UNIVERSITY OF WESTMINSTER

On Classification and Recognition of Emotions by Investigating Equivalent Speech and Textual Data

Ugur Bastug, 157435854

Module: ECSC799

Supervisor: Dr Simon Courtenage

Submission Date: 14 September 2016

Keywords: machine-learning, automatic-speech-recognition, emotion-recognition

Word count: 9,999

Abstract:

The communication between human and computers is a charming topic to machine learning enthusiasts. Today's technology is developing rapidly and allowing visionaries to research and experiment with human-like computer behaviour. This paper discusses a new way to predict emotions using human speech as the input, and investigates the most informative elements of human voice that enables a speech sample to be classified into one of the most essential emotions.

This report is submitted in partial fulfilment of the requirements for the MSc Computer Science Degree at the University of Westminster.

Acknowledgements

The project couldn’t be accomplished without the existence of the external libraries I used [21]-[25], [28], [32], [35], [55]. I would like to thank Dr. Simon Courtenage for his contribution and consultancy throughout the project.

Chapter I: Introduction to Emotion Classification

1.1 Introduction

It is trivial to infer emotions from everyday human to human communications and interactions. It is an phenomenal experience that we are able to understand a speaker's feelings just by listening. Although it can be speculated that we are able to do so because of the unnoticed, changing aspects of the voice of the speaker, we often predict the feelings of others without even knowing how we are able to do so.

This paper will mainly discuss the application that is created in order to fulfil the task of emotion classification. STE (Speech to Emotion) aims to investigate the capabilities of modern day technologies and has the ability to understand human emotions from speech input due to the analysis of their informative features. This paper contains descriptions of the technologies and the tools used. In order to understand the crucial parts of the task, it is assumed that the reader has a background knowledge about computer science but not necessarily about machine learning. The most commonly used terms and abbreviations can be found in the glossary section in the Appendix H.

The paper will explain and discuss the speech features, classifiers that are capable of predicting emotions and the datasets available for emotion recognition tasks. Towards the end of the paper, the test results gathered from different parameters will be shared in order to describe the effect of different combinations of elements and their effects on emotion detection.

1.2 System Requirements

In order to run STE, Python must be installed on the personal computer. Any Linux based or Windows machine is capable to run the program without performance concerns since all the classifiers and performance occupied elements of the program will be loaded beforehand. STE benefits from some external libraries, therefore the installation of the dependencies may become complicated. In order to successfully

satisfy the requirements, you can run `pip install -r requirements` on the console. This will install all the dependencies recursively by using the dependency list that is created for ease of installation. See Appendix F for installation guideline.

1.3 Previous Works

This section of the paper discusses findings, opinions and perspectives of related works. In order to emphasise the important aspects of the common topics the section is divided into sub sections.

1.3.1 The Use of Mel-Frequency-Cepstral-Coefficients

Most of the automatic speech recognition (ASR) tasks that are specialised in emotion detection favour the use of Mel Frequency Cepstral Coefficients [1], [2], [13] [14], [15], [16]. Since MFCCs are capable of representing the human auditory system, inclusion of these features in the final matrix can be seen in many previous research studies.

Poria et al. [1] suggests that use of MFCCs had a lower impact on the accuracy of sentiment analysis, but exclusion of them led to a lower accuracy for the overall score. Since the exclusion of these features are lowering the accuracy of the end result classification output, we can safely say that the MFCCs are indeed quite useful for ASR.

The most interesting finding regarding to MFCCs are about their behaviour in certain frequency ranges. Researchers suggest [10], [16] that MFCCs yield good representations of the audio data, but only in lower frequency bands.

According to Pan et al., using MFCC with high frequencies often yields error in the results [8]. This observation is further supported by Gudetti [12], saying that they used MFCC as it represents the low frequency region more accurately than high

frequency region. They utilised this strategy to categorise the resonance of the vocal track.

Moreover, the use of statistical information gathered from the coefficients can also be beneficial for the classification task. The statistical features such as standard deviation, min, max and mean of the MFCC values are capable of giving information about the speech signal.

Statistical derivations gathered from the coefficients are also useful for the data normalisation. The mean normalisation process is often done by using cepstral mean subtraction. In cepstral mean subtraction, the means of MFCC features are estimated from speech signal and then the mean value is subtracted from MFCC [13]. Optionally, those values can then be divided to the standard deviation.

Additionally, the speed and the acceleration of the MFCCs are also informative features. Since the Δ and ΔΔ coefficients contain the information about the rate of change in a signal, they are also discriminative features for energetic emotions such as **joy** and **anger**. Feraru and Zbancioc [14] used ΔMFCC and ΔΔMFCC values in their study about MFCC Relevance in Emotion Classification for SRol Database. More detailed information regarding to the MFCC and Delta MFCC can be found in Features chapter in this paper.

1.3.2 Hybrid/Fusion Approaches

STE is not the first project to combine text and speech. Previous projects [4], [7] utilised both text emotion classification and speed emotion classification as components of a hybrid emotion classification system. The Hybrid approaches often contain a similar speech to text pipeline to STE, which contains a speech to text translator.

Previous works on hybrid approaches often treated text and audio classification independently and used their individual output to build a hybrid decision mechanism.

Poria et al.'s [1] research for sentiment analysis from visual, auditory and textual data used various sources for training data including experimental extracted data like social media extracts, Youtube videos and also professional datasets such as EmoDB [26].

Moreover, Houjeij et al.'s [4] model used a set of movie quotes as their training data to conduct a Hybrid emotion classification task. They [4] used TF-IDF as a statistical measurement in text classification and observed the inclusion of the "Term Frequency Inverse Document Frequency" with semantic features that increased their text classification rate by 16%, features which I later adapted in my project after reading this paper. Fusion classification task conducted by Houjeij et al. [4] used preprocessing for audio files. According to their findings, using a bandpass filter between 300Hz and 3400Hz is essential in order to isolate the background noise or any other non human voice from the human source audio data. Furthermore, they applied text processing which consisted of elimination of the non-alphanumeric characters and filtering of stop words. Stop words are the set of words that does not contains any discriminative meaning such as 'the', 'as', etc.

1.3.3 Characteristics of Emotions

Many researchers [3], [7], [48] are in consensus about particular emotions having common characteristic behaviours. Before we talk about the characteristics of the emotions, we should understand which emotions are considered as the "essentials". It is agreed that the emotions **anger, fear, happiness, sadness, surprise** and **disgust** are among the basic emotions [43]. According to Toivanen et al. [3], quoting from Cornelius [43], "The basic emotions are considered "basic" because they are seen to represent represent survival-related patterns of responses to events of the environment". It is debatable that other emotions are also derived from the basic emotions but since it is accepted that the list of basic emotions are defined they are the

best candidates to use for class labels. Now that the basic emotions are defined we can further investigate the behaviour similarities between them.

The common characteristics often shares similar pitch **ranges**. According to Chuang and Wu [7], the pitch of speech associated with **anger** or **joy** is higher than the samples tagged as **sadness** or **disappointment**. They further share their findings about the energy similarities by saying the energy associated with **surprise** and **anger** is generally higher than samples tagged as **fear**.

Sudhakar and Anil [48] discuss the similar characteristics of emotions regarding their pitch based **values**. According to their findings, **angry, fear** and **surprise** emotions have a higher pitch mean value than happy emotions (such as **joy**).

Various cultures in different regions may also effect the way of producing the audio data (way of saying) and can therefore change the results gathered from emotions. This case can be interesting to investigate further, especially by utilising different language datasets as training and test sets (language independent classification). An example for this case is the use of Berlin Emotional Database (Emo-DB) as training set versus an English dataset as testing data set. Experiments regarding this topic conducted this by using SAVEE dataset to train classifier and Emo-DB for testing against train model [14] and gathered a score of accuracy by 98.3071%.

1.3.4 Classifier behaviour

KNN, SVM and Gaussian Mixture Models are the most commonly used classifiers in the case of emotion classification. Many previous works [2], [8], [12], [15] used one or more of these classifiers in their research.

Patel et al. [2] used KNN to model prosodic features and GMM to model pitch related features. They point that the gaussian modelling gives the best results to distinguish emotional classes for pitch, pitch range and average pitch feature based classification. Therefore we can agree that GMM can be beneficial for feature sets that heavily rely on pitch features.

Run-time performance with working on big data can play important roles for the classifier selection process which means that some classifiers can perform faster classifications and others may take much time to process. KNN is a good candidate for this discussion.

KNN is a viable classifier for performance related problems. Toivanen et al. [3] shares their experience in regard to the curse of dimensionality by explaining that for their classification task with KNN, they didn't used all the 43 features and instead used a forward-backward method in order to test all possible feature combinations.

On the other hand, Support Vector Machines are known to output to best results in multi dimensional cases. A previous work used LIBSVM's SVM classifier for their multi class classification task [12], and obtained 98.3017% accuracy by using the features Zero-Crossing Rates, MFCC, Spectral Rolloff. Similarly, Krishnan and Guergachi [9] observed that they obtained the best results by using SVM classifier with a feature set consisting of MFCC and Zero-Crossing-Rates with entropy values. In this regard, we can say that SVM utilised the ZCR better than other classifiers.

1.3.5 Most Discriminative Features

The quality and the accuracy of the application depends on the feature extraction stage. In order to make successful predictions, the quality and the discriminative value of the features should be observed and the features that have a lower impact should be removed. This can be done by inspecting the most discriminative features.

A study conducted by Pao et al. [10] on the search of the most discriminative speech features points out which features should be used on particular cases. They observed that the LPCC often decreases the score and they also disagree on the use of them in feature matrix. They explain the inadequacy of LPCC as follows: "Because the human speech production system is not linear while the LPC model is, the LPC

coefficients only represent the linear property and loss the nonlinear part" [10]. Furthermore, they found that the KNN classifier utilises the mean MFCC values more effectively than all other classifiers.

Chapter II: Design

2.1 Methodology

STE is a research project that aims to observe the characteristics of human emotions and be able to create a meaningful prediction of emotions from unknown speech data. Therefore, the project cannot be applied to traditional software project design. The traditional design approach for commercial or end product software projects often requires written and sketched planning with UML diagrams and similar resources in order to complete the end product with ease and at the same time make it maintainable. Research projects are experimental and prototype driven. In order to gather the best results, change is often inevitable, therefore the classic design approach often blocks the way of creativity and changeability. I wish to make a disclaimer about the absence of a traditional software design approach beforehand in order to explain my point of view about the workflow that works best for a scientific and experimental project.

It should be noted that the second part of the project, the demo a web application, is capable of utilising the traditional design methods of a software application. Therefore, I decided to include Use Case Scenarios for the web version of the application. Since the web version [29], is the “simple” and the limited functional version of the STE, the design material is limited, but it is able to satisfy the necessity in terms of the design of a software project.

2.2 Pipeline and Work Flow

In order to fully grasp the application's working scheme, each processing step should be divided into sub modules and inspected individually. The application pipeline, see Figure 2.1, gives brief information about how the data is prepared, processed, converted into a constructed data and then fed into classifiers.

The following steps are followed in order to obtain the set emotion predictions from an audio file.

