Accent Recognition System

Takudzwa Raisi

Thesis presented in fulfilment
of the requirements for the degree of
Bachelor of Science Honours
at the University of the Western Cape

Supervisor: Mehrdad Ghaziasgar
Co-supervisor: Reg Dodds

This version November 21, 2018

Declaration

I, Takudzwa Raisi, declare that this thesis "*Accent Recognition System*" is my own work, that it has not been submitted before for any degree or assessment at any other university, and that all the sources I have used or quoted have been indicated and acknowledged by means of complete references.

Signature: .... Date: ....

Takudzwa Raisi.

Abstract

Many South African languages have different speaking styles called accents or dialects. Identifying the accent is crucial in automatic speech recognition in order to improve speech recognition systems. This research aims to recognize different accents when English is spoken. When an Afrikaans speaking person or Xhosa speaking person, etc., speaks, the system then recognizes their accent.

Key words

Discrete Fourier transform

Fast-Fourier transform

Mel-frequency cepstrum

Mel-scale fitering

Principal component analysis

Support vector machine

Voice processing

Acknowledgment

First, I would like to express my sincere gratitude to my supervisor Dr Mehrdad Ghaziasgar and co-supervisor Reg Dodds for their continuous support of my honours research study. Their patience, motivation, and immense knowledge has played an vital role in my research and writing of this thesis. I could not have imagined having better supervisors and mentors for my honours degree. I would like to also give thanks to the National Research Fund for funding my BSc Honours and I am grateful to everyone who has given their time, effort and support to me throughout my years of studying.

Contents

List of Tables

List of Figures

Glossary

Discrete-Fourier transform (DFT) —the discrete-Fourier is a digital version of the continuous Fourier transform that uses data known at discrete intervals.

Fast-Fourier transform (FFT) —the Fourier transform is a function that fits continuous functions with a weighted series of sines and cosines. The fast-Fourier transform is an $O(n \log n)$ version of the Fourier transform.

Mel-frequency cepstrum (MFC —the Mel-frequency cepstrum represents the short-term power spectrum of a sound.

Mel-frequency cepstrum coefficients (MFCC —Mel-frequency cepstral coefficients collectively make up an MFC.

Principal Components —the principal components the biggest eigen values of of a feature space.

Support vector machine (SVM) —the support vector machine is used to separate exemplars into classes.

Chapter 1

Introduction

Accent recognition is an important aspect of speech recognition. Accent is a distinctive way of pronouncing a language and is often associated with a particular country, region or social class. There are two differences which exist between speakers: acoustic contracts that are related to the size and shape of the vocal tract and variations in pronunciation which are referred to as accent.

There are two speech research areas related to accent: *accent adaptation* through pronunciation modelling and *accent identification*. In South Africa we have 11 spoken languages with many different regional dialects. The way a Xhosa person pronounces certain words in English differs from how an Afrikaans speaking person pronounces the same words. Machine learning enables us to create systems that can recognize and deal with various accents. Deep learning based on neural nets and many other tested methods may be used.

1.1 Problem Statement

The development of computing speech recognition has advanced significantly (Barry et al., 1989) but problems such as accent identification are still under research. As humans, because of our backgrounds, ethnicity and country of origin, we tend to have different accents. Any dialect or language has its own accents and as a result a common language like English is spoken and enunciated differently. This has created one of the most common problems in speech recognition.

1.2 Project Requirements

This research requires people from different backgrounds and ethnicity who speak different languages and therefore have different dialects when talking in English. A microphone is used to record the subjects repeating some English sentences. These recordings are used to extract features to train our machine.

1.3 Prospective Solution

This research aims to solve this problem by designing a system that can identify and recognize accents. Once the system is implemented, a user should be able to speak certain words or a sentence in English through a microphone then the system is able to use the speech and to identify the accent. This is done using machine learning techniques to get features of different accents then training the system with the data.

There are also other means to coming up with a solution, by using hidden Markov model, support vector machine, etc. (Oh, 2014).
To extract features for accent we can use Mel-frequency cepstral coefficients with modelling components such as hidden Markov model, support vector machine, Gaussian mixture model and artificial neural network to recognize English accents.

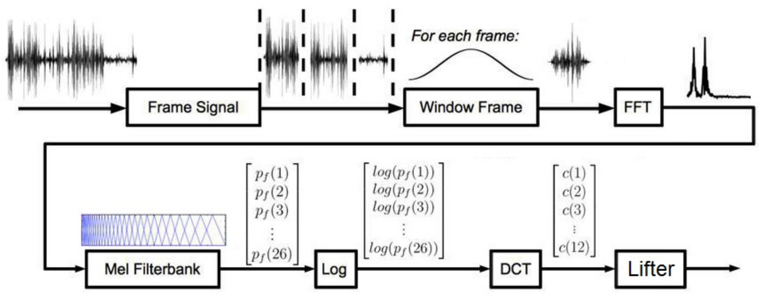

Figure 1.1: Mel-Frequency Cepstral Coefficients (Tomchuk, 2016)

1.3.1 Mel-Frequency Cepstral Coefficients

The Mel-frequency cepstral coefficient is broken into four parts, namely:

1. fast Fourier transform,

2. Mel-frequency spectrum,

3. discrete cosine transform,
4. logarithms of the signal.

Fast Fourier transform tells us more about time wavelength and breaks down the signal into frequencies. It converts a signal into individual spectral components and thereby provides frequency information about the signal. Figure 1.2 represents of an audio signal that has gone through FFT and the frequency information about the signal is depicted.

