between American English and spoken English with an Indian accent. The first step was to normalize the audio files. Following this, the removal of the silence was implemented by extracting the frames whose energy was below 0.15 of the average energy of the entire waveform. Sounds in speech can be divided into three groups:

- voiced,
- unvoiced,
- plosive.

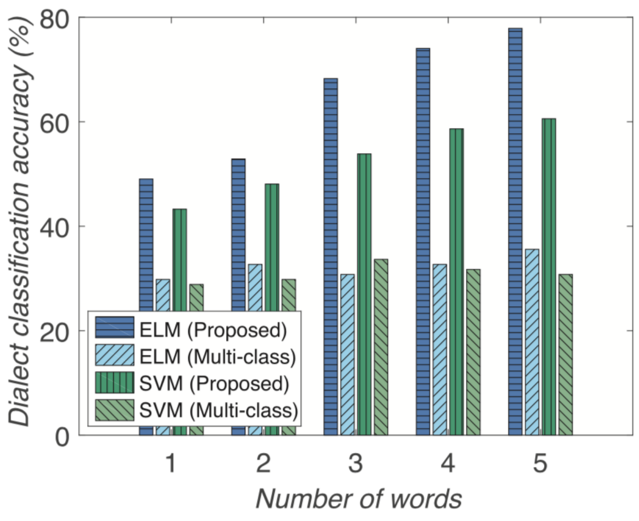


Figure 4.1: Improvement of accuracy with different numbers of words [13].

Accent Group	No of Persons	Training Files	Testing Files	Training	Testing
American	40	56	20	76.78%	85%
Indian	36	56	20	75%	87.5%

Table 4.2: Results achieved by Deshpande et. al [3].

Their next step was to extract only voiced frames of data. In comparison to the approaches introduced before, Deshpande et. al used formants, which have been described in Section 2.5, for feature extraction. Following the suggestion that the second and third formant play a big role in foreign accent classification compared to the remaining formants, they have isolated those using *linear predictive coding* (LPC). The algorithm was built using the Speech accent archive[3] dataset. The speech files have a sampling rate of 8 kHz. The speakers repeated a given paragraph text twice, which indicated that this dataset does not contain any continuous speech. Furthermore, the data corpus was collected in a quiet setting with no background noise using a head-mounted microphone. Deshpande et. al leave out prosody and intonation and concentrate on the formant frequencies since temporal features have larger intra-class variations. Therefore they have chosen a *hidden markov model* (HMM) classifier over a *gaussian mixture model* (GMM) based classifier. The results of this implementation are shown in Table 4.2. Out of the 76 in total, 40 people spoke with an American accent and 36 people spoke with an Indian accent. The training accuracy for the American accent amounts to 76.78%, whereas the Indian accent group reaches 75%. It is noticeable that 85% to 87.5% of the people have been classified correctly in the testing phase. Another approach, which has also

[3]http://accent.gmu.edu/

used formants as a feature extraction method, has been done by Themistocleous [16] in 2017. The purpose of his study was to use machine learning methods to automatic dialect classification using vowel formants and vowel dynamics. In comparison to other attempts, this method does not use English, but Greek dialects. The study provides a model of the two Modern Greek dialects, Athenian Greek (AG) and Cypriot Greek (CG). The dataset was conducted in a recording studio in Athens and at the University of Cyprus in Nicosia, avoiding no background noise and different qualities of the audio. The recorded material consists of a set of nonsense words, each containing one of the five Greek vowels. It is noticeable that no males have been recorded. 45 female speakers participated in the study, 25 from CG and 20 AG speakers. Each person produced 80 utterances resulting in a total of 3600 productions. The features were extracted using the open source software Praat[4]. The vowels were located and segmented manually and extracted by the software. The extracting includes the vowel formants (F1, F2, F3, F4) and vowel duration. For the classification following types of discriminative classifiers were evaluated:

- *linear discriminant analysis* (LDA),
- *flexible discriminant analysis* (FDA),
- C5.0.

One of the interesting part of this study is, it focuses not only on discovering the best method to classify the Greek dialects and vowels, but also how much each Vowel contributes to the classification [16]:

> “Notably, C5.0 classification showed that the most significant acoustic properties for the classification of the two Greek dialects are duration and the polynomial coefficients of F2, F3, F4, and F1—in this order. Also, it showed that the effects of the dialect pertain higher order formants, such as F3 and F4. In fact, F4 contributes more to the classification of dialect than F1, which—along with F2—plays an important role for the classification of vowel and stress.”

Using the C5.0 as an algorithm, an accuracy of 74% has been achieved. Although the approaches from both Deshpande et. al and Themistocleous did not use continuous speech in their implementation, the special aspect of this project relies on the feature extracting. In comparison to other attempts, they did not use MFCCs, but formants.

The last approach, which will be discussed in this section, is by Sunija, Rajisha and Riyas [15]. The objective behind their work is the creation of a new dialect database due to the limited availability of a Malayalam speech database. It is a South Indian language spoken in the state of Kerala. The goal is to differentiate between the two dialects Thrissur and Kozhikode. Because no database existed for this research, a small database has been created for these experiments. The corpus consists of 15 speakers for each dialect. They were asked to read 30 sentences in their own dialect. Furthermore, the audio was recorded in studio quality containing no background noise. For extracting the features from the audio files, MFCC has been used. As the next step, the following classifiers have been chosen:

- *artificial neural networks* (ANN),

[4] http://www.fon.hum.uva.nl/praat/

Classifier	Recognition Accuracy(%)
ANN	90.23
SVM	88.25
Naive Bayes	84.13

Table 4.3: Results achieved by Sunija et. al. [15].

- SVM,
- *naive Bayes.*

Out of those three algorithms, ANN achieved the best results, which is shown in Table 4.3. Although the results are encouraging, the fact of the lack of background noise and no continuous speech are inflating this outcome. Nevertheless, the dealing with the lack of data and the creation of their own data is particularly valuable.

4.2 Continuous Speech

Chu, Lai and Le [21] have used the foreign-accented English database, which has been produced by the Center for Spoken Language Understanding (CSLU) and distributed by the Linguistic Data Consortium (LDC). The dataset consists of English utterances by native speakers of 22 different languages. Each sample lasts for about 20 seconds and involves, in comparison to the approaches before, fluent continuous speech. For this classification task, they have used five languages (Tamil, German, Brazilian Portuguese, Hindi, and Spanish), because they have the most samples. As feature extraction methods MFCC and filter banks have been used. The first 200 features were used in each sequence. *Principal component analysis* (PCA) has been used for feature descriptors. Following algorithms have been used:

- *k-nearest neighbor* (KNN),
- *support vector classifier* (SVC),
- *multi-layer perceptron* (MLP),
- *long short-term memory* (LSTM),
- CNN.

As a baseline model, they have KNN and the SVC without PCA. For KNN *KNeighborsClassifier* was used and for SVC *LinearSVC* was implemented. A major part of their experiments deals with *hyperparamter tuning.* Chu et. al underline multiple times the importance of it because of the increase of the accuracy by as much as 19% in the MLP models. The summary of the best results from all classifiers is displayed in Table 4.4. The best accuracy on the test set could perform the LSTM model with 39.83%. The table also shows that there has been enormous overfitting. The results from the training set are mostly between 95% and 100%, in comparison, the test results reach from 28% to 39%. Furthermore, experiments with fewer languages were made, which resulted in 81.8% for classifying two languages. Although the results for classifying five different languages are not that promising, the accuracy for differentiating two languages is valuable. Furthermore, the project is highly relatable due to the fact that continuous speech

	Baseln. SVC	KNN	SVC PCA	MLP	LSTM	CNN
Test Acc	0.289	0.284	0.347	0.384	0.398	0.352
Train Acc	1.000	0.530	0.439	1.000	0.956	0.997
Tamil	0.215	0.446	0.246	0.339	0.431	0.169
Germany	0.556	0.349	0.603	0.651	0.730	0.635
Brazilian	0.275	0.308	0.297	0.374	0.286	0.341
Hindi	0.261	0.232	0.408	0.304	0.333	0.377
Spanish	0.148	0.066	0.197	0.262	0.262	0.246

Table 4.4: Highest accuracies achieved from all classifiers [21].

was used.

Lei and Hansen [7] have implemented a different approach in order to classify multiple dialects. Their motivation thrives from the importance of dialect as a factor next to gender that influences speech recognition performance. The first dataset consists of Arabic dialects from five different regions, including the United Arab Emirates (UAE), Egypt, Iraq, Palestine, and Syria. The corpus includes 100 speakers for each dialect, having a balance between male and female gender. The audio files contain conversations about topics such as the weather, shopping or travel. All discussions were recorded with a lapel microphone containing no disturbing background noise. The second corpus is based on the MIAMI[5] corpus, which contains dialect speech from Cuba, Peru, and Puerto Rico (PR). In this set, only female speakers are considered. Continuous speech consists of answers to interview questions. All recordings were taken using a close-talk headset microphone. The third and last corpus used includes three Chinese sub-languages:

- Mandarin,
- Cantonese,
- Xiang.

All recordings consist also of spontaneous speech and were achieved in a noise-free setting. Both male and female subjects are represented in this corpus. In all three corpora, silence removal is performed prior to training. Lei and Hansen are following the hypothesis that some parts in the signal are significantly different among the dialects, while others contain information that is dialect-neutral. Figure 4.2 displays the training strategy based on Gaussian mixture selection by the *Kullback–Leibler divergence* (KLD) (KLD-GMM). So in the decoding phase, the frames are divided into two classes: dialect dependent and dialect-neutral frames. Before this separation, the silence frames are first removed from the input data using an energy threshold. Moreover, features have been extracted using MFCC. In the training, the KLD is used to find the most discriminative GMM mixtures.

For the testing phase, the discriminative frames are detected using *frame selection decoding* (FSD) in combination with GMM (FSD-GMM). The best-achieved results from the Spanish dialects are 83.3%, in comparison to the Chinese corpus with 84.6%

[5]http://bangortalk.org.uk/speakers.php?c=miami

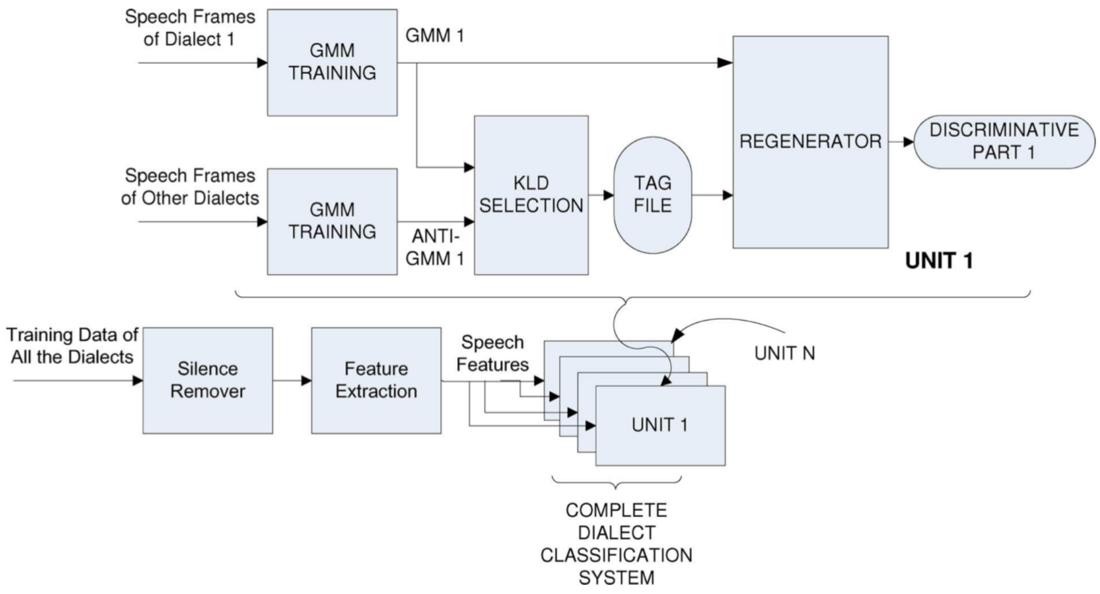


Figure 4.2: Training strategy [7].

and the Arabic corpus with an accuracy of 71.8%. Apart from the promising results, the most interesting feature is the separation between dialect dependent and dialect-neutral frame. The differences between dialects are not as big as the contrast between languages. Therefore only characteristic features should be extracted and used for the training and classification.