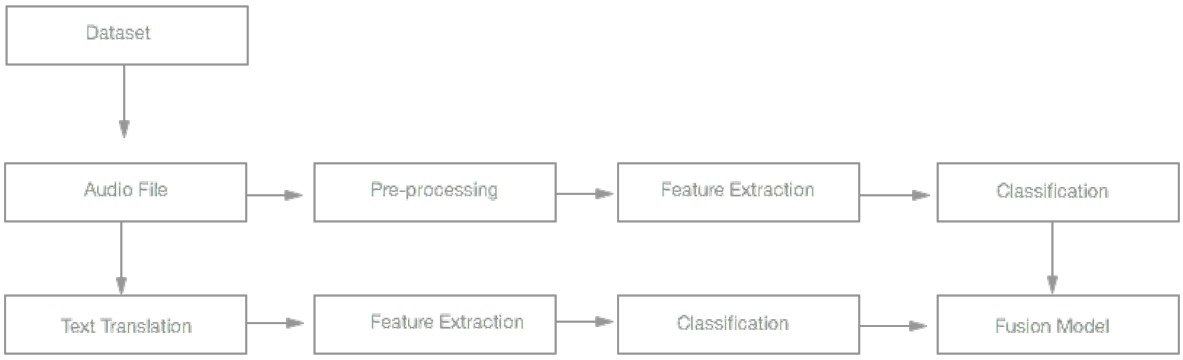


Figure 2.1: Pipeline

Two stages in the pipeline (Translation and Pre-processing) is explained here. The other stages have chapters of their own.

Translation:

In order to obtain the text data, SpeechRecognition is used to translate the audio. SpeechRecognition [28] library supports several engines and is capable of translating the audio file by using Google's Speech Recognition [23] module. It should be noted that, API access is not persistent and the response may take much time. Additionally, the translation is not guaranteed to output a 100% correct result, therefore the hybrid model should adjust the both classifiers weights weights according to the text classifiers reliability.

I created a recovery mechanism that handles any exceptions if the speech recognition fails. In that case, the application passes the translation step and continues to run by taking account of the fact that the text classification will not be evaluated in the hybrid model end result. See Text translation results in Appendix D.1

Preprocessing

We know that the datasets used in the project are all recorded in acoustically perfected environments but for some cases the test results may not be recorded in complete isolation. The preprocessing step is applied for the test samples that are not recorded professionally, especially for the web application inputs.

Some test samples may include unwanted noise in their data such as background noises or sounds that are not produced by human voice. To prevent the noise in the data pre processing steps are applied. Note that the experiments involving split testing does not include the preprocessing step since the recordings are known as acoustically viable.

The first step in the preprocessing stage is to create a bandpass filter. I used a low pass filter and a high pass filter respectively. According to Houjeij et al. [4] the human voice ranges from 300Hz to 3400 Hz, therefore I used the same values.

The next stage is to normalise the audio's amplitude. This step ensures that the audio file does not clip and averages the power of the segment. If the audio file is recorded by using the Web Application, there is a huge chance that the user is also capturing the click button sound in the sample. To avoid this, I sliced the first 300ms of the audio file.

2.3 Data Structure

Python's `defaultdict` is a submodule of the default dictionary class that is easy to work with due to its flexibility in initialisation and object manipulation. The default dictionary is used for various purposes such as storing training data, testing data, feature matrix, probability distribution. In order to iterate elements in a default dictionary `iteritems` method is used in order to iterate over the dictionaries key, value pairs (See Code 2.1).

```
for label,data in test.iteritems():
```

Code 2.1: Sample Iteration

The data is structured for training data in a way that it can be used in both text and audio classification. Audio features are obtained from each sample. I encapsulated the feature extraction (`sf`) which requires the full path of the sample and a parameter that asks whether to use all features or best-selected feature set. After the features are gathered, the dictionary also records the `fullPath`. Extra `fullPath` option is required by the text classification because the text features are obtained after the speech features are extracted and requires the `fullPath` once again in order to read the wav file. This way, audio and text classification can be done by using one constructed efficient data structure (See Code 2.2).

```
test[target].append((sf.getFeatures(fullPath,allFeatures),fullPa
                                                    th))
```

Code 2.2: Test data preparation

Python's JSON module [49] can serialise an object type to a JSON formatted `str`, which is convenient for parsing the `defaultdict` data structure. Example code snippet and output is shown below.

```
ps = dd(list)
ps = ste_knn.prepareKNN(sample,False,training)
#Returns the probability distribution gathered from KNN with
parameters: sample in test set, load classifier option and
training set

>defaultdict(<type    'float'>,    {'joy':    13.544984236201529,
'neutral':  6.9080318050240557,  'sadness':  12.437900068392445,
'disgust':   12.907107415512105,   'anger':   34.78133188746326,
'fear': 19.420644587406613})

print formatJSON(ps)

>{"joy":   13.544984236201529,   "neutral":   6.9080318050240557,
"sadness":  12.437900068392445,  "disgust":  12.907107415512105,
"anger": 34.78133188746326, "fear": 19.420644587406613}
```

Code 2.3: Data-type to JSON serialisation

For the case of the web application, this JSON formatted string is parsed (Code 2.3) in the results page and each emotion's probability is displayed in the view.

In addition to default dictionary, Numpy [22] provides multi-dimensional array objects which I used for creating the feature set. In order to structure the N-dimensional array, feature slicing is applied. The matrix is calculated by taking the the shape of MFCC features(13,13) as the base. `numpy.vstack` is used to vertically concatenate the new array into the matrix. For the features that has more than 13 columns, the column size is divided into sub parts so that they can be concatenated into the final matrix (See code 2.4). The final feature matrix and its shape is further described in the Features chapter.

```
row = calculateRow(fbank_feat,feature_column_size)
ssc_feat = ssc_feat.reshape(row,feature_column_size)
```

Code 2.4: SSC Reshaping

Chapter III:Datasets

3.1 Introduction to Datasets

Choosing the right datasets for a machine learning task is a crucial part of the project since the classifier's predictions can only be good as the quality of the dataset that it is trained with. The selection of datasets used in this project, are recorded in professional acoustically well isolated environments, excluding the experimental movie data set, and used in many speech recognition tasks [14], [8] that are conducted in the past. All dataset samples have similar durations with little to no pause between the beginning and the ending of the recording.

Some samples from the dataset are excluded for particular reasons. Some datasets [26], [33] have extra emotions. In other words some datasets contain emotion classes that are not included in the other datasets. In order to make the prediction unbiased, those samples are removed in order to sustain the probability distribution while doing dataset versus dataset testing.

Below you can find the detailed descriptions and uses of the datasets I've used in this task. The first 3 datasets are used for audio classification and the last dataset is used for text classification.

3.2 Emo-DB: Berlin Database of Emotional Speech

Berlin Database of Emotional Speech [26] consists of 500 audio files that are recorded by professional actors in various emotions. The dataset is recorded by 10 different actors speaking 10 different texts. The database contains the emotions **anger**, **boredom**, **disgust**, **fear**, **joy**, **sadness** and **neutral**. Since the other dataset, TESS [30], does not contains the emotion **boredom**, the samples tagged with **boredom** are excluded from Emo-DB in order to successfully compare the results between TESS and Emo-DB.

All classes have the same amount of data in order to conduct an unbiased experiment. The class **disgust** has the least amount of samples, therefore the other emotion samples are reduced in order to match the size of the samples of **disgust**. Audio files are recorded in German with an average duration of 2 seconds without any noticeable silence or pauses.

One of the main reasons I chose this dataset is to analyse the results of the effect of using different languages and still be able conduct an emotion classification task. Results regarding this task can be observed in Appendix I.6.

3.3 TESS: Toronto Emotional Speech Set

Toronto Emotional Speech Set [30] is an affective speech set that is created by University of Toronto. The dataset contains seven emotions; **anger**, **disgust**, **fear**, **joy**, **pleasant surprise**, **sadness** and **neutral**. In order to choose just the emotions that are shared by all datasets, I excluded the emotion **pleasant surprise**. Since the older and the younger actors have different speaking frequency ranges and TESS dataset is recorded by both younger and older actors, I only used the samples recorded by the actor that is the same age with the Emo-DB actors.

The samples are recorded in 20 hours in several sessions [17]. Each session the actor recorded at least one emotion in order to successfully represent the emotion correctly.

University of Toronto conducted a manual emotion recognition experiment with undergraduate students and the average accuracy of the test is 82% [17], which makes the data reliable and viable for an automatic speech recognition task.

3.4 Experimental Test Set

Conducting cross validation or split dataset tests are a good way to calculate the accuracy and the quality of the classification task, but to make sure the application

responds to real life conditions, it should also be able to output meaningful results from arbitrary data.

In order to test the application for unknown emotional data, I extracted video clips from YouTube and video games to create an experimental test set. The test set is manually constructed to pick the scenes that contain explicit expression of emotions. Sample durations are parallel to the professional training sets: Emo-DB and TESS.

The sources of sample clips are mainly from World of Warcraft: Warlords of Draenor (video-game) [36]-[39] and Harry Potter (cinema movie) [40], [41] series.

3.5 ISEAR

ISEAR [33][34] is a dataset that is created by psychologist during 1990's. The contributors logged their professional experiences into seven major categories; **joy**, **fear**, **anger**, **sadness**, **disgust**, **shame** and **guilt**. The data collection is achieved by logging the reports with their corresponding categories [33]. It is indicated that the dataset contains 3000 respondents for each category. Since the dataset is created by professionals and experts on the science of human emotions, it is viable for use in text classification tasks.

ISEAR is previously used by Thomas et al [53]. They obtained 71.35% accuracy by using Multinomial Naive Bayes Classifier on emotion classification task with seven classes. Their obtained accuracy score indicates that the ISEAR data set is reliable for emotion classification task and therefore I choose to use it on my text classification module.

ISEAR is a CSV formatted file. Each line is a sample with the following structure:

```
tag, content
```

where content is the reported line and the tag is the emotion of the corresponding line.

Chapter IV:Features

4.1 Introduction to Features

In order to prepare the speech samples for analysis, the samples must be represented as arithmetic values by extracting the features. The feature extraction process can be seen as the most important part of the classification process since once we decide which features and identities will be used, the classifier will only take those into account in order to predict the class of the unknown data. To select the best features, many experiments are conducted, including parameter changes, feature reduction and supplementation.

For a speech recognition task, there are some features that are widely and commonly used and proven in practice that they are capable of representing a sound sample meaningfully. Furthermore, speech features can be used for many automatic speech recognition tasks such as speaker identification, gender identification or emotion classification.

4.2 MFCC: Mel-frequency Cepstral Coefficients

As mentioned above, one of the most widely used features in the automatic speech recognition field are the MFCC vectors. In order to fully comprehend the reason why MFCCs are used, Mel scales should be described first.

Mel scales are able to convert the perceived frequency to the actual measured frequency [44]. By using the mel scale, the frequencies can describe the human auditory system in a linear fashion. The formula to convert f hertz into mel scale is shown below:

$$m = 2595 log 10\left(1+\frac{f}{700}\right)$$

Formula 4.1: Mel Scale Mapping [52]

Because the humans are unable hear the loudness and unable to recognise the frequencies in a linear scale, mel scale is applied [44]. The frequency bands of Mel-frequencies cepstrum are equally spaced on the Mel-scale. This transformation approximates the sound to the human auditory system more effectively than linearly spaced frequency bands [45].