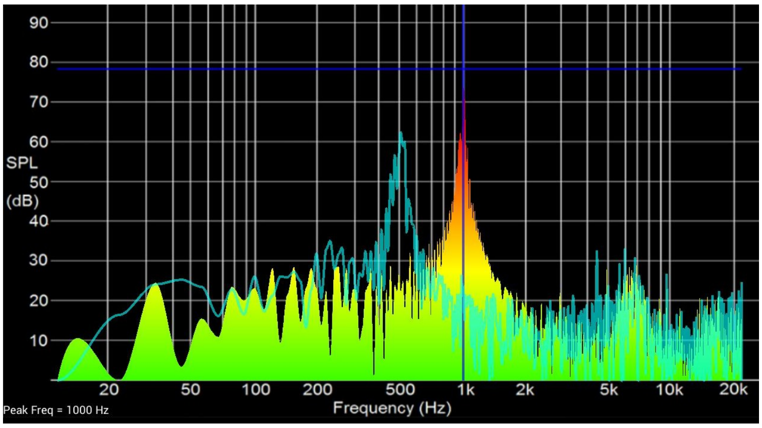

Figure 1.2: Fast Fourier Transform (Yang, 1990)

The Mel scale helps us in interpreting pitch. Our auditory system does not interpret pitch in a linear manner and to solve the Mel scale is applied using triangular bandpass filters.
We use discrete cosine transform that we get from the triangular bandpass filters. After this we implement the logarithm of the signal using logs.

1.4 Related Work

Much work has already been done in the field of automatic speech recognition with accent. 'IBM ViaVoice' is software implemented by IBM to recognize speech and sounds. The software consist of reading certain sentences and words in order to learn speech. The software then adapts itself to the specific users' way of saying certain words, their tone and sound and intonation features.
Huang et al. (2004) propose two different methods for accent adaptation. These methods were pronunciation dictionary modelling to reduce recognition

errors and Gaussian mixture model-based accent detection system was made for model selection.

Other interesting related work that has been done uses the Indian language, Telugu, which is widely spoken in Southern India. Telugu has many different accents and using techniques such as MFCC they were derived features to train their model and detect accent and speakers successfully (Mannepalli et al., 2015).

Other related work includes the 'English Dialects' which is an app that attempts to guess a user accent based on the pronunciation of 26 words. The app creates a heat map then tries to guess where the accent is from.

Humphries and Woodland (2012) proposed using decision trees to build A pronunciation dictionary for the recognition process. These trees are then used to cluster the measured pronunciation variations.

Chapter 2

System Design

2.1 System Interface

The proposed system interface has a simple design which that displays the system name **'Accent Recognition System'** which will consist of a **Record** button, **Predict** button, **Help** button and an **Exit** button and a display underneath to show the output of the program.

The **Record** button performs the action of recording the user's voice and then processing it. This process can be seen running on the command prompt.

The **Predict** button performs the action of predicting the accent after user speaks into the microphone.

The **Help** button gives the user a description about the system and how it works, also a manual and a set of commands of how the program can be run.

The **Exit** button exits the program after the user is finished using the system.

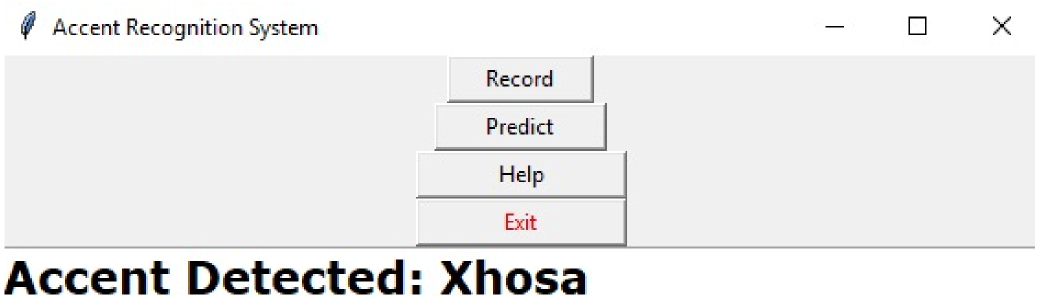

Figure 2.1: Graphical User Interface

2.1.1 Terminal Process

When running the algorithm, it is processed on the terminal and as a user's voice is recorded the terminal runs this process, in five different steps, namely:

- Recording

- Done Recording
- Extracting features
- Processing Accent
- Accent detected

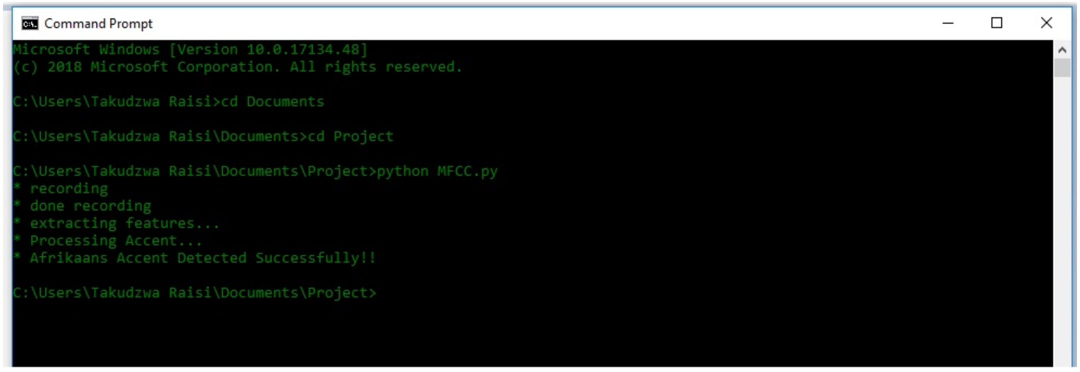

Figure 2.2: Terminal Process

2.2 High Level Design

The high-Level design shows the schematic representation of how the system will function. This design will show the step-by-step procedure of how the system achieves its goal of detecting the accent of the user.

The process starts with the user speaking into the microphone. At this stage the user will be requested to repeat a given sentence or phrase in English.

The audio is then processed and the MFCC is code is run. The `wav` file goes through several stages to turn it into a feature vector. We then use support vector machine for building our model, training and then testing. When this is done successfully, we will then be able to detect the user's particular accent.

2.3 Low-Level Design

Low-level design focuses mainly on the technical process of extracting features of a user's accent using MFCC. After the audio is received from the microphone, the fast Fourier transform algorithm is run (Yang, 1990).