Chapter 5

Data

This chapter discusses the most essential and crucial part of this thesis, the data. Questions like, where does the data come from or which data cleaning steps have been applied, will be answered.

5.1 Creating the Dataset

A lot of challenges concerning data exist, which needed to overcome, in order to gain a valuable result in the end. The first problem deals with the lacking of a dataset containing Austrian dialects. Therefore several approaches have been carried out. The first idea was to gather the data from the Austrian TV Station called ORF directly. The received audio material contained telephone interviews, cultural broadcasts, press conferences, and many others. The quality, background noises, and loudness differentiated quite heavily. Also a lot of manual editing needed to be done in order to retain only those parts of the audio which contain a representative dialect. At first glance, a valuable dataset could be created, but at a closer look, one huge setback could be discovered. Because when people are faced with cameras or interviews, in general, they tend to speak as clear and understandable as possible, at the expense of their dialect. After confirming this assumption, the dataset became impractical.

The second dataset consists of several different approaches. The first method was to ask several students from the University of Applied Sciences Upper Austria to record themselves answering given questions (see Appendix B). The request was to answer one or two questions (e.g., Explain your daily way to work or How do you prepare for exams) and to speak in their original dialect. The second option was to download the audio files directly. The Viennese material could be downloaded from the City of Vienna[1]. Those videos contain interviews with people from society and politics describing their experiences with Vienna. The second website, where audio material has been downloaded, is called Literaturradio Vorarlberg[2]. The goal of this platform is to present and archive audio of the Vorarlberg dialect. Since this is an initiative that emanates from Vorarlberg, texts by Vorarlberg authors are currently being heard, as well as recordings of literary

[1]https://www.wien.gv.at/video/Wir-und-Wien

[2]http://www.literaturradio.at/kategorien/mundart/

Federal State	Source	People
Vienna	Wir und Wien	71
Vorarlberg	University sumbissions	37
Vorarlberg	Literaturadio	33
Vorarlberg	Mundartradio	47

Table 5.1: Sources of the dataset.

Federal State	Female	Male
Vienna	24	47
Vorarlberg	65	52

Table 5.2: Gender distribution.

Federal State	Length in Minutes
Vienna	286
Vorarlberg	245

Table 5.3: Total length of audio files.

events that took place in Vorarlberg. The resource is the Mundartradio[3]. The purpose of this website is the conservation of the Vorarlberg dialect, by uploading videos from various events. Several different people present their written poems and stories.

Table 5.1 displays the different sources for each federal state. For the Viennese data, only the interviews from Wir und Wien were considered because no other dialect recordings or programs existed. Those interviews were taken place with 71 different people. Three sources have been selected for the Vorarlberg data. The first one is the submissions from the University of Applied Sciences Austria, where 37 students have sent in their recordings. The second part comes from the Literaturadio containing 33 different speakers. The third and final source is Mundartradio with 47 people. One crucial aspect of classifying dialects is to have a good balance between female and male speakers, to avoid that a certain gender is always classified as the same dialect. If that happens the algorithm did not create a model, which is able to differentiate between the dialects of Austria but to characterize gender from audio. Therefore one aspect was to create a good balance between the genders, which is shown in Table 5.2. Although there is a slight imbalance in the Viennese data, the gap is not too critical.

All sources added up, the dataset on the Vorarlberg side result in a total of 266 minutes, which is compared in Table 5.3. The original length of the Viennese data is 3203 minutes but has been cut down to four minutes per speaker, which results in a final length of 286. The motivation behind this is to avoid a huge imbalance between the two federal states. That makes it easier for the algorithm to differentiate the two classes

[3]https://www.mundartradio.at/de/mundartradio/

and no class imbalance countermeasure needs to be taken. It is noticeable to mention that all audio files contain no scripted list of words. Although some of the recordings contain the same topic, this dataset contains only continuous speech.

5.2 Data Cleaning

The majority of the data contains hardly any background noise and has been recorded in a silent setting. Nevertheless, there are essential data cleaning steps taken in order to achieve the best basis as possible. So this section describes the process from the original dataset to the cleaned data, that can be used as an input in the preprocessing pipeline, which is described in Chapter 6. The first step is to convert all audio files to the same quality. Following four features have been adapted:

- sample rate,
- bits per sample,
- audio channel,
- format.

Because the audio files have been recorded by different people and settings, they all determine a different quality. Therefore the *sample rate* has been changed to 44.1 kHz, the *bits per sample* were set to 32, the *audio channel* was defined to mono and the *format* was set out to *wav*. The second measure was to manually remove all unwanted sound such as clapping or laughing. The part is similar to the elimination of background noise with the difference because both contain undesirable audio. The difference is that the noise is always present and can not be cut out by hand.

The third and last step was to remove the breaks every speaker makes between words. After splitting all files into small ones, the amplitude of every file is checked. Those with the lowest altitude will be deleted. Program 5.1 takes the `path` and the `files` as an input and returns the files, which should be deleted. Line 5 and 6 define the number of files, which should be removed. In this case, the lowest 5% will be cut out. For every file in the list, the raw data is received by using `wavfile.read`. Because the data also consists of negative values, the absolute value of each data point is taken in Line 13 using `abs(i)`. In Line 16 the name of the file is also added to the array. This is not done for any calculation purposes, but for checking if the program wants to remove files, which still contain voice instead of breaks. If this happens, the percentage needs to be reduced from 5% to 4% and check again if it is now optimal. After summing up all data points, the array is sorted ascending in Line 24. Finally, the unwanted audio files are returned in Line 25.

After applying all the described measures to the data the length of the audio files has changed. It has dropped from 284 minutes to 266 minutes for the Viennese side. The Vorarlberg data length has been cut down from 245 minutes to 229 minutes in total. In both cases, the data has been reduced by around 6.5%. After the data has been converted to the same quality, removed from unwanted sounds and cleaned from silence, it can be now used as an input in the preprocessing pipeline.

```python
def check_for_files_to_delete(path, files):
    result = []
    position = []

    away = len(files)*0.05
    away = round(away+0.49)

    for index, file in enumerate(files):
        try:
            fs, data = wavfile.read(path + file)
            sum = 0
            for i in data:
                sum += abs(i)
            position.append(sum)
            position.append(index)
            position.append(file)
            result.append(position)
            sum = 0
            position = []
        except:
            print ("An error occured")
            print (file)

    to_delete = sorted(result)[:away]
    return to_delete
```

Program 5.1: Remove silence.

Chapter 6

Technical Approach

In this chapter, the technical approach and the connecting pipeline is presented. Figure 6.1 shows the process of the data preparation from the raw data to the state where the data can be used as an input for the machine learning algorithms. It is necessary to understand that in each section only one option is considered to be choosable (e.g., either the EBU normalization or the peak normalization is applied). This possibility of options leads to 48 different paths. Due to the immense complexity and time factor, only a few of those paths are realized. Why and which paths have been implemented will be discussed in Chapter 7. There are four states, which need to be covered and explained. The following stages will be accompanied by the implemented code sections:

- window size,
- normalization,
- noise reduction,
- feature extraction.

6.1 Window Size

The first variable to choose is the window size. In this context, it means the length by which every audio clip is divided. Because the input data consist of a large variety

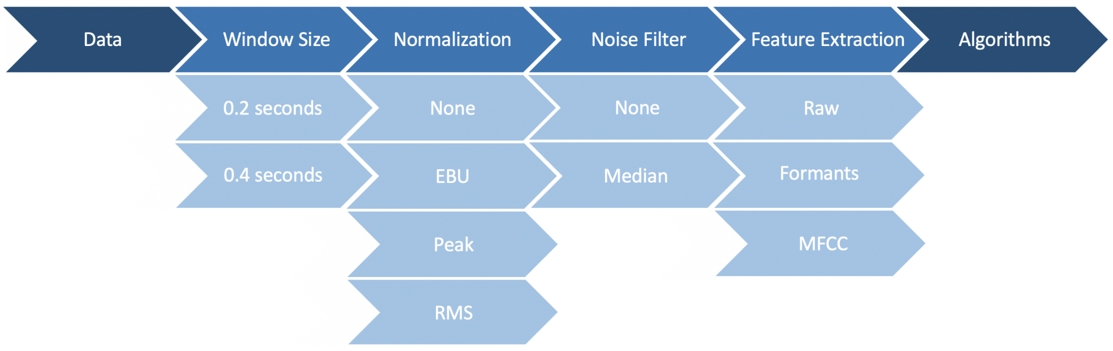


Figure 6.1: Preprocessing pipeline.

```
1 prefix = 0
2 cmd_str = 'ffmpeg -i {tr} -f segment -segment_time {ti} -c copy {path}%03
      d.wav'
3 for track in track_list:
4     command = cmd_str.format(tr=path+track, ti=time_seconds,
5     path=new_path+name+(str(prefix)))
6     call(command, shell=True)
7     prefix += 1
```

Program 6.1: Splitting audio files according to given time in seconds.

of different lengths, it is necessary that all are equal. Furthermore, it is required by the machine learning algorithms that all input data are having the same amount of features. On top of that, the selected window size should not be too large because of the complexity which is not beneficial for the computing time of the algorithms. Following two lengths have been implemented and will be discussed in this section:

- 0.2 seconds,
- 0.4 seconds.

Cutting the audio files to a specific window size and saving them as individual files can be realized using the FFmpeg[1] framework. This framework is capable of decoding, encoding, transcoding, muxing, demuxing, streaming, filtering, playing and many other editing functions related to audio. There are two options for how this framework can be implemented. The first one to use the associated python package and the second option is to execute the FFmpeg command on the shell via python. In this case, the implementation was realized using the latter one. After several experiments, Sheng and Edmund [26] achieved the best result with their classification task using a time window of 0.18 seconds. Applying 0.2 seconds to the FFmpeg framework, most audio clips are trimmed to a length of around 0.18 seconds. Therefore 0.2 seconds are listed above instead of the suggest time window. Because of that, all sections below 0.18 seconds are disposed of. Although Sheng and Edmund achieved valuable results, it is questionable if such a short period contains enough characteristics for each dialect. Therefore not only 0.2 seconds, but also 0.4 seconds have been evaluated.