The whole process of obtaining mel frequency cepstral coefficients can be described in four main parts.

Pre-emphasis:

In order to treat all samples equally, the samples can be down sampled to a fixed sample rate. This step is optional. Since, in this project, all the training files have the same sample size, STE did not used a pre-emphasis method to calculate the MFCCs.

Windowing:

The beginning and the ending of the frames have high frequency peaks because of the overlapping frames. In order to reduce the edge effect and to avoid distortion the windowing method is applied.

STE used the window length analysis of 0.025 seconds and defined the steps between windows as 0.01.

FFT:

In this step, Fast Fourier Transform is applied to all the frames and the absolute value is taken, then squared. I used the FFT with size 512 [44].

Mel-scale:

As mentioned earlier, because humans are not able to recognise the frequency in linear scales the data must be mapped into mel-scale. The output of the FFT is mapped into mel scale and then the logarithm of the energies are calculated.

Discrete Cosine Transform:

The last step is to take the discrete cosine transform of the signal in order to convert them back into the time domain (See Formula 4.2). The first 12 coefficients are called

Mel Frequency Cepstral Coefficients. The rest of the coefficients are ignored because they represent fast changes in energy that cause noise in the data [44].

$$S_i(k) = \sum_{n=1}^{N} s_i(n)h(n)e^{-\frac{j2\pi kn}{N}}$$

Formula 4.2: Discrete Fourier Transform

4.3 Energy and Filter bank Energy

In addition to Mel-frequency Cepstral Coefficients, Filterbank Energy features also contain important characteristic representations of the speech samples. Paliwal [5] argues that MFCCs alone do not have any physical interpretation, therefore additional energy features are required in the final feature matrix. One downside of using energy features is that they increase the complexity of the feature set. In order to reduce the complexity of the feature set, this project utilised the statistical derivations of the filter bank energies such as mean and standard deviation.

STE uses Log Filter Bank energies, calculated by taking the logarithm of the raw filter bank energy. Similar to the MFCC obtainment process, In order to obtain the Filter Bank Energies the windowing process should be applied. STE uses window length of 0.025s and 0.01 per window iteration (window step).

Filterbank energy Features results 26 coefficients(columns). In order to fit all the filter bank energy features to the classifiers confusion matrix, matrix reshape is applied. Evidently, `N * 26` matrix is reshaped into 2 rows of `N * 13` shaped matrix.

4.4 Spectral Subband Centroids

As mentioned earlier, all the datasets are recorded in acoustically isolated environments in high quality. But, in order to build a speech recognition tool that can be used in real life, the application should “tolerate” noise.

Preprocessing can recover a test sample and trim the noise, but this also causes a loss in dynamics of the sound file. Generally, it is not recommended to apply more than required preprocessing to remove the noise in order not to lose dynamics of the sample. Another way to handle noise can be done by exploring the features that can extract information regardless the noise.

Paliwal [5] suggests that Spectral Subband Centroids have properties that are similar to formant frequencies and are quite robust to noise. Those features can be used as supplementary features in order to increase the recognition performance.

STE benefits from Spectral Subband Centroids by calculating the mean value of all the coefficients.

4.5 Linear Predictive Coding

Before the use of MFCCs became popular the most used features were Linear Predictive Coding coefficients. LPC is a linear predictive model that embodies the characteristics of particular channel of speech [8]. This enables LPCs to be used to differentiate the various characteristics of a speech signal. This information suggest that it is best to use if all the samples are all recorded by the same source. It should be noted that STE uses datasets recorded by various actors, therefore using LPCs can affect the program negatively.

Calculation of the linear predictive coding significantly reduces the performance of the system since it works in polynomial time [54]. The default order parameter of LPC algorithm is `length(signal)-1`, which indicates that the larger the signal, more calculation time will required, therefore decreasing the application’s performance dramatically.

Because of the reason mentioned above, I removed the LPC features in the later stages of the project.

It should also be noted that using LPCs are related to phonetic content of utterance and highly sensible to the phonetics [12]. Since STE uses both German and English languages interdependently (in language independent test cases), it is best not to include a feature that is sensible to phonetics since it has the potential to reduce the success rate.

4.6 Deltas and Delta-Delta Features

So far we included power and energy related features but in order to understand the dynamics of the speech we should have information about the speed and the acceleration. Delta and Delta-Deltas provide the dynamic information in a speech sample. The formula for calculating delta coefficients can be seen below:

$$d_t = \frac{\sum_{n=1}^{N} n\left(c_{t+n}-c_{t-n}\right)}{2\sum_{n=1}^{N} n^2}$$

Formula 4.3: Delta coefficient calculation [44]

Previous works relating to speech recognition have conflicting ideas about the inclusion of Delta and Delta-Delta. Feraru and Zbancioc [14] observed that including Delta MFCC coefficients helps to improve the recognition rate accuracy but including Delta-Deltas does not have such an effect. Therefore they removed the Delta-Deltas in their feature set. On the other hand, Hossan and Memon [16] noted that in their task of IDSAD the use of Delta-Delta features increased their recognition rate from 90.36 by using Delta-MFCCs to 91.35%.

General opinions in regard to Delta features are usually positive [16], [15] and the use of Delta coefficients are recommended not for only emotion recognition tasks

but also speaker discrimination tasks. Hongzi [15] points that the use of all Delta features increases the speaker discrimination better than MFCC does.

4.7 Pitch

The role of pitch related features can provide important information about the emotional state of the speaker. Some emotions have low pitched characteristics such as **sadness.** Similarly, some emotions, such as joy, are often related with higher pitch characteristics. A pitch related feature, fundamental frequency (F0), is an acoustic parameter often used for gender discrimination tasks [56]. The pitch features are often extracted by referencing the F0 value.

The statistical derivations gathered from pitch; such as mean, maximum and minimum are commonly used by previous automatic speech related tasks [8], [14] in order to increase the recognition rate.

Pitch related features are often more useful for speaker discrimination, such as gender recognition tasks. Additionally, It is found that the Gaussian modelling benefits the best from pitch, pitch range and average pitch features in distinguishing emotional classes [2]. There is no such information available for the classifiers STE use such as KNN, Random Forest and Support Vector Machines.

Author Notes:

F0 or raw pitch related informations are not included in STE's final feature set for two reasons. Since the F0 values of female speakers and male speakers varies greatly [14], it is capable of producing errors and therefore reducing the prediction rate. This can have very different classification results due to its speaker gender sensivity. Secondly, the MFC Coefficients also holds information about the variation of F0 and F1-F4 formants [14].

4.8 Zero Crossing Rates

Zero crossing rates is an informative feature that detects how many times a signal crosses zero. ZCR is capable of providing the information frequency of a signal.

According to Nath et al. [50], “voiced speech is produced because of excitation of vocal tract by the periodic flow of air at the glottis and usually shows a low zero-crossing count whereas the unvoiced speech is produced by the constriction of the vocal tract narrow enough to cause turbulent airflow which results in noise and shows high zero-crossing count.”

The formula for calculating zero crossing rates is shown below:

$$zcr = \frac{1}{T} \sum_{t=1}^{T} |s(t) - s(t-1)|$$

Formula 4.4: ZCR Calculation

where s is the signal and T is the length of the signal

4.9 Term-Frequency Inverse Document Frequency

In order to classify emotion from text data, the document categorisation approach is used. Since ISEAR [33] have numerous samples for each emotion, the most important part of the text recognition is to decide which words or n-gram of words are capable of representing the emotions effectively. Hence, the importance of words or word groups per emotion should be modelled. TF-IDF is a reliable approach that is able to extract features from text data by creating a weighting scheme. According to Scikit Learn documentations [21], tf-idf is originally developed for information retrieval for search engines and that it is also found useful in document classification and clustering. In order to extract features, a count vectoriser is used to convert a block of text to matrix of words and counts. This count vectoriser is then used to create the TF-IDF features that outputs a matrix of weighted tokens for each class.

4.10 Feature Set Overview

During the implementation, many experiments are conducted in order to form the best feature set possible (see Figure 4.1). According to the research made previously, the most important feature was the MFFC vectors. I experimented with the relationship between the number of MFCCs included and the results gathered.

MFCCs are not directly included in the final feature matrix. After we obtained the MFCC vectors, the values should be normalised. Mean normalisation method is used in order to satisfy this condition(see Code 4.1).

```
normalised = (np.subtract(feats,np.mean(feats)))/np.std(feats)
```

Code 4.1: Normalisation

Because the FBE coefficients were shaped `(N,26)` and one MFCC feature has 13 values, FBE had to be reshaped into `2*(N,13)` in order to fit them into the final feature matrix (see section 4.3).

Reshaping method is also applied to Spectral Subband Centroids and their mean value is calculated. The raw SSC values were not used because they decreased the recognition rate.

The final feature set consists of 13 MFCC, statistical features such as mean, standard deviation, maximum and minimum values of MFCC and Filterbank energies,and speed and acceleration features of MFCC and FBE.

The final feature have 49 rows and 13 columns. The structure of the matrix is shown below. Note that these features are used for speech/audio classification and the text classification uses TF-IDF features solely. The feature set that is shown below has a very high cross validation rate and is also capable of performing meaningful emotion recognition from unknown experimental movie data sets. Further analysis and information can be found in the Appendix I.

{0,1,..,12}
MFCC x 13
MFCC_MEAN
MFCC_STDEV
MFCC_MAX
MFCC_MIN
$\Delta_{\text{MFCC x 13}}$
$\Delta\Delta_{\text{MFCC x 13}}$
FBE_MEAN
FBE_STDEV
FBE[0]
$\Delta_{\text{FBE[0]}}$
$\Delta\Delta_{\text{FBE[0]}}$
SSC_MEAN

Figure 4.1: Final Feature Matrix

Chapter V:Classifiers

5.1 Classifiers for Emotion Recognition

Training classifiers often takes time and because of that, during the project's life-cycle I recorded all the classifiers used in a folder named Deprecated Classifiers. The old classifiers mainly use the MFCC values with a different number of features. The newer versions contain more complex and experimental feature sets.

Performance-wise KNN is significantly faster than all the other classifiers whereas SVM with linear kernel is the slowest.

The one classifier that is not included here is the Gaussian Mixture Models. Similar tasks [2][14] gave viable results by using state based classifiers such as Gaussian Mixture Models and HMM. Pao et al. [10] conducted the same ASR experiment on KNN and GMM classifiers and noted that KNN classifier produced better results in speaker independent cases compared to GMM classifier.

5.2 K-Nearest Neighbours

K-Nearest Neighbours is a machine learning classification algorithm that works with simple mechanics. KNN classifies an unknown point by calculating the Euclidian distance to the training set and predicts the new label accordingly [46].

KNN is very viable to this specific multi-class emotion prediction task due to its capability in handling cases where the decision boundary is complex. Since this task utilises a multi-class classification task with numerous samples per class, it can be said that the potential decision boundary in the classification task can potentially be complex.