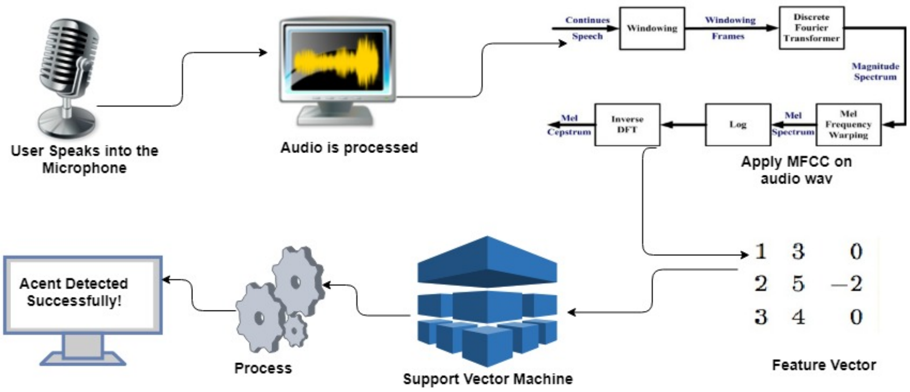

Figure 2.3: High Level Design

2.3.1 Fast Fourier Transform

This algorithm transfers the audio signal into its frequency domain. The FFT is a fast, algorithm to compute the *discrete fourier transform* (DFT). The DFT is given by

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N}.$$

After DFT we then proceed to the *inverse discrete fourier transform* (IDFT)

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{i2\pi kn/N}.$$

The transformation from $x_n \to X_k$ is a translation from configuration space to frequency space, and can be very useful in both exploring the power spectrum of a signal.

2.3.2 Mel Scale Filter Bank

Mel filter bank applies triangular filters,on a Mel-scale to the power spectrum to extract frequency bands. The Mel-scale aims to mimic the non-linear human ear perception of sound, by being more discriminating at lower frequencies and less discriminating at higher frequencies (Oh, 2014). The algorithm can be computed from this formula below:

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

This algorithm as pseudo code :

```
def melscale(X[])
    for i in length(X):
        X[i] = 2595*ln(1+X[i]/700)
    return X
```

Below is a representation of Mel-filter bank containing triangular filters.

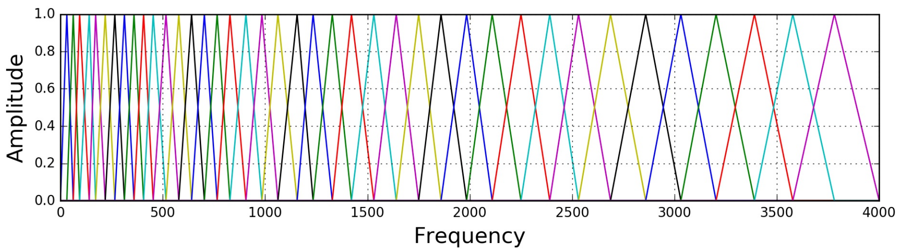

Figure 2.4: Filter bank on a Mel-Scale

2.3.3 Logarithm

This step takes the log of the powers at each Mel frequencies. It also serves to transform a multiplication into an addition as it is part of the computation of the cepstrum.

If we have a source signal x is convolved by some impulse response h. The resulting magnitude spectrum is:

$$|Y(\omega)| = |X(\omega)||H(\omega)|$$

Then after applying algorithm we get:

$$\log|Y(\omega)| = \log|X(\omega)| + \log|H(\omega)|$$

2.3.4 Discrete Cosine Transform

DCT is used for de-correlating speech data or the compression of most of the information in smaller number of coefficients and is given by

$$X_k = \frac{1}{2}(x_0 + (-1)^k x_{N-1}) + \sum_{n=1}^{N-2} x_n \cos\left[\frac{\pi}{N-1}nk\right], \text{ for } k = 0, \dots, N-1.$$

2.3.5 Feature Vector

After implementing MFCC and then running the algorithm on an audio file. The result will be feature vector. Below is a representation of this:

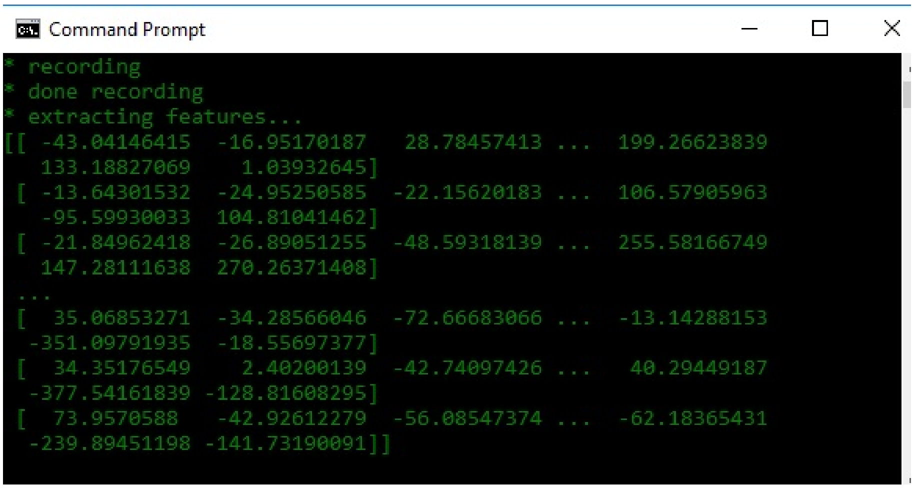

Figure 2.5: Vector Feature

Chapter 3

Implementation

3.1 Code Documentation

3.1.1 Fast Fourier Transform

The Fast Fourier transform converts a signal into individual spectral components and thereby provides frequency information about the signal. Below is an implementation of this in python

```
function fft(audio)
    rate,signal = scipy.io.wavfile.read(audio.wav)
    rate, signal = 44000, np.random.random((9218368,))
    data_length = len(audio)
    channel = np.zeros(2**(int(np.ceil(np.log2(data_length)))))
    channel[0:data_length] = signal
    fourier = np.fft.fft(channel)
```