Program 6.1 displays the implemented approach. In Line 2 the command, which is executed on the shell in Line 6, is listed. For every track in a given list, the trimming will be executed. The prefix variable in Line 1 serves as the possibility to name all resulting files differently so no overwriting can take place. The most important argument is the `time_seconds` which is in this case either 0.2 or 0.4.

6.2 Normalization

The theoretical concepts and the motivation for using normalization as data preparation steps have been explained in the related Section 2.7. As shown in Figure 6.1 there are four different possibilities to choose from:

[1] https://ffmpeg.org/

- none,
- EBU,
- peak,
- RMS.

Using the FFmpeg-normalize[2] package all variants mentioned above can be implemented. The EBU R128 loudness normalization procedure is set to default. This package takes several audio files and writes them to a specific folder. All audio streams are normalized so that they have the same volume based on the selected standards. So the `input` and `output` variable in the given code examples represent the path to the folders, in which the original audio files are stored and the new ones should be saved to. Executing the following command line, the audio will be normalized to the suggested target level of $-23.0\ LUFS$:

```
cmd_str = 'ffmpeg-normalize {input} -o {output}'
```

Using the peak normalization all audio streams are normalized to $0\ dB$, which can be specified with the `target-level` option:

```
cmd_str = 'ffmpeg-normalize {input} --normalization-type peak --target-
    level 0 --output {output}'
```

Following command line represents the code for implementing the RMS-based approach:

```
cmd_str = 'ffmpeg-normalize {input} --normalization-type rms --target-
    level 0 --output {output}'
```

In this case, the audio files are also normalized to $0\ dB$, which can also be adapted to the desired loudness. The fourth option, which does not take any measures, means skipping this step (option none) and evaluate this impact.

6.3 Noise Reduction

The third state of the preprocessing pipeline follows up the removal of unwanted background sounds. The theoretical concept and the necessity behind the usage of noise reduction have been elucidated in Section 2.6. Two different filters have been described in the mentioned Section. One of them, the Kalman filter, can not be realized in this scenario. For this filter, an audio file of the signal with background noises and an audio file with just background noises is necessary. Because the background noise can change over time, an audio clip containing the exact same noise does not exist. Furthermore, there are different disturbances, a universal noise signal also results in no usable outcome. The following python method `reduce_noise_median` applies the median filter to the audio signal:

```
def reduce_noise_median(y):
    y = scipy.signal.medfilt(y,3)
    return (y)
```

[2]https://github.com/slhck/ffmpeg-normalize

```
1 def get_raw_from_files(path, audio_files):
2     raw_list = []
3     for file in audio_files:
4         fs, data = scipy.io.wavfile.read(path + file)
5         data = np.array(data)
6         raw_list.append(data)
7     return raw_list
```

Program 6.2: Extracting features from RAW audio.

The implementation is based on the Noise Reduction project[3] using SciPy[4]. This function receives an audio matrix (y) and returns the matrix after gain reduction on noise. So it performs the median filter on an n-dimensional array. Figure 2.14 in Section 2.6.1 displays the case when only the direct neighbors have been used for calculations. In this case, not one, but three neighbors have been used for the implementation, which indicates the option 3 in Line 2. The second option, which does not take any measures, means skipping the state of noise reduction (option none) and evaluating this impact.

6.4 Feature Extraction

After choosing the window size, applying normalization of the audio data and reducing the background noise, the final state discusses the various options on how the features can be extracted best from the signal. Three different possibilities exist:

- raw,
- formants,
- MFCC.

The first and easiest one is to obtain the wanted features by simply reading the audio in raw form. This method is similar to the none option in the previous two states because no calculation or changes are carried out. The only different part is that although nothing is done to the data itself, it still needs to be prepared for the algorithm so it can be handled as an input. The function shown in Program 6.2 takes the `path` and the names of the audio files (`audio_files`) as an input. In Line 4, using also the SciPy package, the sample rate (in samples/sec) and the raw data from every file in the list are extracted.

The second and more complex option describes the feature extraction with the usage of formants. The theoretical concept behind this method has been described in Section 2.5. Extracting formants from a file is done with the Praat software. The big advantage of this software is, that the signal and all formants can be viewed and edited, such as Figure 2.13. But the main disadvantage is that it is hard to extract multiple formants in several files. Therefore the python package praat-parselmouth[5] by Yannick Jadoul is essential and helpful because it provides a complete and Pythonic interface to the internal Praat code. Function `get_formant_form_files` displayed in Program 6.3

[3]https://github.com/dodiku/noise_reduction
[4]https://www.scipy.org/
[5]https://github.com/YannickJadoul/Parselmouth

```
1  def get_formant_from_files(path, audio_files):
2      formant_list = []
3      for file in audio_files:
4          # get sound
5          sound = parselmouth.Sound(path+file)
6          formant = sound.to_formant_burg(window_length=0.01,
       maximum_formant=5500.0)
7          formant_no = 0
8          formant_section = []
9          # only get first four formants
10         while formant_no < 4:
11             formant_no += 1
12             cur = 0.03
13             end = 0.16
14             formant_rows = []
15             while cur < end:
16                 tmp_formant = formant.get_value_at_time(formant_no, cur)
17                 if math.isnan(tmp_formant):
18                     tmp_formant = 0
19                 formant_rows.append(tmp_formant)
20                 cur += 0.01
21             formant_section.append(formant_rows)
22         formant_list.append(formant_section)
23     return formant_list
```

Program 6.3: Extracting features using formants.

shows the corresponding code section. Identical to the last code example, this function also takes the `path` and the names of the audio files (`audio_files`) as an input. All extracted formants are stored and later returned in the variable `formant_list`. First, for every file, the signal is extracted using `parselmouth.Sound` in Line 5. Following the formant object is created in Line 6 using the class method `to_formant_burg`. The `maximum_formant` parameter describes the highest frequency, up to which formats should be calculated. Defining the `window_length` to 0.01 makes it possible that every 0.01 seconds formants can be extracted later. This object serves as a descriptive file in order to be able to extract the formants. It contains information such as the length of the audio file, the start and the endpoint and the number of frames. The next step is to actually get the formants and add them to the list. Line 16 describes the process of extracting every formant in every time step (0.03 seconds to 0.16). The method `get_value_at_time` in Line 16 takes the formant number (1 to 4) and the current time step as an input and returns the wanted value in Hz. In case a formant could not be extracted at this specific time, it is set to 0, which is displayed in Line 17 and 18. When all formant rows are added, the whole bundle can be finally returned in Line 25.

The third and last option of extracting formants is to use the MFCC based approach, which has been explained in Section 2.4. Several packages exist, that support the extraction of features using Mel-frequency cepstral coefficients, but in this case, the

```python
1 def track_features(path, time_series_length, features_size, index):
2     print("Currently Extracting ", path, index)
3     y, sr = librosa.load(path)
4     mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
5     # creates array filled with zeros
6     features = numpy.zeros((time_series_length, features_size))
7     features[:, 0:13] = mfcc.T[0:time_series_length, :]
8     return features, index
```

Program 6.4: Extracting features from MFCC.

best option was to use the package LibROSA[6]. This option does not only allow the calculation of the MFCCs, but also the representation of this method. The implemented code, which is shown in Program 6.4, has been adapted from the original project, which was done by theSoenke[7]. For every audio file the function `track_features` needs to be called. It has several necessary input fields:

- `path`,
- `time_series_length`,
- `features_size`,
- `index`.

In comparison to the other program codes before, the `path` variable, in this case, contains the path including the name of the audio file. The `time_series_length` input value describes how long the input stream is. The longer the audio file, the higher this number (e.g., for 0.18 seconds it takes the value of 8). The third value is the `feature_size`. As it is in this example, this number is typically set to 13. These two parameters already define the shape of the features, which is in this case, 8×13. The last parameter, the `index`, serves the purpose of keeping track of the extracting files, which is shown in Line 2. After receiving all the necessary input values, the main part starts in Line 4. Here the audio file is loaded as a floating point time series using the `librosa.load` method. The returned values are the audio time series (`y`) and the sampling rate of y (`sr`). Following the MFCCs can be extracted by submitting the audio time series, the sampling rate and the feature size to the `librosa.feature.mfcc` method. This step is displayed in Line 5. Next, a new array needs to be created. The returned MFCC values are stored within this array, which is shown in Line 8 and Line 10. Finally, the finished array along with its index can be returned in Line 12.

After completing all these four states, the data has now been cut to the right window size, normalized to a specific level, removed by unwanted background noises and finally the features have been extracted and stored in a multidimensional array. With all the steps completed the data is now ready to be used as an input for the machine learning algorithms.

[6]https://librosa.github.io/librosa/
[7]https://github.com/theSoenke/VoiceClassification

Chapter 7

Results

In Chapter 6 the preprocessing pipeline and its states have been introduced. The purpose of the current chapter is the discussion and the produced results of the implemented paths. Because of its huge complexity due to the immense number of possible paths, only a few could be evaluated. Those handfuls of paths are tested for the best feature extraction method and for the optimal algorithm. Moreover, *hyperparameter tuning* will be applied to achieve the best results possible. The success of the approaches is measured by test accuracy.

7.1 Baseline Model

The first one serves as a so-called *baseline model*. This signifies that the accuracy of this model will be compared to all the other models. In order to achieve the most basic selection of states, the window size has been set to 0.2 seconds. Furthermore, the raw data will serve as a feature extraction method, which is displayed in Figure 7.1. Because the calculations now have been made with the raw data, every input out of the 118815 training data, has 8064 features. Therefore the training itself is absolutely time-consuming and the support vector machine algorithm could not finish within 24 hours. So the CNN was chosen as the baseline's algorithm. Program 7.1 displays the selected base model. The *input_shape* value in Line 2 depends on the selected feature extraction method (e.g., the MFCC features consist of a 8×13 feature vector). The model achieved a training accuracy of 51.93% and a test accuracy of 51.34%. This result is only slightly better than choosing randomly to which class an instance belongs. Questioning the feature extraction method, where simply the raw data is read from each file, the poor result is predictable.

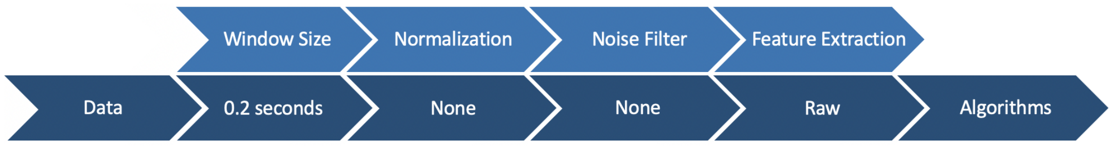


Figure 7.1: States of the baseline model.

```
model = Sequential()
model.add(Conv2D(64, kernel_size=1, activation="relu", input_shape
    =(8,13,1)))
model.add(Conv2D(32, kernel_size=1, activation="relu"))
model.add(Flatten())
model.add(Dense(2, activation="softmax"))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics
    =['accuracy'])
```

Program 7.1: CNN model structure of the baseline model.