The results obtained from KNN classifier is using 10 neighbours with distance weight method. Distance weight method assigns weight proportional to the inverse of the distance from the query point [46]. The KNN fitting scheme can be seen in the code snippet below:

```
for label, data in training.iteritems():
      for sample in data:
            if(len(label_samples[label])==0):
                  label_samples[label] = sample
            else:
                  label_samples[label] =
np.concatenate((label_samples[label],sample))
            y.extend([label] * len(label_samples[label]))
      if(len(x)==0):
            x = label_samples[label]
      else:
            x = np.concatenate((x,label_samples[label]))
clf = neighbors.KNeighborsClassifier(n_neighbors=n_neighbors,
weights='distance')
clf.fit(x, y)
```

Code 5.1: Dataset construction for classifier fitting

The best recognition results are obtained from the KNN classifiers and the data is shared in Chapter 7.

5.3 Support Vector Machines

In order to use a SVM classifier for a multi-class classification task there are several parameters that needed to be chosen and their effects on the end results should be analysed.

The most trivial usage of SVM in a multi-class task is to use one-vs-rest SVM, which is beneficial for avoiding performance related problems [47]. One-vs-rest approach cannot support different kernels and only works in linear.

Scikit's SVC is capable of multi-class classification with different kernel options and uses one-against-one approach [47]. The approach constructs `(number of classes * (number of classes-1))/2` classifiers and each trains data from two classes.

Zhang [51] et al. explain one-vs-one approach as follows: "when classifying an unknown sample, each classifier judges the samples category and votes will be made

for the corresponding category". Therefore, one-vs-one predicts the label result by taking the majority of the votes from the classifiers.

Two different SVM classifiers RBF Kernel and Linear Kernel (see Figure 5.1) is used in order to compare the accuracy of the emotion classification. The results will be inspected with further detail later in this paper.

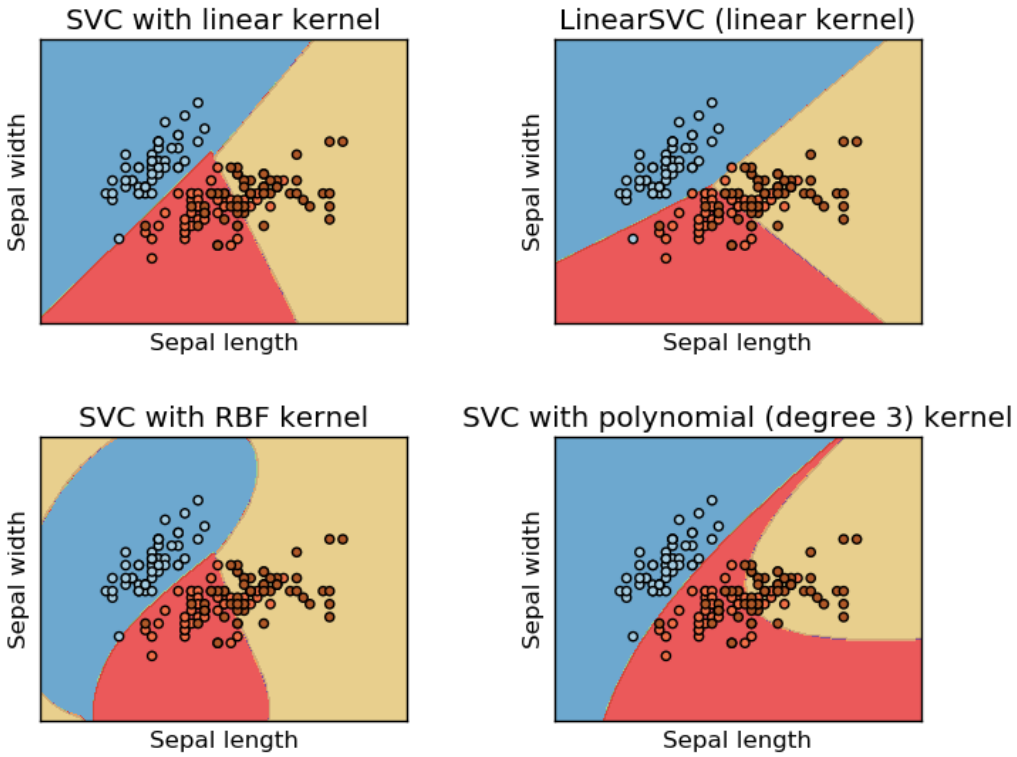


Figure 5.1: SVM Boundaries [47]

5.4 Decision Trees

Decision Trees creates decision rules which is shape varies on the complexity of the data. The created rules are then used to predict the class of the unknown data by using a set of if-then-else decision blocks [57]. Since SVM and KNN classifiers are widely used in ASR tasks and capable of outputting good results, I chose to compare their results with another algorithm which is not so popular in the field. The obtained results are similar to KNN and can be inspected in Appendix I.4.

5.5 Naive Bayes

Naive Bayes classifier is an optional classifier that can be used in the text classification part of the project. The working mechanism of Naive Bayes offers performance solutions since it works faster than SVM in most cases. It should be noted that it is not recommended to use Naive Bayes since the problem has many samples distributed to multiple classes which will potentially make Naive Bayes perform worse than SVM.

Chapter VI:Web Application

6.1 Introduction to Web Application

In order to run STE and conduct an emotion classification experiment, the application's dependent files and libraries should be installed first. The project uses many third party libraries [21]-[25], [28], [31]-[33] and therefore installing the dependencies and the configuration process can be problematic. This problem may be become more complicated for those people who do not have a background in programming. In order to demonstrate the application's capability and working mechanism to the public I created a web environment that can run the simple demo version of the application [29].

The application offers a successfully planned user experience due to its simplicity and clean user interface. The application offers two different ways of uploading speech data. Both methods apply the preprocessing before the file is uploaded to the server.

I. Real time voice recording

The application can record the voice of the user if the user is connected via HTTPS and allowed the microphone input usage. The indicator is shown after the recording process starts and shows the duration of the current recording session. The user can stop the recording by clicking the **Classify** button. The file is automatically exported as 16 bit Wav and uploaded to the server and then routed to the classification process. The voice recording module is using a framework with little modification of the original code [42].

II. Uploading wav file

It is also possible to use pre existing wav files for classification by using the input form. The file is checked if it is a valid operable wav file. If the file is able to pass the requirements, it is then uploaded to the server. When the upload is completed, the file is then routed for the classification (see Figure 6.1).

The Web application can be considered a Lite demo version of the entire project. The Lite version uses the same classifier that is obtained from the full project, therefore, it is capable of providing the same quality of results for the speech classification but it is not capable of conducting a text classification and hence not able to do a hybrid classification. The reasons behind the incapabilities are further observed in Critical Evaluation section in Conclusion chapter.

For debugging purposes, the probability distribution of emotion classes is displayed in the `result` view of the application. User can test and experiment with different wav files and their effect on the probability distributions. The uploaded data is deleted immediately after the user ends the session in order to clean the server space.

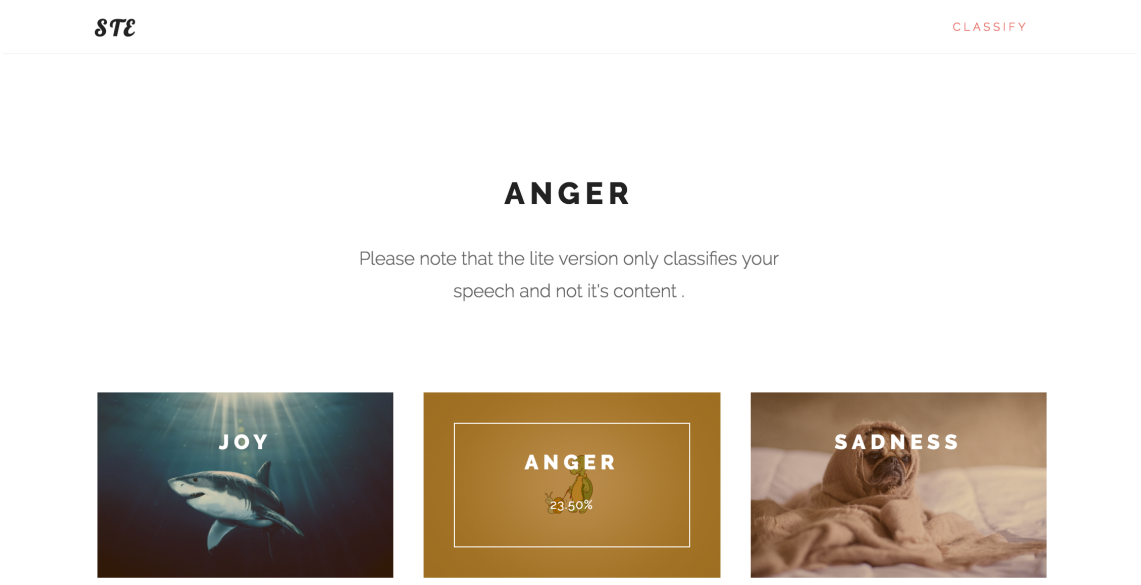


Figure 6.1: Web Interface for classified sample

6.2 Aims and Objectives

As mentioned earlier, the main purpose of creating a web application in addition to the entire project is to enable people to experience the real life examples of an emotion classification task. This aim requires some principles. The application should be easy to use and simple enough to create a test case by following basic steps of uploading a wav file. This also requires the application to have a clean user interface.

It was also necessary to avoid enabling classifier configurations. I chose to only use the best dataset and the best classifier in order to get viable results and avoid potential confusion for everyday users.

6.3 Environment

Heroku [20] is a PaaS that supports Python programming language to create a web application that runs on the cloud. Although the instance offers little to no configuration, it is capable of operating natively python based projects.

The back-end of the application is implemented using Flask [19], which is a Python implemented web framework. Flask is used to import the most essential modules of STE in order to parse, request and respond the data for the whole classification processes.

In order to ensure the Heroku instance can run STE, the dependencies are imported by using a local Virtual Environment. The recorded dependencies are stored in a file named `requirements.txt`. This file is later used in Heroku instance to download all the necessary dependencies that STE uses for the web instance.

6.4 Requirements

A microphone is optional as the user can upload the wav file using the input form. Web application requires HTTPS connection to access the user's microphone and make the recording.

Internet Explorer and Safari does not support getUserMedia whilst current versions of Chrome (52.0.2743.116) and Firefox (43.0.4) does. Because of browser incapability issues, it is not possible to run the classification by using microphone input with IE and Safari.

Chapter VII:Testing, Results and Discussion

7.1 Results & Testing

These results were obtained in unbiased experiments with the purpose of displaying the different outcomes that are produced by experimenting with classifiers, datasets and features. The main objective for inclusion of this section is to prove that the task is reliable, stable and is able to output consistent results if used with recommended set of choices. For space optimisation all the tests are included in the Appendix and only the final test results are included in the main body.