3.1.2 MelScale Filtering

Mel filter bank applies triangular filters,on a Mel-scale to the power spectrum to extract frequency bands. The Mel-scale aims to mimic the non-linear human ear perception of sound, by being more discriminating at lower frequencies and less discriminating at higher frequencies. Below is pseudo code to do this:

```
def melscale(X[])
    for i in length(X):
        X[i] = 2595*ln(1+X[i]/700)
    return X
```

3.1.3 Logarithm

The logarithm serves to transform a multiplication into an addition. It is part of the computation of the cepstrum which take the logarithm of all filter bank energies. Below is a pseudo code of this:

```
function Log(Arr[]):
    newArr[]
    for i in length(Arr):
        newArr[i] = log(Arr[i])
    return newArr
```

3.1.4 Discrete cosine transform

Thw DCT can be thought as a compression step, it is used for de-correlating speech data or the compression of most of the information in smaller number of coefficients. Below is the pseudo code of this:

```
function DCT(newArr[]):
    dctarray[]
    k = length of newArr
    for (i in k-1):
        for (j in k):
            for (j in k):
                hold = hold + newArr[j]*cos(k*(2*j-1)*pi/2*k)
            dctarray = hold
            hold = 0
    return newArr
```

3.1.5 Feature Vectorization

Vectorization takes the data extracted from audio files using MFCC and convert the data into a 3D format. This data becomes our features which will be used for training and testing. The pseudo code can be seen below:

```
function vectorize(dctarray):
    vecarray = zeros(length(dctarray) + 2)
    for (i = 1 in length(vecarray) - 1):
        vecarray[i] = dctarray[i - 1]
```

```
    delta[]
    k = 2
    for (i in length(dctarr)):
        delta[i] = fill[j] fill[i]
        j++
    populatedelta = zeros(length(delta) + 2)
    for (i = 1 in length(delta) - 1):
        populatedelta[i] = delta[i-1]
    deltatwo[]
    j = 2
    for (i in length(delta)):
‘ deltatwo[i] = populatedelta[j] - populatedelta[i]
        j++
    featVec[][][]
    for (i in length(dctarray)):
        featVec[i][i][i] = dctarray[i], delta[i], deltatwo[i]
    return featVec
```

3.1.6 Description of Data Collected

In this research the focus is on three South African accents, namely Xhosa-English accent, Afrikaans-English accent and English-English accent. For each of these three accents five different sentences were collected for each of five speakers

1. “Coding teaches you how to think”,
2. “Life is full of adventures”,
3. “Yesterday you said tomorrow”,
4. “I love music a lot”, and
5. “Cape Town is a beautiful city”.

Each of five subjects recorded 10 samples of each sentence making 50 audio `wav` files for each speaker. In total the data set consists of 250 `wav` files for each of the three different accents giving a total of 750 recordings.

3.1.7 Training

The features extracted from the audio files are used for training. The recognition process is done through a support vector machine which is a machine learning module with associated learning algorithms that analyse data used for classification and regression analysis (Yang, 1990).

In order to achieve training, the feature set for each accent should have a y-label to represent it in the SVM model. English-English accent is labelled by "1", Xhosa-English is labelled "2" and Afrikaans-English is labelled by "3". Figure 3.1 is a representation of features randomized with labels for training. After the features are stored in the prescribed format for the `train()`

```
3.0   1:5.569528264641479 2:5.079998386229159 3:5.078237667488721 4:5.
3.0   1:5.4071699463386445 2:3.3263025309229755 3:6.52997463782761 4:7
3.0   1:6.130668895401391 2:5.537732899865538 3:4.728006640097908 4:6.
1.0   1:0.5368900498610927 2:0.09588284951990138 3:0.19152357274681556
1.0   1:5.940725325199447 2:5.527652465202837 3:5.425600072013739 4:3.
3.0   1:6.938864924997355 2:5.268005940375446 3:5.107884053196935 4:4.
2.0   1:4.484389254776994 2:-0.3355811719304151 3:3.106119670941961 4:
3.0   1:3.084090391831905 2:-0.2797564311519444 3:2.701870214102499 4:
3.0   1:6.8194781082985 2:4.458903537873645 3:5.8535392114815945 4:6.5
1.0   1:4.903801824153416 2:3.2386999101452654 3:5.143570968056572 4:5
```

Figure 3.1: Labelled Features

procedure they are used for training. Below is a part of pseudo code to perform classification training.

```
arrayX = [features]
arrayY = [1, 3, 2, 1, 3, 2, 1, ....]
X_train, Y_train = model_selection.train(arrayX,arrayY)
```

After training has been done the SVM creates a training model and we use testing file created to form a prediction model which then is able to count the number of hits and misses to give the accuracy as a percentage.

3.2 Libraries Used

The following libraries are used:

1. `import scipy.io.wavfile` as `wav`—For reading the `wav` audio files.

2. `from python_speech_features import mfcc`—Provides common speech features for ASR including MFCCs and filter bank energies.

3. `from sklearn import model_selection` —This helps us in model selection.

4. `from sklearn.model_selection import train_test_split`—Used for training and testing using `LibSVM`.

Chapter 4

Testing and Optimization

4.1 Testing

4.1.1 Overview

In order to achieve testing results and predictions, the system is first trained on the data set to produce a model using the procedure explained in Chapter 3. Once the model is created, the test set data is fed into the model using the `predict()` function. The predictions made by the model can then be used to determine the accuracy to see how the model behaves and how precisely accents can be recognized.

4.1.2 Data set

The data set contains of 750 audio clips taken in the same room with the same microphone. These audio clips consist of three accents, five subjects, five sentences and ten samples for each of these. Figure 4.2 shows these details.