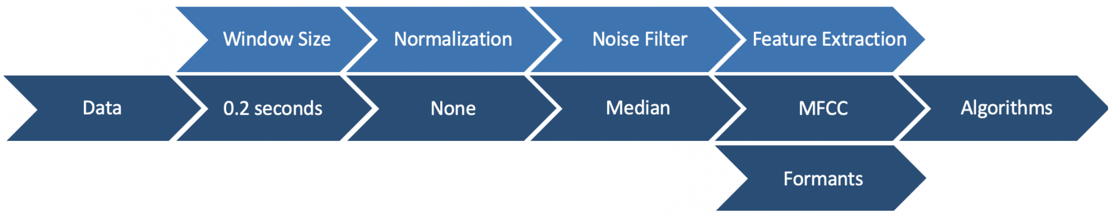


Figure 7.2: States of the noise filter model.

7.2 Noise Filter Model Using MFCC and Formants

After the rather dissatisfied performance of the baseline model, the second approach implies two main differences. The first one is the addition of a noise filtering method, the median filter. The second change covers the choice of the feature extraction method. Both, the MFCC and the formants, have been tested. The full selection of the states is displayed in Figure 7.2. The first big change for the implementation was the reduction of the features. In contrast to the baseline model, where 8064 features were taken under consideration, the MFCC method only used 104. Even smaller was the feature size using the vowel formants with 52 features per instance. The second change was that this reduction concluded to a desired decrease in the training time, so the support vector machine algorithm could be used as well.

Figure 7.3 displays the test accuracy of the two feature extraction methods using CNNs and SVM. Both models consist of a basic implementation with no hyperparameter tuning done so far. For the support vector machine algorithm, the *SVC* model with its default settings was used. The first and important takeaway from this result is the outperformance of the MFCC in comparison to the formants. The accuracy from the later feature extraction method could only achieve 46.99% using CNNs, which makes it not only worse than the baseline model, but also worse than pure guessing. Using support vector machines could increase the performance reaching 64.93%, but is still beaten by the MFCC approaches with 75.16% and 79.68%.

After evaluating the basic models of these two algorithms, the goal is now to optimize those results performing hyperparameter tuning. Program 7.2 displays the final model. The first aspect is the *filters* used in the convolutional layers (128 and 64).

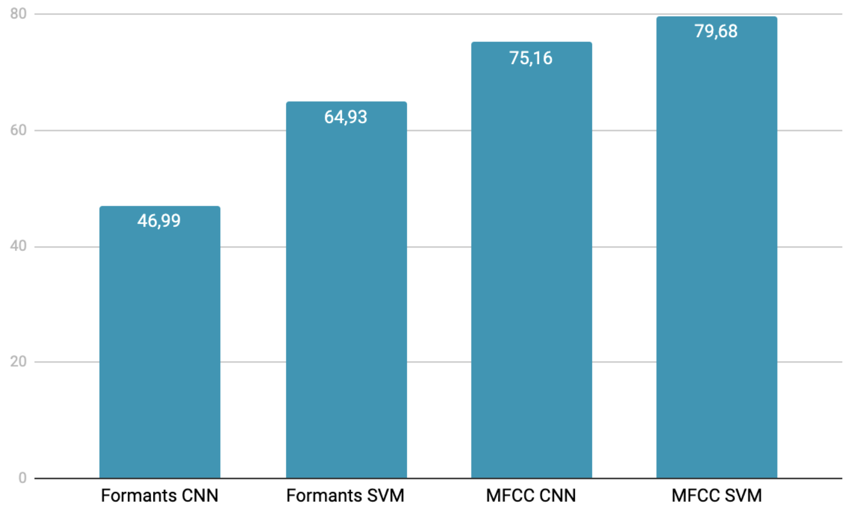


Figure 7.3: Test accuracies of the noise filter model.

```
1  model = Sequential()
2  model.add(Conv2D(filters=128, kernel_size=(4, 4), activation="relu",
       input_shape=(8,13,1)))
3  model.add(Conv2D(filters=128, kernel_size=(3, 3), activation="relu"))
4  model.add(Conv2D(filters=64, kernel_size=(3, 3), activation="relu"))
5  model.add(Dropout(0.4))
6  model.add(Flatten())
7  model.add(Dense(2, activation="softmax"))
8  model.compile(optimizer='adam', loss='categorical_crossentropy', metrics
       =['accuracy'])
```

Program 7.2: CNN model structure of the improved model.

Those values define the number of output filters in the convolution. Second the *kernel_size* has been set to (4,4) and (3,3). This specifies the height and width of the 2D convolution window. Different *activation* functions have been tested and the best performance could achieve the so-called *relu*. Following a *dropout* rate is applied. This consists of randomly setting 40% of the input weights to 0 at each update during training time. Moreover, the layers are flattened in Line 6 and the *softmax* activation to achieve the same shape as the input. In the last step, the model gets compiled using the *adam* optimizer. It is noticeable that *batch normalization* and *max pooling* have not led to better results in this case and also in the following approaches.

Figure 7.4 shows the achieved improvements using hyperparameter tuning on the CNN model. The initial accuracy could reach 75.15%. After adding more hidden layers the performance increased by 3.15%. Following the kernel size has been increased and the dropout layer has been added. Finally, in the training five instead of three epochs have been used. All changes have led to the final accuracy of 82.05% on the test data.

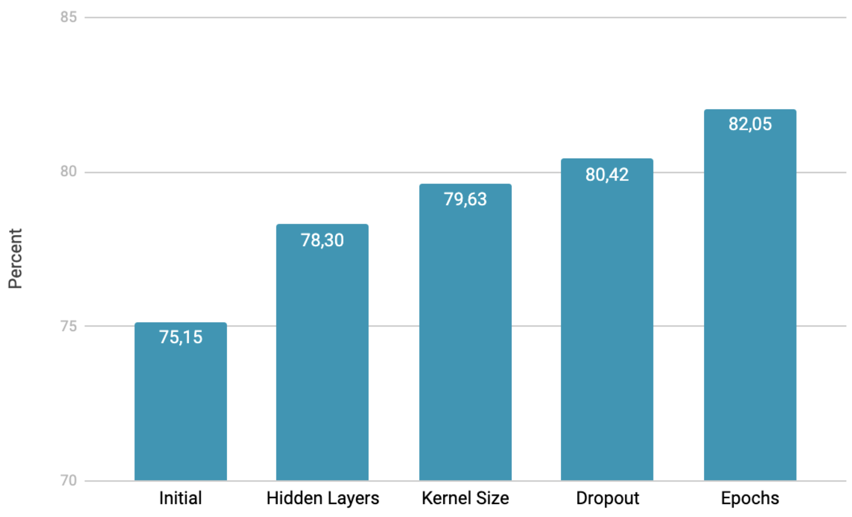


Figure 7.4: CNN test accuracies of the tuned noise filter model.

	Train Accuracy	Test Accuracy
Initial model	87.41%	79.46%
Improved model	87.67%	80.54%

Table 7.1: SVM accuracies of the noise filter model.

Because the MFCC feature extraction method outperformed the vowel formant approach, this path has been chosen too for the hyperparameter tuning for the support vector machine algorithm. At first, the baseline-model performance needed to be measured in order to compare the hyperparameter tuning. Therefore the standard *SVC* model with its default settings has been used. This approach has achieved a test accuracy of 79.46%, which is shown in Table 7.1. Following, multiple grid searches have been performed resulting in the tuning of two parameters. The first one is increasing the C from 1.0 to 3.0, which can be described as the penalty parameter of the error term. Increasing this value may lead to better results, but also carries the risk of overfitting the training data. The second modified hyperparameter is called *gamma*. This is a parameter for non-linear *hyperplanes*. The higher the value, the more exactly does it fit the training data. Its default value depends on the number of features used. In this case, the default value has been increased from 0.0096 to 0.02. Changing those two hyperparameters, the test accuracy reached 80.54%.

In this first approach of improving the baseline model, which has been introduced in Section 7.1, the CNN algorithm in combination with the MFCC feature extraction method could achieve the best results with a test accuracy of 82.05%.

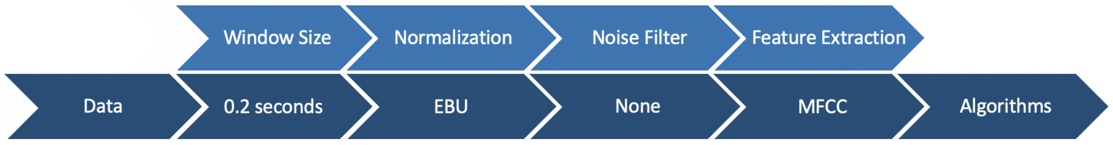


Figure 7.5: States of the audio normalization model.

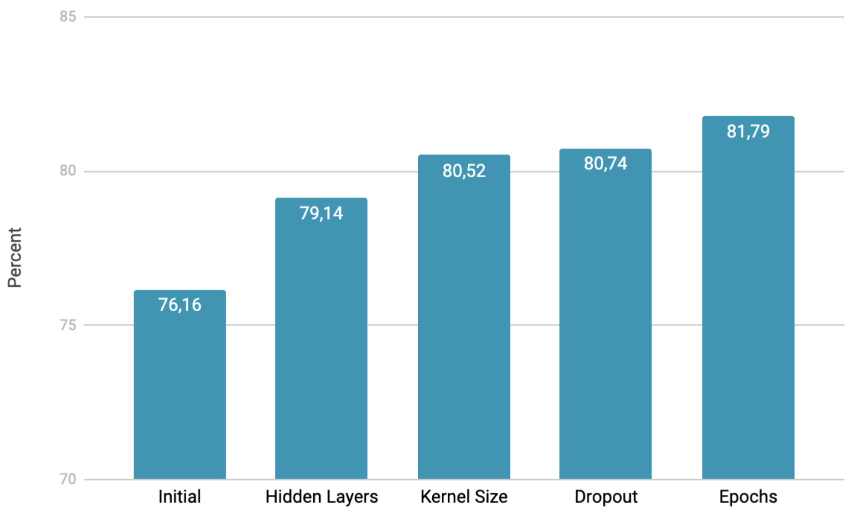


Figure 7.6: CNN test accuracies of the tuned audio normalization model.

7.3 Audio Normalization Model Using MFCC

In Section 7.2 a huge improvement could have been achieved using different feature extraction methods and adding the median filter to the states. In this section, the EBU normalization standard is added. Due to its prior high performance, only MFCC will be used for feature extracting. Figure 7.5 displays the current choice of states for this path. In comparison to the previous path, no noise filter has been used. After implementing the baseline model, which is the same as in Section 7.2, similar results were accomplished. The CNN algorithm reached an accuracy of 76.16%, whereas the SVM performed even better with 79.57%. The next step was to improve those results using again hyperparameter tuning. Figure 7.6 shows the development for the CNN classifier. After adding more hidden layers, increasing the kernel size, adding a dropout rate of 40% and raising the epochs, the final accuracy has climbed to 81.79%. Surprisingly the best model, which has been discovered using the hyperparameter tuning approach, is the same as the one from the previous path, which is described in Program 7.2.