7.2 Single Sample Testing

In order to read the accuracy results first the emotion classification score of single samples should be explained. The examples below show two samples from TESS database. Both samples are taken from the **joy** tagged audio files therefore they are expected to be classified as **joy**

```
Input:  TorontoESS/joy/OAF_witch_happy.wav
{"joy": 44.41, "neutral": 6.18, "sadness": 13.23, "disgust": 14.04,
"anger": 8.43, "fear": 13.69}
joy
```

Code 7.1: Speech Classification Output

The test sample above (Code 7.1) shows that the classifier is certain that the audio file should be classified as **joy** with 44.41%. The second most common prediction is **disgust** with 14.04 but the score is not close to the score of **joy**.

```
Input:  TorontoESS/joy/OAF_wire_happy.wav
{"joy": 21.74, "neutral": 14.53, "sadness": 10.62, "disgust": 21.89,
"anger": 13.021, "fear": 18.16}
disgust
```

Code 7.2: Speech Classification Output

The sample above (Code 7.2) is also taken from the same dataset and should be classified as **joy**. As you can see the score of **joy** and **disgust** is very similar but the

score of **disgust** score is bigger than **joy**. This is an error case for the classification. By further investigating these scores we can make an assumption about which emotions are alike.

It can also be generalised that emotions tagged as **joy** could also have high **disgust** probabilities. In order to form such a conclusion, many test should be done in order to prove that this generalisation is not a matter of coincidence. Therefore I tried to be careful about such implications in this article.

7.3 Accuracy Results

The accuracy results are calculated by using cross validation testing on TESS dataset. For example, the column named **joy** represents the overall recognition rate of **joy** testing samples. Note that these accuracy results are calculated solely on speech input.

	Joy	Sadness	Anger	Fear	Disgust	Neutral
KNN	95%	95%	100%	90%	90%	100%
RBF - SVM	70%	75%	95%	80%	55%	75%
Linear - SVM	70%	75%	10%	70%	55%	95%
Decision Tree	85%	90%	100%	80%	80%	100%

Table 7.1: Classifier Results For Speech Input

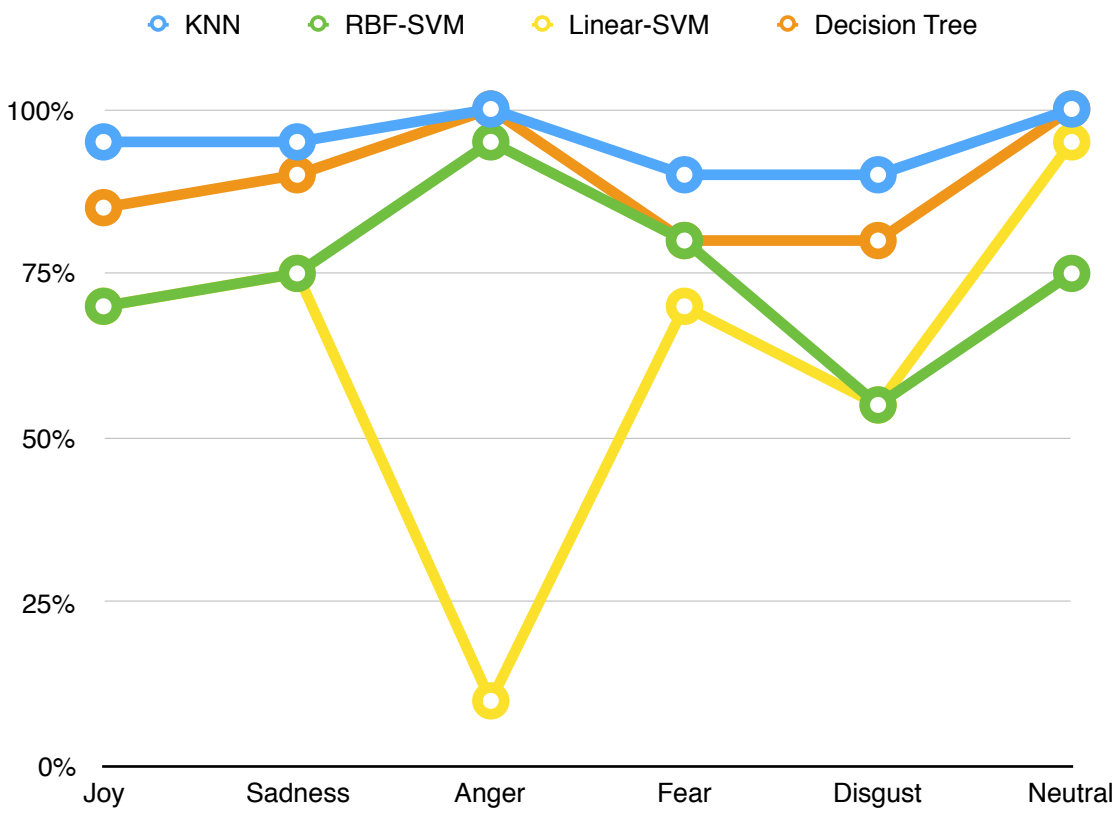


Figure 7.1: Classifier Results For Speech Input

As it can be seen the best score is gathered by using KNN classifier by using the feature set mentioned in Chapter 4.10 on TESS Dataset. All the results gathered by using other Datasets, Different Feature Sets, can be observed in the Appendix I.

7.4 The Case of Hybrid Classification

The Hybrid Model utilises the output of both the Text Classifiers and Audio Classifiers. In order to obtain the mixture results, the best out of {audio,text} classifiers must be selected independently. The experiments and tests show that KNN classifier gives the most reliable predictions for both audio and text classifications (see Appendix I.1 and Appendix I.9). Therefore, the hybrid model is using those two classifiers.

The model takes the data structures and breaks down the probabilities of components. Each component is weighted with a constant that is determined by the classifier's accuracy score. This method is used in order to create a model that gives predictions that is based on the used classifier's reliability.

```
Input:  wow/voldemort_joy_1.wav
{"joy": 23.298929799334971, "neutral": 14.255030324571992, "sadness":
14.607596251010728,     "disgust":     10.198686485271438,     "anger":
23.003846530019157, "fear": 14.635910609791729}
joy

Harry Potter is dead
{"anger":  0.0,  "joy":  0.0,  "fear":  30.670498803304781,  "sadness":
9.7943129542071237, "disgust": 59.535188242488104}
disgust

{"joy": 16.30925085953448, "sadness": 13.163611261969645, "neutral":
9.9785212272003943,     "disgust":     24.999637012436438,     "anger":
16.10269257101341, "fear": 19.446287067845645}
joy
```

Code 7.3: Hybrid Model Classification Output

The test example above (Code 7.3) is obtained by using the Hybrid Model. The first result(**joy**) is gathered by speech, second(**disgust**) by text and third(**joy**) by the hybrid model. It shows us the actor's way of saying of "Harry Potter is dead" is indicating **joy**. But on the other hand, the words itself (content), without considering the

“way of saying”, is indicating **disgust.** The hybrid classifier outputs the final decision of **joy.** It should be noted that the probability of **joy** is decreased compared to the probability of **joy** in the speech classifier.

Chapter IIX:Conclusion

8.1 Related Works

This section of the paper is focused on comparing the similarities and differences of STE and the previous similar applications and research projects.

Dupuis, Pechora-Fuller [17] conducted emotion recognition experiments with the same dataset I used, Toronto Emotional Speech Set. Their classification is made by human interaction and they noted that the emotion classes **anger** and s**adness** had the highest accuracy while the emotion **disgust** had the lowest accuracy. Comparing my results with the same dataset (Appendix A.1), it can be seen that the emotions **anger** (100%), **joy** (95%) and **sadness** (95%) have the highest score while the score of the emotions **disgust** and **fear** is slightly lower(90%). In this perspective I can say the outcome of this research and my results are parallel. Since, as mentioned above, Dupius, Pechora-Fuller's [17] task is conducted by human interaction and not automatic machine classification it is safe to say that my application predictions are very parallel to human predictions.

A previous work [1] used social media and video snippets for datasets similar to what I constructed with Experimental Data Set and similarly to my investigation, they used decision level fusion. The decision level fusion works with separate classifiers with audio and text data and then creates a hybrid model to fuse the results. Moreover, they claim the exclusion of MFCC and SSC lowers the accuracy dramatically. The result section of this paper contains the exclusion of the SSC values and it can be seen that the accuracy lowers. Previous work's use of movie samples [5] and video extraction [1] inspired me to build my own experimental test set by downloading videos that contains strong emotions from Youtube [37]-[41].

Feraru and Zbancioc [14] claim that the Delta MFCC coefficients help to improve the recognition rate accuracy but this is not available for the Delta-Delta MFCC coefficients. My own experiments show that the inclusion of both the delta and delta-deltas of MFCC values results in greater recognition rate on unknown datasets.

Accurate results could not be obtained from STE's language independent experiments. German (Emo-DB) and English (TESS, Experimental) datasets are used in order to achieve a language independent classification task but the results were not satisfying to be considered as successful (see Appendix I.6). On the other hand, the results gathered from a previous task [12], English (SAVEE) versus German (Emo-DB) experiment, is able to obtain an accurate (98.3071%) result. The language dependency may be caused by the constant framing size decision while calculating the MFCC values since the smaller the frame size the more data to examine. But, this also increases the feature matrix and can reduce the performance. On the other hand, it is said that, "the system can be used for capturing voice frames in real time and can be passed as an input to the prediction part of the Speech Emotion Recognition system"[14]. I implemented the web application in order to achieve this feature, enabling all users to record their voice and use it in an emotion classification task.

8.2 New Findings & Propositions

A) Standard Deviation of Delta Filterbank Energy Feature

I observed that the inclusion of the standard deviation of a delta filter bank energy increases the classification score. The accuracy improvement is first observed while working with TESS vs Experimental Dataset.

```
Score List:  {"anger": 87.5, "joy": 60.0, "sadness": 60.0, "fear":
0.0}
Total Score:  51.875
Time elapsed:  33.0720329285
```

Code 8.1: Result Acquired with STD(D_FBANKFEAT)

```
Score List:  {"anger": 75.0, "joy": 60.0, "sadness": 60.0, "fear":
0.0}
Total Score:  48.75
Time elapsed:  31.1210019588
```

Code 8.2: Result Acquired without STD(D_FBANKFEAT)

B) Importance of normalisation of MFCC

The mean normalisation is previously suggested [13] but it can rarely be seen in any other conducted researches, therefore I decided to include this as a new finding or at least a supportive finding that the mean normalisation of the MFCC values improves the accuracy greatly.

C) Emotion Similarity:

During the experiments I've noted some similarities between emotions. I noted that the emotions **anger** and **joy** often give interchangeable results. One possible explanation of this phenomenon may be that the common characteristics, in terms of high magnitude and relatively high pitched, of both **anger** and **joy** emotions. A similar observation is also noted for the emotions **fear** and **disgust**.

8.3 Further Work

The problem with language independent classification can be improved in the future. A possible solution would be the change of window length of MFCCs. Since MFCC windows are the determining factor of a speech signals interval, the shortest interval of a speech signal may capture the language independent aspects of an emotion.

No such previous tasks observed regarding to the inclusion of the Standard Deviation of Delta Filterbank Energy, therefore the effect of inclusion of this feature should further be inspected.

Text classification suffers from the error in translation that is caused by Speech Translation library (see Appendix I.8). Therefore, more test cases should be made in order to improve the accuracy of both the Text Classifier and the Hybrid Classifier. The reason mentioned above ultimately caused the unsatisfactory results gathered from text classification (see Appendix I.9). The overall negative effects are prevented in the final Hybrid Model, but in order to improve the results permanently, more persistent solutions should be implemented. Possible solutions can only be found by, as suggested above, conducting more testing with new test datasets.