Table 4.1: Data set arrangement

Accent	Subjects	Sentences	Samples
English	5	5	10
Xhosa	5	5	10
Afrikaans	5	5	10

4.1.3 Training

In this research we had four questions to answer to test *how well we can predict accent*, so the training was split into four parts. These questions were:

1. If subject is seen and sentence known but ignoring some samples, how well can we predict accent.

2. If subject is seen but sentence is unseen, how well can we predict accent.

3. If subject is unseen but sentence is seen, how well can we predict accent.

4. If subject is unseen and sentence is unseen, how well can we recognize accent.

Before training, it is key to scale the features so that we can standardize the range of independent variables or features of data. After this has been done the next phase was to build the features into the prescribed `LibSVM` format. Below is the python code to achieve this.

```python
def custom_dump_svmlight_file(X_train,Y_train,filename):
    featinds = [" " + str(i) + ":" for i in range(1,len(X_train[0])+1)]
    with open(filename, 'w') as f:
        for ind, row in enumerate(X_train):
            f.write(str(Y_train[ind]) + " " + "".join([x for x in itertools.chain.from_iterable(zip(featinds,map(str,row))) if x]) + "\n")
```

The next step is optimization which is paramount to get the best results from our model and to improve the performance of the system. To optimize the SVM cross validation is performed to find the best Cost and Gamma parameter. For each question we had to perform cross validation and train to produce a prediction model.

4.1.4 Testing

The scaled features and normalized data are then used to test the accuracy of the prediction model. In order to achieve testing and training the pseudo-code below was used to predict using the model.
The `clf.fit()` function fits the x-values and the y-values of our training to the model.
The `predict()` function is then used to test the correctness of the model using our testing values.
The `load()` function loads the prediction model produced by training.
The label that is produced by the model predicts the accent when a user speaks into the microphone.

```
import SVM

clf.fit(X_train,Y-train)
modelPrediction = clf.predict(X_test)
print("The model accuracy is:accuracy_score(Y_test,modelPrediction))

clf = joblib.load('all.model')
label = clf.predict([fbank_feat])
print(["English", "Xhosa", "Afrikaans"][label - 1] + " accent
    detected")
```

For each question training and testing was done differently. Below is a table showing how it was broken into segments . In Question1 five samples were used to train and to test, then in Question2 four sentences were used to train and one was used to test, then Question3 three of the subjects were used for training and two subjects for testing and finally in Question4 we used four subjects to train and four sentences to train thereby completely removing one person per accent for testing. Below is a table showing how it was broken into segments.

Table 4.2: Training and Testing

Question 1	Subjects	Sentences	Samples
Train	5	5	5
Test	5	5	5
Question 2	Subjects	Sentences	Samples
Train	5	4	10
Test	5	1	10
Question 3	Subjects	Sentences	Samples
Train	3	5	10
Test	2	5	10
Question 4	Subjects	Sentences	Samples
Train	4	4	10
Test	1	1	10

4.1.5 Evaluation

4.1.6 Results

The results from the testing for each question showed satisfactory results.

Table 4.3: Test Scores

Question	Accuracy	Precision	Recall	F1 score
1	93.6%	94.39%	93.6%	93.67%
2	91.3%	91.66%	91.33%	91.36%
3	86%	86.59%	85.67%	85.7%
4	63%	69%	63%	61%

The overall accuracy of Question 1–3 was 90% and for Question 4 the aim was to test whether if we would correctly predict the accent of an unknown subject that would speak an unknown sentence. Therefore analyzing the results the classifier has done better than expected as random guessing the classifier would have predicted 33% but it has managed to produce double the expected results.

4.1.7 Further Testing using Four Accents

To further test and verify our system we added a 'Cape Coloured' accent which is an accent spoken by so-called coloured people from Cape Town. This then increased the accents used to four, namely English-English accent, Xhosa-English accent, Afrikaans-English accent and Cape-Colored-English accent. The results obtained after this testing were very good and satisfactory.

Table 4.4: Test Scores using four accents

Question	Accuracy	Precision	Recall	F1 score
1	91.6%	92%	91.6%	91.66%
2	89.5%	89.8%	89.5%	89.48%
3	85%	85.78%	85%	85%
4	71%	75.25%	71%	70%

4.2 Conclusion

The support vector machine proved to be a perfect classifier for this project and the Mel-frequency cepstral coefficients technique was suitable for the accent recognition problem and solving it. The overall results for this project were strong and it was also able to meet its expected requirements and therefore it was a success.

Bibliography

Barry, W. J., Hoequist, C. E., and Nolan, F. J. (1989). An approach to the problem of regional accent in automatic speech recognition. *Computer Speech and Language*, 3(4):355–366.

Huang, C., Chang, E., and Chen, T. (2004). Accent issues in large vocabulary continuous speech recognition. *Speech Technology*, 7(23):141–153.

Humphries, J. J. and Woodland, P. C. (2012). Using accent-specific pronunciation modelling for improved large vocabulary continuous speech recognition. *Speech Modelling*, 5(3):155–187.

Mannepalli, K., Sastry, P. N., and V.Rajesh (2015). Accent detection of Telugu speech using supra-segmental features. *International Journal of Soft Computing*, 10(5):287–292.

Oh, S. (2014). Bayesian method recognition rates improvement using HMM vocabulary recognition model optimization. *Digital Convergence*, 12(7):273–278.

Tomchuk, K. K. (2016). Frequency masking in speech MFCC-parameterization in presence of noise. *Information and Control Systems*, 82(3):8–14.

Yang, D. (1990). Fast discrete radon transform and 2-D discrete Fourier transform. *Electronics Letters*, 26(8):550–551.

Appendix A

System Manual

A.1 Description

This manual provides the instructions of installations and functions for the Accent Recognition Program. It also instructs on configuration other external component for suitable accuracy and performance of the system.

A.2 System Requirements

The following requirements are to be installed for the Accent Recognition program to operate.