After calculating and improving the results of the CNN model, the next step is to do the same for the support vector machine classifier. First, the base model was determined using the *SVC* standard model from sklearn. This resulted in a test accuracy of 79.57%, which is similar to the one achieved in Section 7.2. Following, for improving the result,

	Train Accuracy	Test Accuracy
Initial model	88.28%	79.57%
Improved model	88.82%	80.10%

Table 7.2: SVM accuracies of the audio normalization model.

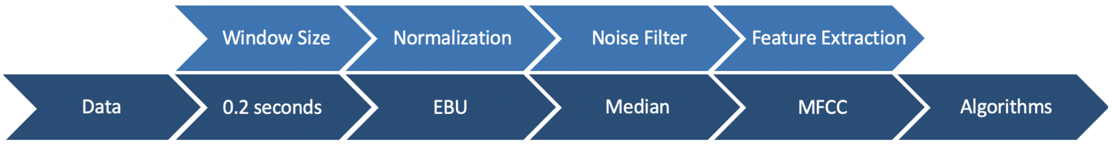


Figure 7.7: States of the combined model.

the *GridSearchCV* has been used, to find the optimal parameters. This search also found out that two hyperparameters needed to be changed. The first one is the gamma, which increased, similar to the previous Section, to 0.02. The second modified value is the C. In this case, it is now set to 4. After changing those two values, the accuracy increased to 80.10%, which is displayed in Table 7.2. After performing hyperparameter tuning on both algorithms, the CNN algorithm outperformed the SVM approach reaching a test accuracy of 81.79%.

7.4 Combined Model

In this section, the last approach using a window size of 0.2 seconds will be covered. Figure 7.7 displays the choices of states for this approach. Additionally to the EBU normalization standard, the median filter is also used trying to filter out the unwanted background noise. As a feature extraction method, MFCC is used, due to its performance on the previous attempts.

The first step here was also to implement the baseline model, using the standard SVC classifier for the support vector machine algorithm and a basic model for CNN. The initial result was similar to the approaches before reaching an accuracy of 79.89% for the SVM and 76.16% for the CNN. Following, hyperparameter tuning was used on both algorithms to improve the performance. Figure 7.8 shows the progress of the CNN algorithm. Similar to the other paths before, the accuracy increased by adding hidden layers and expanding the kernel size. In comparison to the first two approaches, a dropout rate of 10% instead of 40% has been used. Additionally, only four epochs, instead of five have been needed in order to achieve the best result possible. The final accuracy of this algorithm could improve by 5.63% reaching 81.79%.

After tuning the hyperparameters for the CNN algorithm, the next step is to do the same for the SVM classifier. Similar to the previous attempts, the parameters C and gamma have been altered. The best result has been achieved by setting C to 3.0 and gamma to 0.02. Table 7.3 shows those results. The test accuracy slightly improved from 79.89% to 80.65%. In this case, overfitting is an issue because the training accuracy

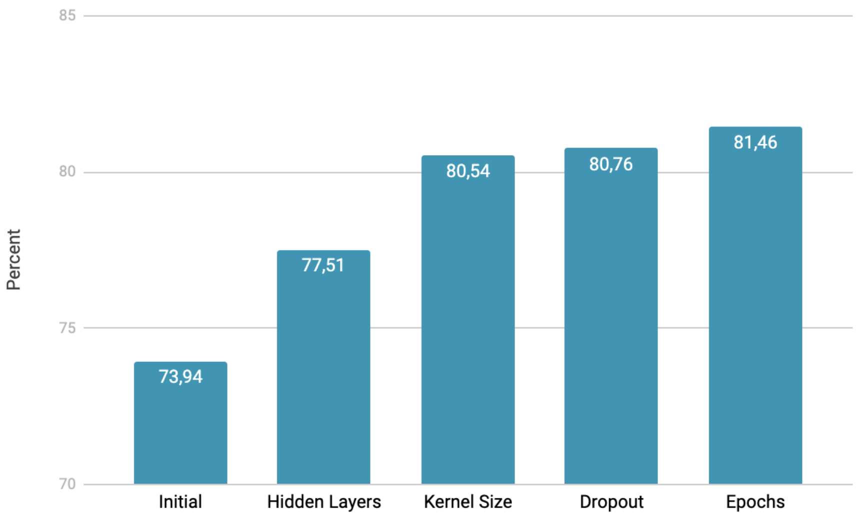


Figure 7.8: CNN test accuracies of the tuned combined model.

	Train Accuracy	Test Accuracy
Initial model	88.28%	79.89%
Improved model	95.80%	80.65%

Table 7.3: SVM accuracies of the combined model.

after the hyperparameter tuning is too high in comparison with its test accuracy. In this case the best method is to use the CNN algorithm in combination with the MFCC feature extraction method, which leads to the test accuracy of 81.79%.

7.5 Large Window Size Model

After testing three different states of the preprocessing pipeline, the best test accuracy has been achieved by the noise filter model, which has been described in Section 7.2. The SVM classifier reached 80.54%. The CNN classifier performed even better with an accuracy of 82.05%. This section evaluated the influence of doubling the window size from 0.2 seconds to 0.4 seconds Figure 7.9 displays those selected states. After increasing the window size to 0.4 seconds, the feature size went up as well. Initially, 104 features existed. Following, 221 features were extracted due to the lager window. In contrast, the number of data items dropped. The training items decreased from 118815 to 60112. The test items declined from 29757 to 14973.

At first, the baseline models for both algorithms were measured. As baseline models, the SVC standard model from scikit-learn[1] has repeatedly been used for the SVM classification. For the CNN classifier, the model structure, which is displayed in Program 7.1

[1] https://scikit-learn.org/stable/

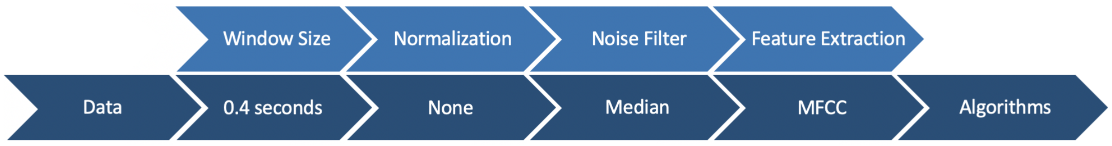


Figure 7.9: States of the large window size model.

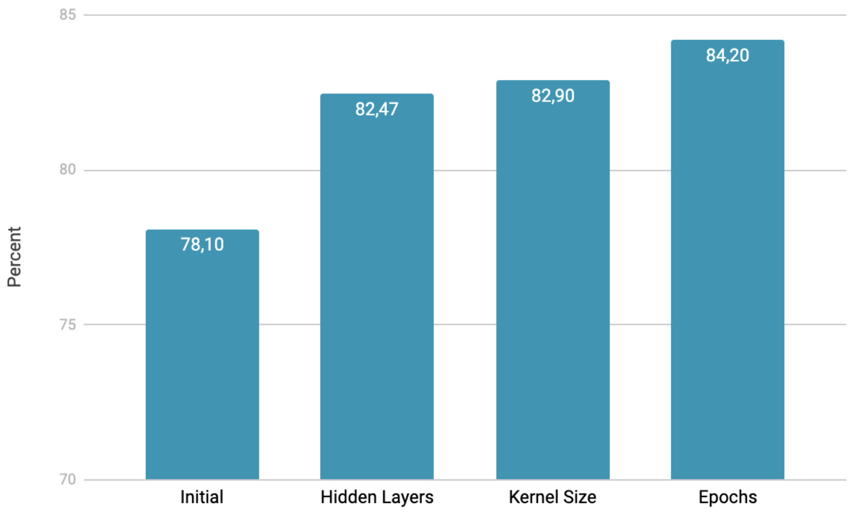


Figure 7.10: CNN test accuracies of the tuned large window size model.

applied. This baseline model for using the SVM algorithm could achieve a test accuracy of 81.95%. At first, these results appear promising, but overfitting is an issue due to its high training accuracy of 90.93%. In contrast, this problem did not affect the CNN algorithm. In this case, the test accuracy resulted in 78.10% with a training accuracy of 78.00%.

The next step was to perform hyperparameter tuning for the CNN algorithm in order to achieve better results. Figure 7.10 displays those improvements. The first step here was the increase the *hidden layers*, which lead to the test accuracy of 82.57%. The next step was to change the *kernel size* from 1 to (2, 2), which resulted in slightly better accuracy of 82.90%. The last approach was to increase the epochs. In this case, eight epochs were the optimal choice for reaching the best results with a test accuracy of 84.20%. Program 7.3 displays the model structure of this approach. In Line 2 it is visual that the window size has changed by the different *input_shape* of (17,13,1) in contrast to the previous (8,13,1).

After tuning the CNN model to its best, the SVM model also had a similar procedure. Using the *GridSearchCV* from scikit-learn, the results have improved slightly. As in the previous attempts tuning the C parameter and the *gamma* value lead to the best performances. First, the C has been increased from its initial value of 1.0 up to 3.0. Second, the gamma has been changed from 0.0045 to 0.01. Both adjustments have led

```python
model = Sequential()
model.add(Conv2D(128, kernel_size=(2, 2), activation="relu", input_shape
    =(17,13,1)))
model.add(Conv2D(128, kernel_size=(2, 2), activation="relu"))
model.add(Conv2D(64, kernel_size=(2, 2), activation="relu"))
model.add(Flatten())
model.add(Dense(2, activation="softmax"))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics
    =['accuracy'])
```

Program 7.3: CNN model structure of the large window size model.

	Train Accuracy	Test Accuracy
Initial model	90.93%	81.95%
Improved model	98.68%	82.05%

Table 7.4: SVM accuracies of the large window size model.

	SVM	CNN
Baseline model	-	51.34%
Noise filter model	80.54%	82.05%
Audio normalization model	80.10%	81.79%
Combined model	80.65%	81.46%
Large window size model	82.05%	84.20%

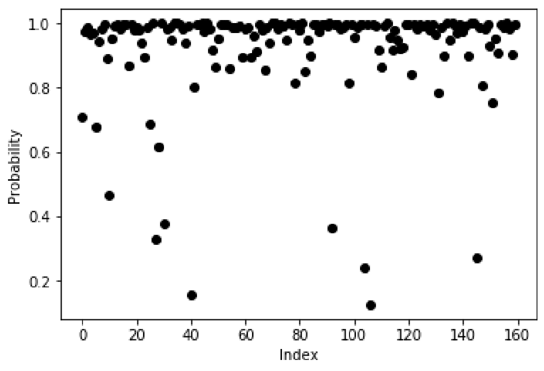
Table 7.5: Highest achieved test accuracies of all approaches.

to the final test accuracy of 82.05% which is 0.1% better than the baseline model. Furthermore, overfitting is also an issue here, due to the high train accuracy of 98.68%. Table 7.4 sums up the performance of the SVM classifier for this approach.