Additionally, another minor further work suggestion can be the option of applying preprocessing in the web application. This is suggested by considering a very specific case. In web application, an audio sample that will be uploaded might already have been processed or recorded in an acoustically perfected environment. In those cases, automatically applying the preprocessing can cause loss of information or disrupt the quality of the original sample.

8.4 Critical Evaluation

This section of the paper mainly discusses the overall quality of the application in terms of its functionality and its capability to satisfy the requirements that have previously been stated in the project proposal. All the evaluated topics can be found in this paper in detail and the strong and weak points of the applications are inspected in an unbiased perspective. Additionally, all the topics in this section are discussed by considering the application's life cycle.

- **Web Application:** I proposed to create a web application for everyone to conduct a classification task easily. I completed this requirement and it is currently functional. The only downside of this requirement is the limitation of the application because it only does the audio classification. Since the instance I use does not support many configurations, discovered later on, it was impossible to install the text translation tools I used and therefore I wasn't able to create the Text and Hybrid classification on the web application. This situation later converted in to an advantage to create a simple UI for web.
- **New Proposed Approach:** The proposal noted that one requirement of the project is to propose a new classification mechanism. The entire paper describes the whole mechanism of the new classification project, therefore the requirement is satisfied completely.
- **Changes in features and datasets:** In the proposal I've mentioned that I will be using NRC Lexicon for dataset and keyword extraction for the feature extraction method, specifically in text classification module. Both of them have been replaced with alternatives. I used ISEAR over NRC-Lexicon because of its reliability. As mentioned above, ISEAR is created by professional psychologists and therefore the data is trustable. Discovery of this information changed my mind regarding dataset selection. My further research about text classification also led me to use TF-IDF

instead of keyword instruction because of its popularity in document classification tasks.

As it is mentioned in the Design chapter the traditional design methods are not followed in order to make the project truly changeable and modifiable. On the other hand, the design of the web application is successful in terms of its ease of use and user experience perspective. The use cases for the Web Application aims to produce a way for the user to use the application in very few steps and requirements. Since the web application only offers to classify an audio file the best alternative was to offer the variations of audio upload in a single interface in order to reduce the “step size” of completing the use case.

Python, unarguably, was the best programming language choice due to its community and resources in the machine learning and speech recognition field. The flexible and easy to maintain behaviour of Python enables the restructuring of the code of a module various times without the concern of breaking things. It can be gladly said that no aid of any other programming languages is needed during the implementation process. Also the data structures provided by Python was the best choice in terms of their capability of alteration and modification.

It can be clearly seen that the split dataset results, the test set and the train set using the same data but unknown to each other, outputs satisfactory results ranging between 90% — 100%. Knowing that this is a satisfactory accuracy score but the real world uses of the application can not reach a score high similar to that. My notes regarding to this problem points out few reasons.

- **The quality of the recorded environment:** It is possible to record with a professional condenser microphone but the users are inclined to use their built in microphone while using the web application’s recorder, therefore the quality of the recording will be bad. Preprocessing is capable of eliminating only the removal of the background noise in certain degree and in practical uses it is not possible to entirely convert a bad recording into a professional recording.

- **The speaker gender and unique voice:** The dataset is what determines the output of the speech. TESS dataset is recorded by a middle aged male, therefore the classification takes his voice as a baseline. Which means that the accuracy of the score will decline if the user is not in the same age group and the same gender as the actor that recorded the TESS dataset.

  These reasons can be considered as barriers to conduct a viable emotion classification task by using my web application.

- **Speech to text translation accuracy:** The text translation is a complex task and it is expected to see error in translation. This might be tolerable in some long audio data but for the short samples, like the datas that are used in STE, the talking voice often consists of a few words. The failure in translation in short audio segments creates a huge problem. The results for Text Classification and Hybrid classification shows the error in translation clearly. In order to prevent this, the Hybrid classification weight for text classification reduced but currently there is not a stable solution to this problem. At the first stages of the implementation, this problem couldn't be foreseen, therefore the datasets with small durations were used. Another possible downside of using the library is the fact that it is fully reliable on Google's service. Since it works online and requires an API access, the application's reliability might be at stake if Google chooses to restrict access to outside requests.

Bibliography

[1]S. Poria, E. Cambria, N. Howard, G. Huang and A. Hussain, "Fusing audio, visual and textual clues for sentiment analysis from multimodal content", Neurocomputing, vol. 174, pp. 50-59, 2016.

[2]K. Patil, P. Zope and S. Suralkar, "Emotion Detection From Speech Using Mfcc & Gmm", International Journal of Engineering Research & Technology, vol. 1-9, 2012.

[3]J. Toivanen, E. Vayrynen and T. Seppanen, "Automatic Discrimination of Emotion from Spoken Finnish", Language and Speech, vol. 47, no. 4, pp. 383-412, 2004.

[4]A. Houjei, L. Hamieh, N. Mehdi and H. Hajj, "A Novel Approach for Emotion Classification based on Fusion of Text and Speech", Telecommunications (ICT), 2012 19th International Conference, 2012.

[5]K. Paliwal, "Spectral Subband Centroid Features For Speech Recognition", Automatic Speech Recognition and Understanding, 1997.

[6]C. Alm, D. Roth and R. Sproat, "Emotions from text: machine learning for text-based emotion prediction", HLT '05 Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing, pp. 579-586, 2005.

[7]Z. Chuang and C. Wu, "Multi-Modal Emotion Recognition from Speech and Text", Computational Linguistics and Chinese Language Processing, vol. 9, no. 2, pp. 45-62, 2004.

[8]Y. Pan, P. Shen and L. Shen, "Speech Emotion Recognition Using Support Vector Machine", International Journal of Smart Home, vol. 6, no. 2, 2012.

[9]T. Tabatabaei, S. Krishnan and A. Guergachi, "Emotion Recognition Using Novel Speech Signal Features", Circuits and Systems, 2007.

[10]T. Pao, C. Wang and Y. Li, "A Study on the Search of the Most Discriminative Speech Features in the Speaker Dependent Speech Emotion Recognition", Proceedings of the 2012 Fifth International Symposium on Parallel Architectures, Algorithms and Programming, 2012.

[11]X. Jiang, "Sample Reconstruction and Secondary Feature Selection in Noisy Speech Emotion Recognition", Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2016.

[12]K. T. and R. Guddeti, "Multiclass SVM-based language-independent emotion recognition using selective speech features", Advances in Computing, Communications and Informatics (ICACCI, 2014 International Conference), 2014.

[13]M. Shirali-Shahreza and S. Shirali-Shahreza, "Effect of MFCC normalization on vector quantization based speaker identification", Signal Processing and Information Technology (ISSPIT), 2010.

[14]M. Feraru and M. Zbancioc, "A Study about MFCC Relevance in Emotion Classification for SRoL Database", Electrical and Electronics Engineering (ISEEE), 2013.

[15]Y. Hongzhi, "An Research on Recognition of Tibetan Speakers Based on MFCC and Delta Features", Computer Science-Technology and Applications, IFCSTA, vol. 2, pp. 234-238, 2009.

[16]M. Hossan, S. Memon and M. Gregory, "A Novel Approach for MFCC Feature Extraction", Signal Processing and Communication Systems (ICSPCS), pp. 1-5, 2010.

[17]K. Dupuis and M. Pichora-Fuller, "Recognition of Emotional Speech For Younger and Older Talkers: Behavioural Findings From The Toronto Emotional Speech Set", Journal of the Canadian Acoustical Association, vol. 39, no. 3, 2011.

[18]S.A. Kain, "Voice Conversion Using Deep Neural Networks With Speaker-Independent Pre-Training", IEEE Spoken Language Technology Workshop (SLT), 2014.

[19]"Welcome | Flask (A Python Microframework)", Flask.pocoo.org, 2016. [Online]. Available: http://flask.pocoo.org/. [Accessed: 08- Jul- 2016].

[20]"Cloud Application Platform | Heroku", Heroku.com, 2016. [Online]. Available: https://www.heroku.com/. [Accessed: 08- Jul- 2016].

[21]"scikit-learn: machine learning in Python — scikit-learn 0.17.1 documentation", Scikit-learn.org, 2016. [Online]. Available: http://scikit-learn.org/stable/. [Accessed: 30- Apr- 2016].

[22]"NumPy — NumPy", Numpy.org, 2016. [Online]. Available: http://www.numpy.org/. [Accessed: 30- Apr- 2016].

[23]"Speech API - Speech Recognition | Google Cloud Platform", Google Developers, 2016. [Online]. Available: https://cloud.google.com/speech/. [Accessed: 30- Apr- 2016].

[24]2016. [Online]. Available: https://github.com/ jameslyons/python_speech_features. [Accessed: 30- Apr- 2016].

[25]"ybdarrenwang/python_speech_features", GitHub, 2016. [Online]. Available: https://github.com/ybdarrenwang/python_speech_features. [Accessed: 08- Aug- 2016].

[26]"Emo-DB", Emodb.bilderbar.info, 2016. [Online]. Available: http://emodb.bilderbar.info/start.html. [Accessed: 30- Apr- 2016].

[27]"Natural Language Toolkit — NLTK 3.0 documentation", Nltk.org, 2016. [Online]. Available: http://www.nltk.org/. [Accessed: 30- Apr- 2016].

[28]"SpeechRecognition 3.4.6 : Python Package Index", Pypi.python.org, 2016. [Online]. Available: https://pypi.python.org/pypi/SpeechRecognition/. [Accessed: 08- Sep- 2016].

[29]"SPEECH TO EMOTION", 2016. [Online]. Available: https://ste-lite.herokuapp.com/. [Accessed: 08- Sep- 2016].

[30]"Toronto emotional speech set (TESS) | TSpace Repository", Tspace.library.utoronto.ca, 2016. [Online]. Available: https://tspace.library.utoronto.ca/handle/1807/24487. [Accessed: 02- Jul- 2016].

[31]"shamidreza/pyvocoder", GitHub, 2016. [Online]. Available: https://github.com/shamidreza/pyvocoder. [Accessed: 12- Aug- 2016].

[32]"Scipy", 2016. [Online]. Available: https://www.scipy.org/. [Accessed: 30- Apr- 2016].

[33]"ISEAR Databank — AAAC emotion-research.net - Association for the Advancement of Affective Computing", Emotion-research.net, 2016. [Online]. Available: http://emotion-research.net/toolbox/toolboxdatabase.2006-10-13.2581092615. [Accessed: 08- Sep- 2016].

[34]"bogdanneacsa/tts-master", GitHub, 2016. [Online]. Available: https://github.com/bogdanneacsa/tts-master/tree/master/ISEAR. [Accessed: 08- Sep- 2016].

[35]"jiaaro/pydub @ GitHub", Pydub.com, 2016. [Online]. Available: http://pydub.com/. [Accessed: 08- Aug- 2016].

[36]"The Tomb of Sargeras Audio Drama - WoW", World of Warcraft, 2016. [Online]. Available: https://worldofwarcraft.com/en-us/news/20220886/the-tomb-of-sargeras-audio-drama. [Accessed: 15- Aug- 2016].