Table A.1: Requirements

Type	**Name**	**Recommended**
Environment	Python 3	Python 3.6.4
OS	Linux or MS Windows	MSWin 10 or 64bit Ubuntu 16.04
Arithmetic library	NumPy and SciPy	N/A
SVM library	Sklearn or Libsvm	N/A
Audio library	Pyaudio or Libsvm	N/A
Microphone	Studio Condenser Mic	C01U PRO Samson

Note: If the microphone that will be used requires a driver, please make sure the driver is functional.

A.3 Installations

All software installations are done through the terminal. Make sure there is sudo rights for clearance. If using ubuntu open a new terminal and execute the following:

```
sudo apt-get install python3
sudo apt-get install python3-numpy
sudo apt-get install python3-scipy
```

```
sudo apt-get install python3-pyaudio
sudo apt-get install python3-sklearn
```

If using windows first download python and install it then open the command prompt and execute the following:

```
python -m pip install numpy
python -m pip install scipy
python -m pip install pyaudio
python -m pip install sklearn
python -m pip install libsvmn
```

Once the user runs the program and speaks into the microphone and press predict the graphical user interface will display the accent, here is an example:

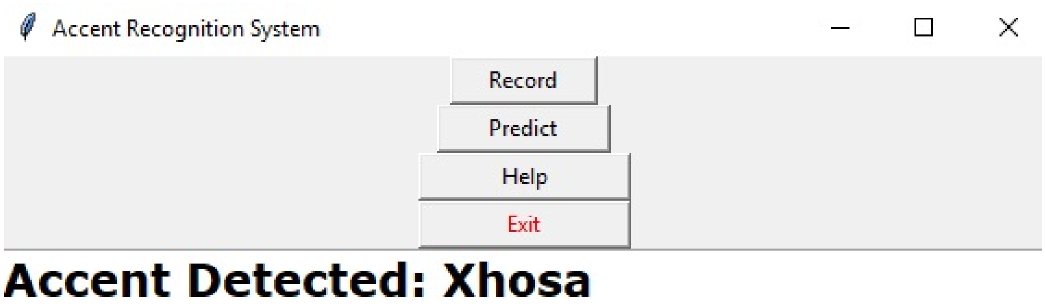

Figure A.1: Graphical User Interface

A.4 Functions

The main functions of the system are:

Table A.2: Main functions

Record	Performs the action of recording the user's voice and then processing it
Predict	Performs the action of predicting the accent after user speaks into the microphone.
Help	Gives the user a description about the system and how it works.
Exit	Exits the program after the user is finished using the system.

Appendix B

Code Listings

B.1 Description

This chapter includes all the source code used for implementing the Accent Recognition System.

B.2 Feature Extraction, Training and Testing

```
from python_speech_features import mfcc
from python_speech_features import delta
from python_speech_features import logfbank
import scipy.io.wavfile as wav
import scipy.interpolate as interpol
import glob
import wave
import pickle
import sklearn
import scipy.io.wavfile as wav
import csv
import os
import array
import re
import numpy as np
import itertools
from time import time
from sklearn.svm import SVC
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.externals import joblib
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.svm import libsvm
import subprocess
from sklearn import metrics
from subprocess import Popen
from sklearn import preprocessing
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.metrics import accuracy_score
from matplotlib import pyplot as plt
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import average_precision_score

```

```

gnuplot_exe = r"C:\Users\Takudzwa Raisi\Desktop\accent_recognition\gnuplot\binary\gnuplot.exe"
grid_py = r"C:\Users\Takudzwa Raisi\Desktop\accent_recognition\libsvm-3.22\tools\grid.py"
svmtrain_exe = r"C:\Users\Takudzwa Raisi\Desktop\accent_recognition\libsvm-3.17-GPU_x64-v1.2\windows\svm-train-gpu.exe"
svmpredict_exe = r"C:\Users\Takudzwa Raisi\Desktop\accent_recognition\libsvm-3.17-GPU_x64-v1.2\windows\svm-predict.exe"

def paramsfromexternalgridsearch(filename, crange, grange, printlines=False):
  #printlines specifies whether or not the function should print every line of the grid search verbosely
  cmd = 'python "{0}" -log2c {1} -log2g {2} -svmtrain "{3}" -gnuplot "{4}" "{5}"'.format(grid_py, crange, grange, svmtrain_exe, gnuplot_exe, filename)
  f = Popen(cmd, shell = True, stdout = subprocess.PIPE).stdout

  line = ''
  while True:
    last_line = line
    line = f.readline()
    if not line: break
    if printlines: print(line)
  c,g,rate = map(float,last_line.split())
  return c,g,rate

def accuracyfromexternalpredict(scaled_test_file, model_file, predict_test_file, predict_output_file):
  cmd = '"{0}" "{1}" "{2}" "{3}"'.format(svmpredict_exe, scaled_test_file, model_file, predict_test_file)
  f = Popen(cmd, shell = True, stdout = subprocess.PIPE).stdout
  #f = subprocess.Popen(cmd, shell = True, stderr=subprocess.STDOUT, stdout=subprocess.PIPE)

  line = ''
  while True:
    last_line = line
    line = f.readline()
    if not line: break

  return last_line.split(" ")[3][1:-1].split("/")[0], last_line.split(" ")[3][1:-1].split("/")[1]

def normalize(inSig,outLen):
  #This function normalizes the audio signal.
  #It first produces an interp1d structure that readily interpolates between points
  #Then it sets the size of the space to outLen=200000 points, and interp1d interpolates to fill in gaps
  #In essence, it takes every audio signal and produces a signal with outLen=200000 data points in it = normalization
  inSig = np.array(inSig)
  arrInterpol = interpol.interp1d(np.arange(inSig.size),inSig)
  arrOut = arrInterpol(np.linspace(0,inSig.size-1,outLen))
  return arrOut


def writetopcklfile(outpath, data):
  with open(outpath, 'wb') as f:
    pickle.dump(data, f)

def readfrompcklfile(outpath):
    with open(outpath, 'rb') as f:
      return pickle.load(f)