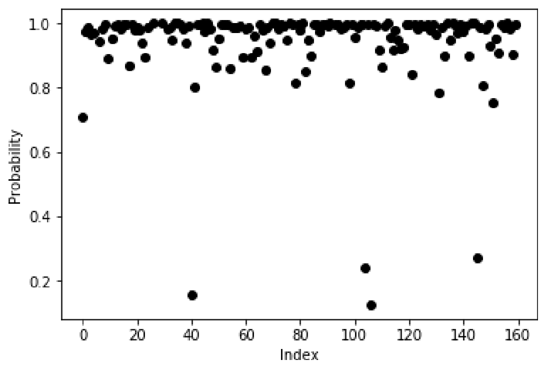
After implementing both algorithms, it is clear that the CNN approach is the better choice in this case. This decision is not only based on the higher test accuracy of 84.20% in comparison to 82.05%, but also on the fact that the SVM model deals with overfitting. In this section, five paths were implemented. This resulted in nine models. Those results were compared and the best approach was evaluated. Table 7.5 displays the test accuracies from all models. The best approach, in this case, is those with the state choices from the large window size model in combination with the CNN algorithm.

7.6 Validation of Results

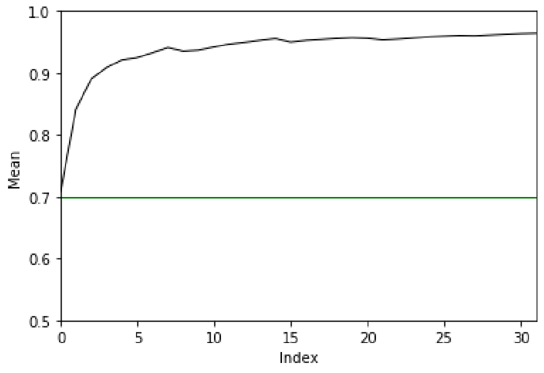
In the previous sections five different paths have been implemented, compared and the best model was chosen. The next step is to validate this model. Therefore five unseen Vorarlberg audio recordings and five unseen Viennese files are used for this evaluation.



(a)



(b)



(c)

Figure 7.11: Different validation approaches. *All data points considered* (a), *Data points above 70% and below 30% considered* (b), *Sequential decision with data points above 70% and below 30% considered* (c).

Every single file is tested separately. Those files run through the same states as the training data and test data. The chosen steps of the preprocessing pipeline are displayed in Figure 7.9. After splitting the file into 0.4-second pieces, removing the breaks and applying the median filter, every file is validated by the model.

Three different approaches exist for the evaluation. The difference lies in the provision of the data points and in the decision theory. Figure 7.11 shows those different approaches. In all cases, the same Vorarlberg file has been taken for testing. Each fragment of the audio file is predicted separately. Therefore several accuracies for one single file are produced and displayed. In the first picture, every data point is considered. The second and more interesting picture displays only those data points which have achieved an accuracy that is higher than 70% or lower than 30%. In both cases, the algorithm is relatively sure that the item belongs either to Vorarlberg or Vienna. Those data points which have an accuracy e.g., of 53% should not be taken under consideration due to its uncertainty. The third approach is to change the decision theory, using a sequential decision making, which has been described in Section 2.1.5, based model. In this case, also only those data points which are higher than 70% or lower than 30% are considered. Furthermore, the calculation is not done by looking through all instances, but the

```python
import statistics

pred = []
means = []
index_pred = []
earliest_stop = round(len(acc)*0.2+0.49)
current_mean = 0

for i, value in enumerate(acc):
    pred.append(float(value))
    index_pred.append(i)

    cur_mean = statistics.mean(pred)
    means.append(current_mean)

    if i>=earliest_stop and current_mean>0.7:
        print ("Stop possible")
        print (current_mean)
        break

print (current_mean)
```

Program 7.4: Sequential validation of the best model.

decision making is done sequentially. The third picture shows this approach. The green horizontal line displays the decision boundary, which is, in this case, 80%.

For the first two approaches for every new instance, the accuracy has been calculated by summing all accuracies from the 0.4-second files divided by their quantity. The third approach looks at each new item at a time. In the beginning it takes in the first item with a chance of e.g., 70%. Next, it calculates the mean of all seen predictions. This results in a new *current mean*. The algorithm stops taking in new arguments when two conditions apply. The rules are that the program needs to look at at least 20% of the whole data and the accuracy needs to be at least 70%. If both conditions apply the program does not look at any further items and the current mean accuracy becomes the final one. Program 7.4 displays this calculation.

The final results are displayed in Table 7.6. All data points considered, the average accuracy amounts to 72.90%. In this case, the algorithm has more difficulties in the detection of the Viennese files than of the Vorarlberg ones. This does not change by considering a different amount of data points. However, the accuracy increases when considering only those data points, which achieved an accuracy over 70% or under 30%. In this instance, the accuracy came to 77.85%. Here, the performance of the Vorarlberg accuracy also excels the Viennese accuracy with 82.17% in comparison to 73.53%. Using the sequential decision making approach, the average validation accuracy amounts up to 77.46%. Here, too, the Vorarlberg perform better than the Viennese, with 81.66% in comparison to 73.27%. One explanation for the poor performance of the Viennese files is the background noise. In those examples, in which the result is disappointing, noise is present. Another possibility is the limitation of setting diversity in the training data.

	All Data Points	70/30 Data Points	70/30 Sequential
Vorarlberg 1	65.65%	73.28%	70.10%
Vorarlberg 2	75.73%	84.83%	82.12%
Vorarlberg 3	94.15%	95.36%	96.44%
Vorarlberg 4	60.31%	63.20%	63.20%
Vorarlberg 5	92.25%	94.17%	96.44%
Wien 1	66.76%	70.96%	70.60%
Wien 2	68.49%	74.07%	70.10%
Wien 3	72.52%	78.32%	76.86%
Wien 4	73.98%	81.73%	86.21%
Wien 5	59.29%	62.57%	62.57%
Vorarlberg acc	77.16%	82.17%	81.66%
Wien acc	68.21%	73.53%	73.27%
Avg acc	72.90%	77.85%	77.46%

Table 7.6: Validation results.

All things considered, the model does not perform as well on the validation set, as on the training data and test data, considering the standard decision making process. In general, all approaches have achieved promising results especially in the detection of the Vorarlberg class. The algorithm also works fine for the Viennese files, but some settings need to be given for achieving usable results. The most important factor is the loudness of the background noises. If those sounds are too present, the algorithm misclassifies the item in most cases.

Chapter 8

Conclusion

The result of this thesis proposes that Austrian dialect classification using machine learning based on audio features is possible and achieved similar performance as in other languages.

8.1 Challenges and Results

The motivation behind this research question was the limitations of possible interactions due to the lack of natural communication with computers, in particular with ASR-systems. Because those systems have trouble understanding speech by non-native speakers, the effort was made on classifying two Austrian dialects. Before investing deeper in the field of linguistic science, fundamental technical concepts have been explained. Machine learning in general, the various algorithms that can be used, general challenges in machine learning and feature extractions methods have been described. The choice of the selected classes, Vorarlberg and Vienna, is based on a linguistic perspective. Those two dialects have in consideration of history and characteristics the most differences. Furthermore, it has been investigated if different pronunciations from one man from Vorarlberg and one man from Vienna, display different feature representations.

After proving this theory, the first big challenge was to gather the appropriate data. In contrast to the English language, where several dialect databases exist, nothing alike was available for Austrian dialects. After several approaches have been carried out, a mixed approach was chosen. The Viennese audio data could be downloaded from the City of Vienna directly, containing interviews with people from society and politics describing their experiences with the city. The Vorarlberg material could be downloaded from the Literaturradio Vorarlberg and the Mundartradio. Both platforms share the same goal by presenting and archiving audio of the Vorarlberg dialect. Moreover, students from the University of Applied Sciences Upper Austria recorded themselves answering given questions.

The second big challenge is the number of possibilities of how to treat different volumes, quality, and length of the audio files. Therefore a lot of preprocessing and data cleaning steps have been inevitable. Because there were different steps needed, such as audio normalization, defining the window size, and choosing the right algorithm, hundreds of contrasting paths were possible. The decision on which possible solutions should have been implemented was challenging. Based on research the window size was defined to 0.2 seconds. The best model with this size then has been investigated how the performance changes by doubling the window size. For audio normalization, the choice needed to be made between three different methods. Based on research the EBU R128 standard was chosen. As a noise filter option, only one appropriate filter was available and its impact has been tested. For the feature extraction methods formants, RAW data and MFCC have been tried out on the first model. The MFCC approach was considered the most promising based on those results. In the end, five different paths have been implemented and compared.

The best approach, in this case, is those with the state choices from the large window size model. This means that the audio files were split into 0.4-seconds files, a median filter was applied, the features were extracted using MFCC, and the model was created using the CNN algorithm. This resulted in a training accuracy of 86.70%, 84.20% on the test set and 77.85% on the validation set.

8.2 Limits and Future Work

This thesis focuses on data preparation, data cleaning and data wrangling. Furthermore, the main part of these classification experiments deals with dialect classification based on spontaneous speech. Potential improvements and possible ongoing projects can be separated into two classes:

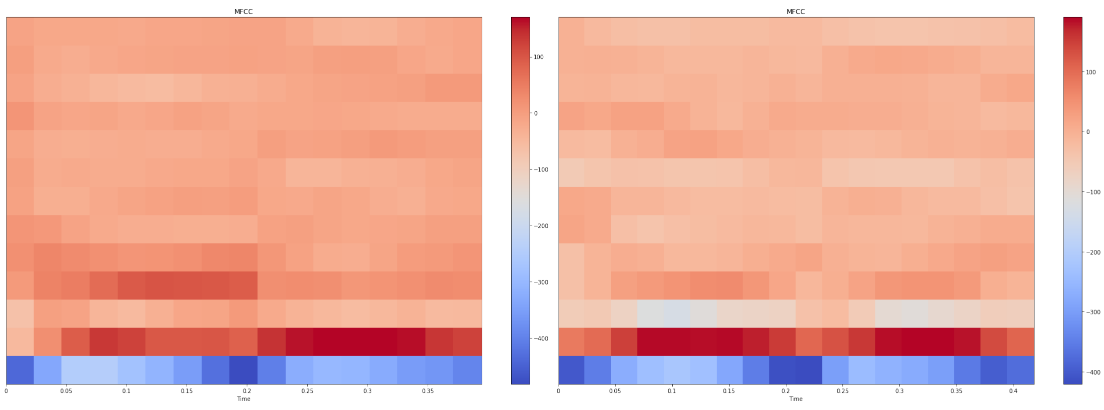
- increasing numbers,
- using the model as an input.

The first category can be applied to several parts along the project. The first aspect would be covering all nine federal states of Austria. Until now two dialects have been considered, due to the accompanying increasing complexity and time. Not only because of the increased states, but also in general more audio files could be downloaded and recorded. The states along the preprocessing pipeline could also be increased by adding more filters along the path. Here more possible solutions could be implemented. Currently, only a handful of those solutions have been tested. It is questionable if those choices have led to the optimal solution or if another path would have been better. Finally, the quantity of the algorithms could also change. At present, the SVM and CNN approach have been evaluated. As other researches have shown Random Forrest, HMM, GMM or LSTM could also be an option. Realistically speaking the most potential and valuable outcome lies probably in the increment of the dialects, the number of audio files and the preprocessing steps.