[37]"[HQ] All Male Goblin Voice Overs (World of Warcraft Cataclysm)", YouTube, 2016. [Online]. Available: https://www.youtube.com/watch?v=xstSp2U5C3k. [Accessed: 15- Aug- 2016].

[38]"Khadgar Voice Over Audio - Warlords of Draenor", YouTube, 2016. [Online]. Available: https://www.youtube.com/watch?v=KWa5_lC7ci8. [Accessed: 15- Aug- 2016].

[39]"Khadgar Voice Over Audio - Warlords of Draenor", YouTube, 2016. [Online]. Available: https://www.youtube.com/watch?v=KWa5_lC7ci8. [Accessed: 15- Aug- 2016].

[40]"Best scenes in Harry Potter.", YouTube, 2016. [Online]. Available: https://www.youtube.com/watch?v=TT9vPftessg. [Accessed: 15- Aug- 2016].

[41]"Sad Movie Moments", YouTube, 2016. [Online]. Available: https://www.youtube.com/watch?v=CJYqqm3apyA. [Accessed: 15- Aug- 2016].

[42]"Simple Web Audio Recorder", Typedarray.org, 2016. [Online]. Available: http://typedarray.org/wp-content/projects/WebAudioRecorder/. [Accessed: 08- Sep- 2016].

[43]R. Cornelius, The science of emotion. Upper Saddle River, N.J.: Prentice Hall, 1996.

[44]"Practical Cryptography", practicalcryptography.com, 2016. [Online]. Available: http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/. [Accessed: 08- Sep- 2016].

[45]L. Chee, O. Ai, M. Hariharan and S. Yaacob, "MFCC based recognition of repetitions and prolongations in stuttered speech using k-NN and LDA", Research and Development (SCOReD), 2009.

[46]"1.6. Nearest Neighbors — scikit-learn 0.17.1 documentation", Scikit-learn.org, 2016. [Online]. Available: http://scikit-learn.org/stable/modules/neighbors.html. [Accessed: 08- Sep- 2016].

[47]"1.4. Support Vector Machines — scikit-learn 0.17.1 documentation", Scikit-learn.org, 2016. [Online]. Available: http://scikit-learn.org/stable/modules/svm.html. [Accessed: 08- Sep- 2016].

[48]R. Sudhakar and M. Anil, "Analysis of Speech Features for Emotion Detection: A Review", Computing Communication Control and Automation (ICCUBEA), 2015.

[49]"18.2. json — JSON encoder and decoder — Python 2.7.12 documentation", Docs.python.org, 2016. [Online]. Available: https://docs.python.org/2/library/json.html. [Accessed: 08- Sep- 2016].

[50]D. Nath and S. Kalita, "An effective age detection method based on short time energy and zero crossing rate", Business and Information Management (ICBIM), 2014.

[51]W. Zhang, X. Meng, Z. Li, Q. Lu and S. Tan, "Emotion Recognition in Speech Using Multi-classification SVM", Ubiquitous Intelligence and Computing, 2015.

[52]D. O'Shaughnessy, Speech communication. Reading, Mass.: Addison-Wesley Pub. Co., 1987.

[53]T. Bincy, K. Dhanya and P. Vinod, "Synthesized feature space for multiclass emotion classification", Networks & Soft Computing (ICNSC), 2014.

[54]"Formant estimation with LPC coefficients,". [Online]. Available: http://uk.mathworks.com/help/signal/ug/formant-estimation-with-lpc-coefficients.html. Accessed: Sep. 11, 2016.

[55]D. Bellini, "danilobellini/audiolazy", *GitHub*, 2014. [Online]. Available: https://github.com/danilobellini/audiolazy. [Accessed: 11- Sep- 2016].

[56]E. Pepiot, "Male and female speech: a study of mean f0, f0 range, phonation type and speech rate in Parisian French and American English speakers", Speech Prosody, pp. 305-309, 2014.

[57]"1.10. Decision Trees — scikit-learn 0.17.1 documentation", Scikit-learn.org, 2016. [Online]. Available: http://scikit-learn.org/stable/modules/tree.html. [Accessed: 13- Sep- 2016].

Appendices

Appendix A: TESS Split Test Results

KNN, Toronto vs Toronto, Final Feature Set

Test sample size = 20

Test Sample Class: Joy

```
KNN
Correct class:  joy
Input:  TorontoESS/joy/OAF_youth_happy.wav
{"joy": 31.261096607464253, "neutral": 11.197873478479133, "sadness":
15.424732356209436, "disgust": 15.690424527461214, "anger": 11.426318215855868, "fear":
14.999554814530118}
joy

Input:  TorontoESS/joy/OAF_young_happy.wav
{"joy": 31.163491264589776, "neutral": 16.235536079346215, "sadness":
16.087017029780732, "disgust": 10.997931085416651, "anger": 8.5569258418569891, "fear":
16.959098699009633}
joy

Input:  TorontoESS/joy/OAF_yes_happy.wav
{"joy": 40.890256789295627, "neutral": 12.658720556292312, "sadness":
11.919464349453852, "disgust": 14.430342332039274, "anger": 10.043380508070264, "fear":
10.057835464848671}
joy

Input:  TorontoESS/joy/OAF_yearn_happy.wav
{"joy": 46.836497485205598, "neutral": 8.4577722888139579, "sadness":
11.316930372702387, "disgust": 9.5477572858969779, "anger": 9.7996850530309736, "fear":
14.041357514350091}
joy

Input:  TorontoESS/joy/OAF_witch_happy.wav
{"joy": 44.416338250469522, "neutral": 6.1806320283270075, "sadness":
13.236310649926766, "disgust": 14.041405998424983, "anger": 8.4329059193102367, "fear":
13.692407153541499}
joy

Input:  TorontoESS/joy/OAF_wire_happy.wav
{"joy": 21.748403686909917, "neutral": 14.536369619436703, "sadness":
10.629359863941673, "disgust": 21.895932914089478, "anger": 13.021596315051079, "fear":
18.168337600571157}
disgust

Input:  TorontoESS/joy/OAF_wife_happy.wav
{"joy": 29.865846321121655, "neutral": 12.482851381569919, "sadness":
9.2559342153077733, "disgust": 10.280053090194547, "anger": 23.13815482340479, "fear":
14.97716016840133}
joy

Input:  TorontoESS/joy/OAF_white_happy.wav
{"joy": 24.271656751710921, "neutral": 16.289671814536852, "sadness":
16.480251762662665, "disgust": 12.591550204147801, "anger": 7.7198134087558996, "fear":
22.647056058185854}
joy

Input:  TorontoESS/joy/OAF_whip_happy.wav
{"joy": 43.868352464808396, "neutral": 11.213455679234583, "sadness":
12.249573824228239, "disgust": 9.8134570480306706, "anger": 9.4539595814479878, "fear":
13.401201402250104}
joy

Input:  TorontoESS/joy/OAF_which_happy.wav
```

```
{"joy": 20.707621878620081, "neutral": 19.548414533832144, "sadness":
13.250258817252174, "disgust": 16.031931223412354, "anger": 16.330475101251587, "fear":
14.131298445631659}
joy

Input:  TorontoESS/joy/OAF_when_happy.wav
{"joy": 51.003285106772111, "neutral": 6.550795099928088, "sadness": 12.579922623533164,
"disgust": 10.748555543848267, "anger": 11.034483948164134, "fear": 8.0829576777542282}
joy

Input:  TorontoESS/joy/OAF_wheat_happy.wav
{"joy": 52.783808417731905, "neutral": 8.5451600866511974, "sadness":
7.6844655642245101, "disgust": 10.886388494443885, "anger": 10.943396225245566, "fear":
9.1567812117029472}
joy

Input:  TorontoESS/joy/OAF_week_happy.wav
{"joy": 23.730527770044475, "neutral": 20.865438662412394, "sadness":
15.776066304628676, "disgust": 15.556837433893186, "anger": 16.569612003538172, "fear":
7.5015178254830959}
joy

Input:  TorontoESS/joy/OAF_wash_happy.wav
{"joy": 38.386831615738089, "neutral": 8.2931769760193621, "sadness":
7.3131047426284947, "disgust": 12.377051159050765, "anger": 23.466669017315095, "fear":
10.163166489248187}
joy

Input:  TorontoESS/joy/OAF_walk_happy.wav
{"joy": 41.839038664257821, "neutral": 10.742482145075215, "sadness":
10.798100160135606, "disgust": 12.100714637293004, "anger": 14.995340628389481, "fear":
9.5243237648488766}
joy

Input:  TorontoESS/joy/OAF_wag_happy.wav
{"joy": 31.377631572802894, "neutral": 13.414487910319471, "sadness":
11.000138175853865, "disgust": 16.966017805073257, "anger": 11.887578889253996, "fear":
15.3541456466965}
joy

Input:  TorontoESS/joy/OAF_vote_happy.wav
{"joy": 27.060795139890871, "neutral": 10.657311404727213, "sadness":
14.516708233085355, "disgust": 12.467983928448236, "anger": 23.61168397522027, "fear":
11.685517318628049}
joy

Input:  TorontoESS/joy/OAF_void_happy.wav
{"joy": 48.515923594168697, "neutral": 7.866745253117517, "sadness": 6.8946223410847214,
"disgust": 10.919217482354473, "anger": 12.892239668104336, "fear": 12.911251661170253}
joy

Input:  TorontoESS/joy/OAF_voice_happy.wav
{"joy": 34.745911909183668, "neutral": 14.404918652588657, "sadness":
13.602074756388854, "disgust": 13.983868848218549, "anger": 10.797570181821797, "fear":
12.46565565179848}
joy

Input:  TorontoESS/joy/OAF_vine_happy.wav
{"joy": 50.751631321059399, "neutral": 9.2550506231910177, "sadness":
10.189350062563872, "disgust": 9.6969756876599327, "anger": 8.7042160375808511, "fear":
11.402776267944946}
joy

-----
Score List:  {"joy": 95.0}
Total Score:  95.0
Time elapsed:  24.224023819
```