```

```
def custom_dump_svmlight_file(X_train,Y_train,filename):
  #This function inserts the extracted features in the libsvm format
  featinds = [" " + str(i) + ":" for i in range(1, len(X_train[0])+1)]
  with open(filename, 'w') as f:
    for ind, row in enumerate(X_train):
      f.write(str(Y_train[ind]) + " " + "".join([x for x in itertools.chain.
          from_iterable(zip(featinds,map(str,row))) if x]) + "\n")

def main():

  start = time()

  path =r'accents'
  files = os.listdir(path)

  features = []
  label = []
  Filenames = {}
  X =[]
  Xdir = {}
  y =[]
  for filename in glob.glob(os.path.join(path, '*.wav')):
    Filenames[filename[filename.find('\\')+1:]] = filename


  for sounddata in Filenames:
    (rate,sig) = wav.read(path+'/'+sounddata)

    # rate = sampling rate, sig = data; the data will
    # be a two-tuple array format where the first item
    # of each row will be the left channel data, and
    # the second item will be the right channel data
    # This code below is used to extract features
    # from audio samples using MFCC
    newSig = []
    for i in range(len(sig)):newSig.append(sig[i][0])
    newSig = normalize(newSig,200000)
    rate = newSig.shape[0]/sig.shape[0]*rate
    mfcc_feat = mfcc(newSig,rate)#,nfft=)
    d_mfcc_feat = delta(mfcc_feat, 1)
    fbank_feat = logfbank(newSig,rate)
    fbank_feat = fbank_feat.ravel()
    fbank_feat = normalize(fbank_feat,11778) #Normalize features
    features.append((normalize(fbank_feat.ravel(),11778),sounddata[:1]))
    Xdir[sounddata.replace(".wav", "")] = len(features) - 1
    X.append(fbank_feat)

  y.append(sounddata[:1])
  finish = time()
  print("Time to load data %.3f s" % (finish - start))


  #------------------------------------------------
  #    1. If subject is seen and sentence known
  #       but taking some samples out
  #------------------------------------------------
  # 3 accents x 5 subjects x 5 sentences x 10 samples
  # First question: sub known, sent known
  print("Computing Question 1: If subject is seen and sentence known but
      taking some samples out")
  totalaccents = 4
  subsperaccent = 5
  totalsents = 5
  totalsamples = 10
  trainsamples = 5