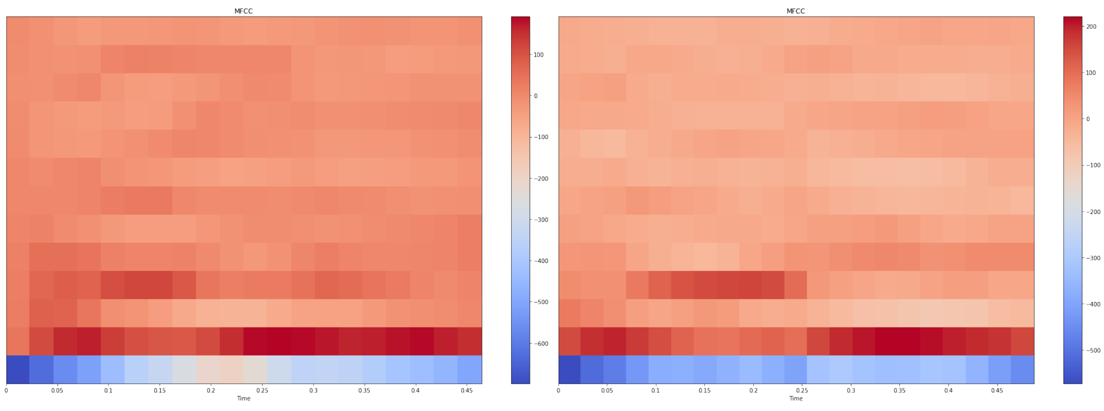
The second category contains three possible ways how the outcome of this thesis can be used as an input for further projects. The first idea would be trying to explain which sounds or overtones are decisive for the classification of dialects. Moreover, it would be a benefit of creating a live demo that classifies your dialect based on continuous speech. Users can interact with the system and become more aware and familiar with this topic. Lastly, it would be of interest to evaluate the impact on speech recognition performance in Austrian that would come from a specially trained ASR-System for each dialect.

Appendix A

Technical Details

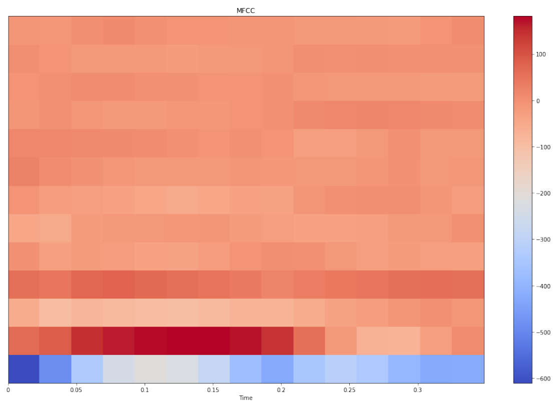


(a) (b)

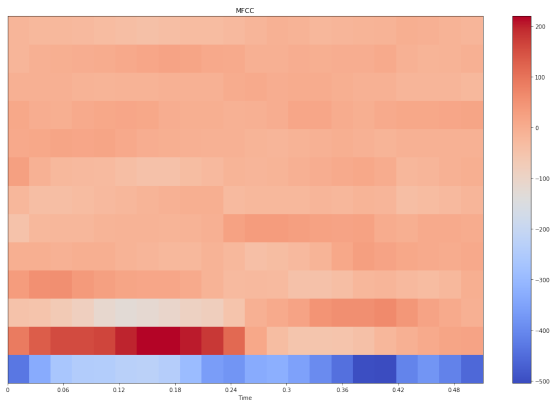


(c) (d)

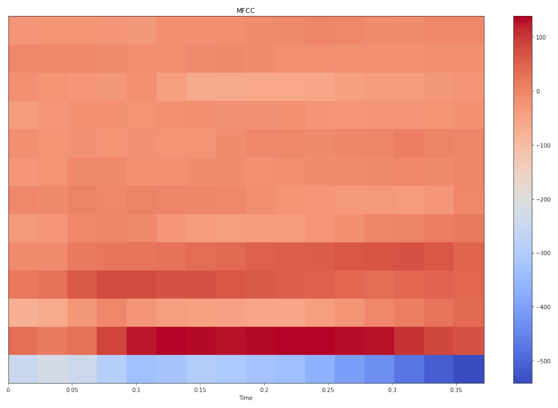
Figure A.1: Different pronunciation of words first part. *Vorarlberg Freitag* (a), *Vienna Freitag* (b), *Vorarlberg Nebel* (c), *Vienna Nebel* (d).



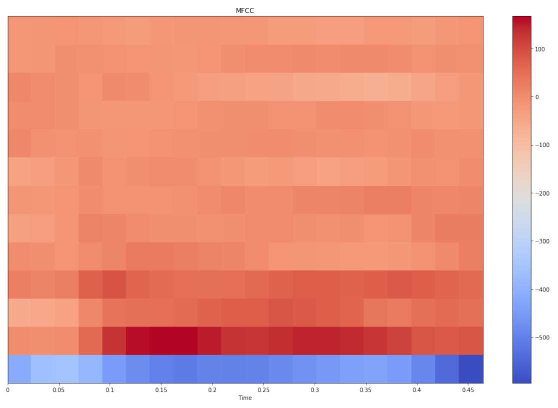
(a)



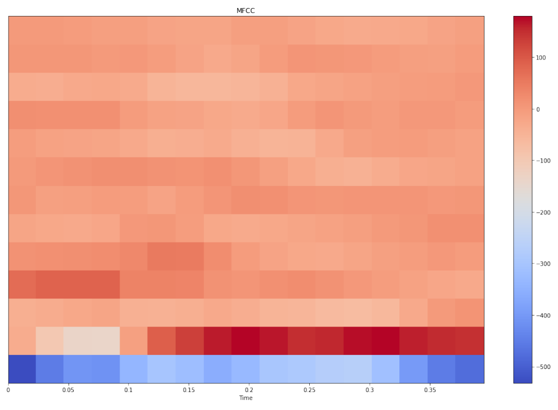
(b)



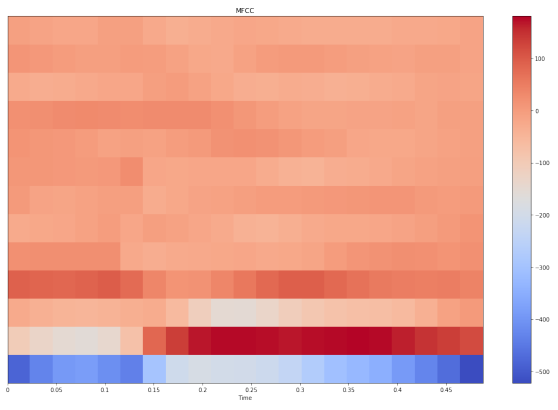
(c)



(d)

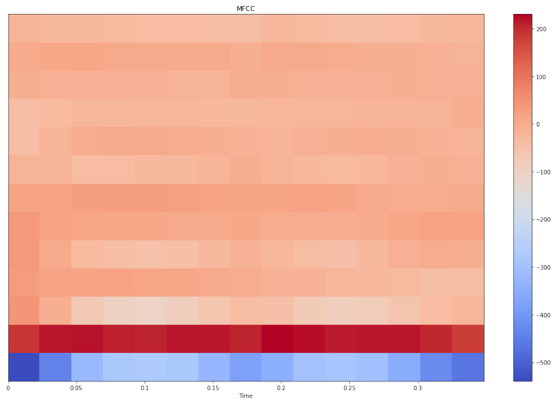


(e)

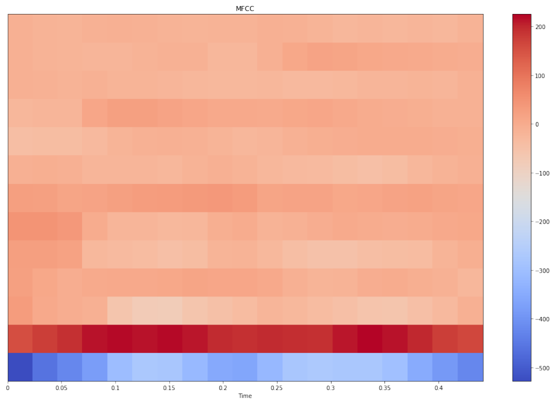


(f)

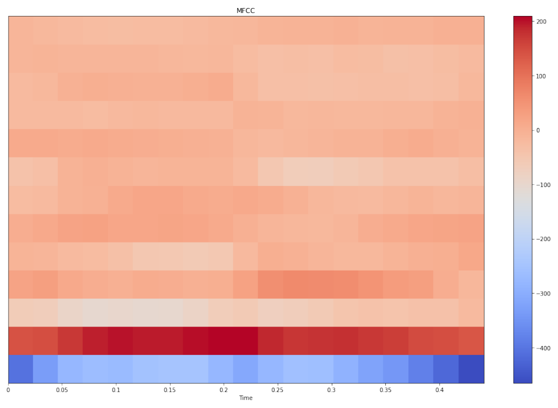
Figure A.2: Different pronunciation of words second part. *Vorarlberg Herbst* (a), *Vienna Herbst* (b), *Vorarlberg Knie* (c), *Vienna Knie* (d), *Vorarlberg Stein* (e), *Vienna Stein* (f).



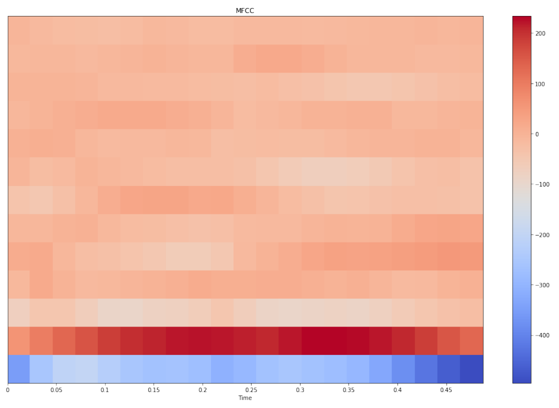
(a)



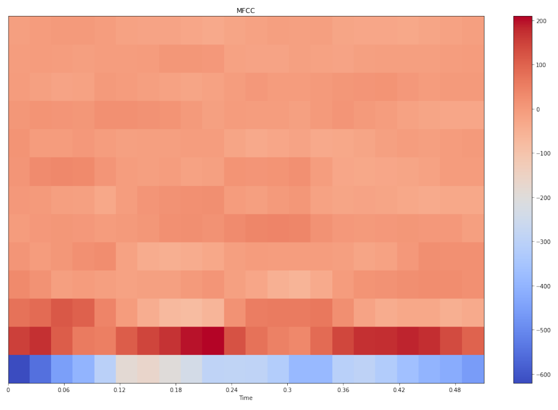
(b)



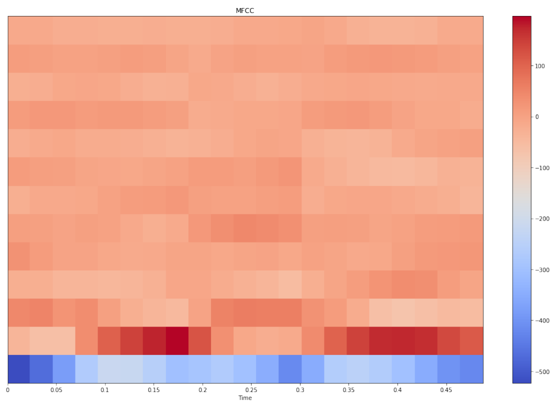
(c)



(d)

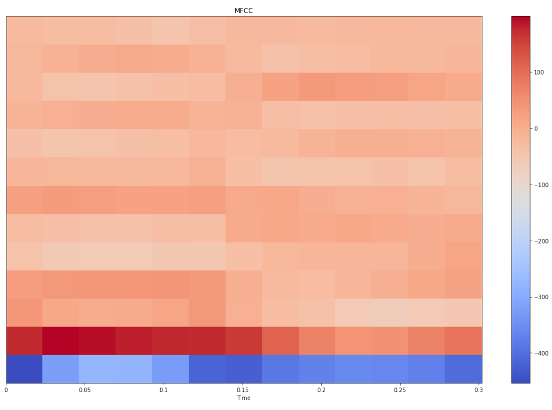


(e)

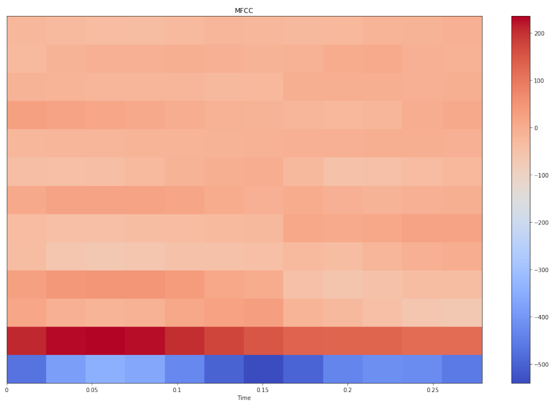


(f)

Figure A.3: Similar pronunciation of words. *Vorarlberg Mama* (a), *Vienna Mama* (b), *Vorarlberg Gabel* (c), *Vienna Gabel* (d), *Vorarlberg Samstag* (e), *Vienna Samstag* (f).



(a)



(b)

Figure A.4: Different pronunciation of the word, but similar MFCC. *Vorarlberg Bock* (a), *Vienna Bock* (b).

Appendix B

Data Gathering

Following questions and situations have been asked to describe in order to gather the necessary data. Those topics served as examples in order to get the most realistic accent possible:

- Beschreibe deinen Weg in die Arbeit.
- Was machst du in den ersten 10 Minuten nachdem du aufwachst?
- Was machst du in den letzten 10 Minuten vor dem Einschlafen?
- Beschreibe wie du dein Lieblingsrezept zubereitest.
- Wie bereitest du dich auf eine Prüfung vor?
- Wie ist deine Führerscheinprüfung abgelaufen?
- Was war die letzte Ausrede, warum du dich nicht mit jemandem treffen konntest?
- Beschreibe dein Auto.
- Beschreibe deine Wohnung.
- Erkläre warum dein Lieblingsfilm der Beste ist.
- Erkläre warum dein Lieblingsessen das Beste ist.
- Warum hast du dich für dieses Studium entschieden?
- Wenn du leben könntest wo du wolltest, wo wäre das und warum?
- Beschreibe deinen Arbeitsalltag wie du ihn in dir in 5 Jahren vorstellst.
- Wie schaut dein Traumurlaub aus?

Appendix C

Packages

Following packages and libraries have been used in order to accomplish the technical implementation:

```
    concurrent
    joblib
    keras
    librosa
    math
    matplotlib
    numpy
    os
    parslemouth
    pydub
    pysndfx
    python_speech_features
    scipy
    shutil
    sklearn
    soundfile
    subprocess
    tensorflow
```

Appendix D

CD-ROM Contents

D.1 PDF Files

Path: /

Wagner_Hanna_2019.pdf	Master thesis

D.2 Code Files

Path: /code/python

audioDownloadSplit.py	Split audio files into smaller pieces
convertMP3ToWav.py	Convert MP3 files to wav format
extractFeatures.py	Extract MFCC features
parslemouth.py	Extract Formants

Path: /code/jupyter

CNN.ipynb	CNN classification
dataPreparation.ipynb	Data preparation (filter etc.)
saveFeatures.ipynb	Save Features (MFCC, Formants etc.)
SVM.ipynb	SVM classification
validateOne.ipynb	Validate one new file

D.3 Feature Files

Path: /features

classes-10-test-0.4-MFCC.npy	Test classes of best model
features-10-test-0.4-MFCC.npy	Test features of best model
classes-10-train-0.4-MFCC.npy	Train classes of best model
features-10-train-0.4-MFCC.npy	Train features of best model

D.4 Model Files

Path: /model

10-CNN-best-MFCC.h5	H5 file of best model
10-CNN-best-MFCC.json	Json file of best model

D.5 Others

Path: /

README.md	Readme file
requirements.txt	Required packages

Bibliography

Print Resources

[1] F. Brandl. *Die Kunst der Stimmbildung auf physiologischer Grundlage.* München: Eigenverlag, 2001 (cit. on p. 15).

[2] S. Davis and P. Mermelstein. “Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences”. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 28.4 (Aug. 1980), pp. 357–366 (cit. on p. 12).

[3] S. Deshpande, S. Chikkerur, and V. Govindaraju. “Accent classification in speech”. In: *Proceedings of the fourth IEEE Workshop on Automatic Identification Advanced Technologies (AutoID’05).* Oct. 2005, pp. 139–143 (cit. on pp. 26, 27).

[4] Adele Diederich. “Decision and Choice: Sequential Decision Making”. In: *International Encyclopedia of the Social & Behavioral Sciences.* Ed. by James D. Wright. Second Edition. Oxford: Elsevier, 2015, pp. 906–910 (cit. on pp. 8, 9).

[5] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow. Concepts, tools, and techniques to build intelligent systems.* Sebastopol, CA: O’Reilly Media, Inc., 2017 (cit. on pp. 2, 6–13).

[6] John Karat et al. “Speech User Interface Evolution”. In: *Human Factors and Voice Interactive Systems.* Ed. by Daryle Gardner-Bonneau. Boston, MA: Springer US, 1999, pp. 1–35 (cit. on p. 2).

[7] Y. Lei and J. H. L. Hansen. “Dialect Classification via Text-Independent Training and Testing for Arabic, Spanish, and Chinese”. *IEEE Transactions on Audio, Speech, and Language Processing* 19.1 (Jan. 2011), pp. 85–96 (cit. on pp. 30, 31).

[8] “Median Filters”. In: *Encyclopedia of Microfluidics and Nanofluidics.* Ed. by Dongqing Li. Boston, MA: Springer US, 2008, pp. 1078–1078 (cit. on p. 16).

[9] R. Marchthaler and S. Dingler. *Kalman-Filter: Einführung in die Zustandsschätzung und ihre Anwendung für eingebettete Systeme.* Wiesbaden: Springer, 2017 (cit. on p. 17).

[10] John Paul Mueller and Luca Massaron. *Machine Learning for Dummies.* Hoboken, NJ: John Wiley & Sons, 2016 (cit. on p. 6).

[11] A. P. Nair, S. Krishnan, and Z. Saquib. “MFCC based noise reduction in ASR using Kalman filtering”. In: *Proceedings of the 2016 Conference on Advances in Signal Processing (CASP).* June 2016, pp. 474–478 (cit. on p. 17).

[12] Josh Patterson and Adam Gibson. *Deep learning. A practitioner's approach.* Sebastopol, CA: O'Reilly Media, Inc., 2017 (cit. on pp. 1, 6, 10, 11).

[13] Muhammad Rizwan and David V. Anderson. "A weighted accent classification using multiple words". *Neurocomputing* 277 (2018). Hierarchical Extreme Learning Machines, pp. 120–128 (cit. on pp. 2, 25–27).

[14] M.R. Schroeder, H. Quast, and H.W. Strube. *Computer Speech: Recognition, Compression, Synthesis.* Heidelberg: Springer, 2004 (cit. on p. 15).

[15] A.P. Sunija, T.M. Rajisha, and K.S. Riyas. "Comparative Study of Different Classifiers for Malayalam Dialect Recognition System". *Procedia Technology* 24 (2016), pp. 1080–1088 (cit. on pp. 28, 29).

[16] Charalambos Themistocleous. "Dialect classification using vowel acoustic parameters". *Speech Communication* 92 (2017), pp. 13–22 (cit. on pp. 16, 28).

[17] European Broadcasting Union. *R128. Loudness Normalisation and Permitted Maximum Level of Audio Signals.* Geneva, June 2014. URL: https://tech.ebu.ch/docs/r/r128.pdf (cit. on p. 19).

Online Resources

[18] Hack Audio. *Peak Normalization.* 2018. URL: https://www.hackaudio.com/digital-signal-processing/amplitude/peak-normalization/ (visited on 06/19/2019) (cit. on p. 19).

[19] Hack Audio. *RMS Amplitude.* 2018. URL: https://www.hackaudio.com/digital-signal-processing/amplitude/rms-amplitude/ (visited on 06/19/2019) (cit. on p. 19).

[20] Amit Paul Chowdhury. *Did Siri finally get an Indian accent too?* May 2017. URL: https://www.analyticsindiamag.com/siri-finally-get-indian-accent/ (visited on 06/19/2019) (cit. on p. 2).

[21] Albert Chu, Peter Lai, and Diana Le. *Accent Classification of Non-Native English Speakers.* 2017. URL: http://web.stanford.edu/class/cs224s/reports/Albert_Chu.pdf (visited on 06/19/2019) (cit. on pp. 29, 30).

[22] Sabine Erkinger-Kovanda. *Dialekte, AEIOU.* Apr. 2017. URL: https://austria-forum.org/af/AEIOU/Dialekte (visited on 06/19/2019) (cit. on pp. 21, 22).

[23] Haytham Fayek. *Speech Processing for Machine Learning. Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between.* Apr. 2016. URL: https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html (visited on 06/19/2019) (cit. on pp. 14, 15).

[24] Hasa. *Difference Between Accent and Dialect.* June 2016. URL: https://pediaa.com/difference-between-accent-and-dialect/ (visited on 06/19/2019) (cit. on p. 20).

[25] James Lyons. *Mel Frequency Cepstral Coefficient (MFCC) tutorial.* Jan. 2019. URL: http://www.practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/ (visited on 06/19/2019) (cit. on pp. 12–14).

[26] Leon Mak, An Sheng, and Mok Edmund Wei Xiong. *Deep Learning Approach to Accent Classification.* 2018. URL: http://cs229.stanford.edu/proj2017/final-reports/5244230.pdf (visited on 06/19/2019) (cit. on pp. 2, 25, 26, 37).

[27] Karl Rosaen. *K-fold cross-validation.* June 2016. URL: http://karlrosaen.com/ml/learning-log/2016-06-20/ (visited on 06/19/2019) (cit. on p. 7).

[28] Suesch. *Sprachen der Republik Österreich.* June 2018. URL: https://commons.wikimedia.org/w/index.php?curid=70219855 (visited on 06/19/2019) (cit. on p. 21).

[29] techopedia. *Noise.* 2019. URL: https://www.techopedia.com/definition/2025/noise (visited on 06/19/2019) (cit. on p. 16).

[30] Marco Varone, Daniel Mayer, and Andrea Melegari. *What is Machine Learning? A definition.* 2018. URL: http://www.expertsystem.com/machine-learning-definition/ (visited on 06/19/2019) (cit. on p. 5).