```

```python
testsamples = totalsamples - trainsamples #Don't change
#Accent,Subject,Sentence,Sample
X_train = []
X_test= []
Y_train = []
Y_test = []


for acc in range(1,totalaccents+1):
  for sub in range(1,subsperaccent+1):
    for sent in range(1,totalsents+1):
      for samp in range(1,totalsamples+1):

        filename = ",".join([str(it) for it in [acc,sub,sent,samp]])

        if samp <= trainsamples :

          X_train.append(X[Xdir[filename]]) #Features of audio sample 1-5
          Y_train.append(int(filename[0])) #labels (1,2,3,4)

        else:     #Otherwise test on the remaining

          X_test.append(X[Xdir[filename]]) #Features of audio sample 6-10
          Y_test.append(int(filename[0]))


#feature scaling in order to standardize the features
scaler = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

#Create a training file that will contain the training data
trainfile = "Question1.dat"
#Cross validation in order to get the best C and Gamma parameter
custom_dump_svmlight_file(X_train, Y_train, trainfile)
crange = "-5,13,2"#"1,5,2"
grange = "-15,5,2"#"-3,2,2"

C,gamma,cvrate = paramsfromexternalgridsearch(trainfile, crange, grange,
    printlines=True)
#for 2 accents: best was C=2**3, gamma=2**-15
clf = SVC(gamma=gamma,C=C, kernel="rbf")
clf.fit(X_train,Y_train)

#Passing in the test samples against the created model
modelPrediction = clf.predict(X_test)
print("The model accuracy is:",metrics.accuracy_score(Y_test,
    modelPrediction)*100,"%")
print("precision scores")
print("Macro: ",precision_score(Y_test, modelPrediction, average='macro')
    *100,"%")
print("recall scores")
print("Macro: ",recall_score(Y_test, modelPrediction, average='macro')
    *100,"%")
print("f1 scores")
print("Macro: ",f1_score(Y_test, modelPrediction, average='macro')*100,"%
    ")


writetopcklfile("Question1.model",clf) #Writing the model to a picklefile

finish = time()
print("Time to compute Q1 %.3f s" % (finish - start))
#-----------------------------------------------
#    2.If subject is seen but sentence is unseen
#-----------------------------------------------
print("Computing Question 2: If subject is seen but Sentence is unseen")
```

```
totalaccents = 4
subsperaccent = 5
totalsents = 5
train_sentence = 4
totalsamples = 10

testsentence= totalsents - train_sentence #Don't change
#Accent,Subject,Sentence,Sample
X_train2 = []
X_test2 = []
Y_train2 = []
Y_test2 = []

for acc in range(1,totalaccents+1):
  for sub in range(1,subsperaccent+1):
    for sent in range(1,totalsents+1):
      for samp in range(1,totalsamples+1):

        filename = ",".join([str(it) for it in [acc,sub,sent,samp]])

        if sent <= train_sentence:

          X_train2.append(X[Xdir[filename]]) #Features of audio sample 1-5

          Y_train2.append(filename[0]) #labels (1,2,3,4)

        else:     #Otherwise we test on the remaining

          X_test2.append(X[Xdir[filename]]) #Features of audio sample 6-10

          Y_test2.append(filename[0])

#feature scaling in order to standardize the features
scaler = StandardScaler().fit(X_train2)
X_train2 = scaler.transform(X_train2)
X_test2 = scaler.transform(X_test2)


#Creating a training file that will contain the training data
trainfile = "Question2.dat"

custom_dump_svmlight_file(X_train2, Y_train2, trainfile)

#Cross validation in order to get the best C and Gamma parameter
crange = "-5,13,2" #"1,5,2"
grange = "-15,5,2" #"-3,2,2"
C,gamma,cvrate = paramsfromexternalgridsearch(trainfile, crange, grange,
    printlines=True)
clf = SVC(gamma=gamma,C=C, kernel="rbf")
clf.fit(X_train2,Y_train2)

#Passing in the test samples against the created model
modelPrediction2 = clf.predict(X_test2)
print("The model accuracy is:",metrics.accuracy_score(Y_test2,
    modelPrediction2)*100,"%")
print("precision scores")
print("Macro: ",precision_score(Y_test2, modelPrediction2, average='macro
    ')*100,"%")
print("recall scores")
print("Macro: ",recall_score(Y_test2, modelPrediction2, average='macro')
    *100,"%")
print("f1 scores")
print("Macro: ",f1_score(Y_test2, modelPrediction2, average='macro')*100,
    "%")

writetopcklfile("Question2.model",clf) #Writing the model to a picklefile
```

```python
#-------------------------------------------------
#    3.If subject is unseen but sentence is seen
#-------------------------------------------------
print("Computing Question 3: If subject is unseen but sentence is seen")
totalaccents = 4
subsperaccent = 5
totalsents = 5
train_subjects = 3
totalsamples = 10

testsubject= subsperaccent - train_subjects #Don't change
#Accent,Subject,Sentence,Sample
X_train3 = []
X_test3 = []

Y_train3 = []
Y_test3 = []

for acc in range(1,totalaccents+1):
  for sub in range(1,subsperaccent+1):
    for sent in range(1,totalsents+1):
      for samp in range(1,totalsamples+1):

        filename = ",".join([str(it) for it in [acc,sub,sent,samp]])

        if sent <= train_subjects:

          X_train3.append(X[Xdir[filename]]) #Features of subjects 1-3

          Y_train3.append(filename[0]) #labels (1,2,3,4)

        else:     #Otherwise we test on the remaining

          X_test3.append(X[Xdir[filename]]) #Features of subjects 3-5
          Y_test3.append(filename[0])


#feature scaling in order to standardize the features
scaler = StandardScaler().fit(X_train3)
X_train3 = scaler.transform(X_train3)
X_test3 = scaler.transform(X_test3)

#Creating a training file that will contain the training data
trainfile = "Question3.dat"

custom_dump_svmlight_file(X_train3, Y_train3, trainfile)
#Cross validation in order to get the best C and Gamma parameter
crange = "-5,13,2"#"1,5,2"
grange = "-15,5,2"#"-3,2,2"
C,gamma,cvrate = paramsfromexternalgridsearch(trainfile, crange, grange,
    printlines=True)

clf = SVC(gamma=gamma,C=C, kernel="rbf")
clf.fit(X_train3,Y_train3)
#Passing in the test samples against the created model
modelPrediction3 = clf.predict(X_test3)
print("The model accuracy is:",metrics.accuracy_score(Y_test3,
    modelPrediction3)*100,"%")
print("precision scores")
print("Macro: ",precision_score(Y_test3, modelPrediction3, average='macro
    ')*100,"%")
print("recall scores")
print("Macro: ",recall_score(Y_test3, modelPrediction3, average='macro')
    *100,"%")
print("f1 scores")
print("Macro: ",f1_score(Y_test3, modelPrediction3, average='macro')*100,
    "%")
```

```
#writetopcklfile("try3.model",clf)


#-------------------------------------------------
#    4.If subject is unseen but sentence is unseen
#-------------------------------------------------
print("Computing Question 4: subject is unseen but sentence is unseen")
totalaccents = 4
subsperaccent = 5
totalsents = 5
train_sentence = 4
train_subjects = 4
totalsamples = 10

testsubject= subsperaccent - train_subjects
test_sentence = totalsents - train_sentence #Don't change
#Accent,Subject,Sentence,Sample
X_train4 = []
X_test4 = []
Y_train4 = []
Y_test4 = []


for acc in range(1,totalaccents+1):
  for sub in range(1,subsperaccent+1):
    for sent in range(1,totalsents+1):
      for samp in range(1,totalsamples+1):

        filename = ",".join([str(it) for it in [acc,sub,sent,samp]])

        if sub <= train_subjects and sent <= train_sentence:

          X_train4.append(X[Xdir[filename]])
          #X_train_labels3.append(filename)
          Y_train4.append(filename[0]) #labels (1,2,3)

        else:     #Otherwise we test on the remaining

          X_test4.append(X[Xdir[filename]]) #Features of subjects 3-5
          #X_test_labels3.append(filename) #contains subjects 3-5
          Y_test4.append(filename[0])


#feature scaling in order to standardize the features
scaler = StandardScaler().fit(X_train4)
X_train4 = scaler.transform(X_train4)
X_test4 = scaler.transform(X_test4)

trainfile = "Question4.dat"

custom_dump_svmlight_file(X_train4, Y_train4, trainfile)

#Cross validation in order to get the best C and Gamma parameter
crange = "-5,13,2" #"1,5,2"
grange = "-15,5,2" #"-3,2,2"
C,gamma,cvrate = paramsfromexternalgridsearch(trainfile, crange, grange,
    printlines=True)
#for 2 accents: best was C=2**3, gamma=2**-15

clf = SVC(gamma=gamma,C=C, kernel="rbf")
clf.fit(X_train4,Y_train4)

#Passing in the test samples against the created model
modelPrediction4 = clf.predict(X_test4)

print("The model accuracy is:",metrics.accuracy_score(Y_test4,
    modelPrediction4)*100,"%")
```

```
    print("precision scores")
    print("Macro: ",precision_score(Y_test4, modelPrediction4, average='macro
        ')*100,"%")
    print("recall scores")
    print("Macro: ",recall_score(Y_test4, modelPrediction4, average='macro')
        *100,"%")
    print("f1 scores")
    print("Macro: ",f1_score(Y_test4, modelPrediction4, average='macro')*100,
        "%")

    writetopcklfile("Question4.model",clf)

main()
```