Test Sample Class: Sadness

```
Correct class:  sadness
Input:  TorontoESS/sadness/OAF_youth_sad.wav
{"joy": 6.8171290448089845, "neutral": 31.011480145923667, "sadness":
30.558365418066359, "disgust": 10.924530413790931, "anger": 5.2385443124284512, "fear":
15.449950664981609}
neutral

Input:  TorontoESS/sadness/OAF_young_sad.wav
{"joy": 15.017042421894342, "neutral": 10.611426418514265, "sadness":
35.789070459738646, "disgust": 16.231667236014875, "anger": 10.243466209843815, "fear":
12.107327253994059}
sadness

Input:  TorontoESS/sadness/OAF_yes_sad.wav
{"joy": 6.8656575026093378, "neutral": 23.75193525581631, "sadness": 31.716752074636958,
"disgust": 9.7122914162521337, "anger": 8.7229734813958295, "fear": 19.230390269289433}
sadness

Input:  TorontoESS/sadness/OAF_yearn_sad.wav
{"joy": 6.7441729403809667, "neutral": 24.974669180512247, "sadness":
32.418064734196221, "disgust": 15.823862285357732, "anger": 6.954429208740339, "fear":
13.084801650812491}
sadness

Input:  TorontoESS/sadness/OAF_witch_sad.wav
{"joy": 11.837763482268588, "neutral": 18.842938624902853, "sadness":
30.915560705202221, "disgust": 11.93244783614657, "anger": 14.361090136986203, "fear":
12.110199214493566}
sadness

Input:  TorontoESS/sadness/OAF_wire_sad.wav
{"joy": 9.5330298845207402, "neutral": 20.731865981976721, "sadness":
35.050224200823138, "disgust": 16.343719304902045, "anger": 7.6888513395651552, "fear":
10.652309288212214}
sadness

Input:  TorontoESS/sadness/OAF_wife_sad.wav
{"joy": 9.8770991428347745, "neutral": 10.695998438205994, "sadness":
41.655962375112985, "disgust": 12.986670121009238, "anger": 5.3068998565613281, "fear":
19.477370066275689}
sadness

Input:  TorontoESS/sadness/OAF_white_sad.wav
{"joy": 9.3498592712551929, "neutral": 8.6991133453217362, "sadness":
42.334683981563259, "disgust": 8.8294736832963086, "anger": 9.1213145466162491, "fear":
21.665555171947258}
sadness

Input:  TorontoESS/sadness/OAF_whip_sad.wav
{"joy": 15.454942182553994, "neutral": 13.484883097610227, "sadness":
33.258549775463017, "disgust": 8.8851875880257829, "anger": 6.4675321296470045, "fear":
22.448905226699978}
sadness

Input:  TorontoESS/sadness/OAF_which_sad.wav
{"joy": 7.3127412824210651, "neutral": 16.634791073124141, "sadness":
41.873134009205849, "disgust": 13.519989478825648, "anger": 11.319564849904943, "fear":
9.3397793065183539}
sadness

Input:  TorontoESS/sadness/OAF_when_sad.wav
{"joy": 18.443907664158328, "neutral": 16.180136847527859, "sadness":
31.041158300697425, "disgust": 14.431345414890805, "anger": 7.3994639073076236, "fear":
12.503987865417955}
sadness

Input:  TorontoESS/sadness/OAF_wheat_sad.wav
```

```
{"joy": 12.099209748387034, "neutral": 7.2057478294006918, "sadness":
50.569309833676066, "disgust": 10.987355462091429, "anger": 7.5779395943653816, "fear":
11.560437532079394}
sadness

Input:  TorontoESS/sadness/OAF_week_sad.wav
{"joy": 10.385583972043717, "neutral": 17.761967766923558, "sadness":
40.454058963221634, "disgust": 9.1108594690279148, "anger": 5.3462518289174792, "fear":
16.94127799986569}
sadness

Input:  TorontoESS/sadness/OAF_wash_sad.wav
{"joy": 15.052234639967697, "neutral": 13.178241312069396, "sadness":
35.114843261211988, "disgust": 14.976615390451428, "anger": 5.7352019975173159, "fear":
15.942863398782174}
sadness

Input:  TorontoESS/sadness/OAF_walk_sad.wav
{"joy": 13.787917089844242, "neutral": 9.7731583677217149, "sadness":
39.940953150330358, "disgust": 17.194841289453681, "anger": 7.1466644933618344, "fear":
12.156465609288176}
sadness

Input:  TorontoESS/sadness/OAF_wag_sad.wav
{"joy": 8.5689068324925923, "neutral": 24.313906553993927, "sadness":
31.144779833119959, "disgust": 11.501493882988459, "anger": 15.036171894583479, "fear":
9.4347410028215783}
sadness

Input:  TorontoESS/sadness/OAF_vote_sad.wav
{"joy": 20.006650400581016, "neutral": 7.4034020240643832, "sadness":
43.787851422827472, "disgust": 11.921271648080193, "anger": 8.4837074688613008, "fear":
8.3971170355856337}
sadness

Input:  TorontoESS/sadness/OAF_void_sad.wav
{"joy": 12.043007200139833, "neutral": 19.455318078492642, "sadness":
38.454197566259971, "disgust": 15.795907913658382, "anger": 5.6278048511327583, "fear":
8.6237643903163974}
sadness

Input:  TorontoESS/sadness/OAF_voice_sad.wav
{"joy": 11.238761253430463, "neutral": 15.796467984500351, "sadness":
40.313916635046695, "disgust": 12.873464994605969, "anger": 10.395803102176371, "fear":
9.3815860302401415}
sadness

Input:  TorontoESS/sadness/OAF_vine_sad.wav
{"joy": 10.357669084146947, "neutral": 14.360675056556087, "sadness":
31.491025655275436, "disgust": 13.207041600079847, "anger": 8.4729473913899209, "fear":
22.110641212551762}
sadness

-----
Score List:  {"sadness": 95.0}
Total Score:  95.0
Time elapsed:  24.7354068756
```

Test Sample Class: Fear

```
Correct class:  fear
Input:  TorontoESS/fear/OAF_youth_fear.wav
{"joy": 34.891180000790598, "neutral": 7.5043426057074107, "sadness": 13.14042900893446,
"disgust": 10.922384307830908, "anger": 10.652934204082824, "fear": 22.888729872653798}
joy

Input:  TorontoESS/fear/OAF_young_fear.wav
{"joy": 11.453153342994089, "neutral": 7.460244582486161, "sadness": 21.671441135267216,
"disgust": 13.89801719657599, "anger": 7.8186969378533258, "fear": 37.698446804823213}
fear

Input:  TorontoESS/fear/OAF_yes_fear.wav
{"joy": 15.097023890132911, "neutral": 16.262711815872049, "sadness":
12.281853458761852, "disgust": 13.532882810217743, "anger": 15.181812630046815, "fear":
27.64371539496862}
fear

Input:  TorontoESS/fear/OAF_yearn_fear.wav
{"joy": 18.178084359789757, "neutral": 13.053668714309953, "sadness":
5.1445845216753963, "disgust": 12.311979024301893, "anger": 20.57376061292355, "fear":
30.737922766999461}
fear

Input:  TorontoESS/fear/OAF_witch_fear.wav
{"joy": 12.731096279689126, "neutral": 14.483655123652323, "sadness":
19.240867591705001, "disgust": 8.9929849850885777, "anger": 11.843384968897961, "fear":
32.708011050967023}
fear

Input:  TorontoESS/fear/OAF_wire_fear.wav
{"joy": 8.0006207417214696, "neutral": 14.6441499764029, "sadness": 18.453130682509144,
"disgust": 12.455043319762325, "anger": 14.87534695090941, "fear": 31.571708328694758}
fear

Input:  TorontoESS/fear/OAF_wife_fear.wav
{"joy": 12.173665776935596, "neutral": 8.1981671911472009, "sadness":
15.210871037555988, "disgust": 16.71035072722589, "anger": 11.771086119623595, "fear":
35.935859147511735}
fear

Input:  TorontoESS/fear/OAF_white_fear.wav
{"joy": 9.8140311960390267, "neutral": 23.752878982234506, "sadness":
16.063791750818833, "disgust": 11.425219770580874, "anger": 7.2549739952391334, "fear":
31.689104305087614}
fear

Input:  TorontoESS/fear/OAF_whip_fear.wav
{"joy": 17.055847598506261, "neutral": 11.954043536257233, "sadness":
14.708773079763168, "disgust": 17.808615377877011, "anger": 5.9242364873961382, "fear":
32.548483920200198}
fear

Input:  TorontoESS/fear/OAF_which_fear.wav
{"joy": 30.833506644657561, "neutral": 5.6713781120464164, "sadness":
16.214274392357996, "disgust": 10.679657382572946, "anger": 8.6981124366376612, "fear":
27.903071031727432}
joy

Input:  TorontoESS/fear/OAF_when_fear.wav
{"joy": 18.251038355178416, "neutral": 9.2352813495862094, "sadness":
14.215097167144918, "disgust": 11.104703134446069, "anger": 12.376796937994596, "fear":
34.817083055649789}
fear

Input:  TorontoESS/fear/OAF_wheat_fear.wav
{"joy": 10.710447389023514, "neutral": 16.006874757691318, "sadness":
15.278648823217896, "disgust": 17.698925024980856, "anger": 7.2042127749247511, "fear":
33.100891230161665}
fear
```

```
Input:  TorontoESS/fear/OAF_week_fear.wav
{"joy": 10.728150797874262, "neutral": 9.7663802377327578, "sadness":
18.455215244112932, "disgust": 11.111253986488755, "anger": 11.158878778327479, "fear":
38.780120955463808}
fear

Input:  TorontoESS/fear/OAF_wash_fear.wav
{"joy": 18.37674510870762, "neutral": 6.0531971747068507, "sadness": 11.977074622090308,
"disgust": 14.597138171472404, "anger": 14.332781023458384, "fear": 34.663063899564428}
fear

Input:  TorontoESS/fear/OAF_walk_fear.wav
{"joy": 11.80514934984638, "neutral": 14.660932579223662, "sadness": 10.5391642509323,
"disgust": 14.965348411371922, "anger": 14.031375833660553, "fear": 33.99802957496518}
fear

Input:  TorontoESS/fear/OAF_wag_fear.wav
{"joy": 11.545824637571013, "neutral": 10.190075026938882, "sadness": 16.44463358708251,
"disgust": 12.951558846868389, "anger": 8.0097235888159908, "fear": 40.858184312723203}
fear

Input:  TorontoESS/fear/OAF_vote_fear.wav
{"joy": 18.585827314966675, "neutral": 11.1884416132824, "sadness": 10.714548229298581,
"disgust": 14.221025639336997, "anger": 14.516757866845802, "fear": 30.773399336269527}
fear

Input:  TorontoESS/fear/OAF_void_fear.wav
{"joy": 12.280958589554798, "neutral": 9.7347362175689156, "sadness":
12.544233219626625, "disgust": 14.102101454492457, "anger": 19.5099820503583, "fear":
31.827988468398917}
fear

Input:  TorontoESS/fear/OAF_voice_fear.wav
{"joy": 18.810215835643213, "neutral": 12.75941154114448, "sadness": 10.800484970403211,
"disgust": 11.854893337936682, "anger": 11.172734791745112, "fear": 34.602259523127309}
fear

Input:  TorontoESS/fear/OAF_vine_fear.wav
{"joy": 12.398511303723694, "neutral": 10.191669578400248, "sadness":
17.201542684576612, "disgust": 15.049393311899681, "anger": 11.681132589438743, "fear":
33.477750531961028}
fear

-----
Score List:  {"fear": 90.0}
Total Score:  90.0
Time elapsed:  24.8652257919
```

Test Sample Class: Disgust

```
Correct class:  disgust
Input:  TorontoESS/disgust/OAF_youth_disgust.wav
{"joy": 9.3088935772917658, "neutral": 16.339744056177327, "sadness":
13.494074465309479, "disgust": 26.986408935771223, "anger": 7.738513839720417, "fear":
26.132365125729773}
disgust

Input:  TorontoESS/disgust/OAF_young_disgust.wav
{"joy": 23.354851794325992, "neutral": 14.98042562932873, "sadness": 12.475017531400557,
"disgust": 25.249818686045494, "anger": 7.2416008171267201, "fear": 16.698285541772517}
disgust

Input:  TorontoESS/disgust/OAF_yes_disgust.wav
{"joy": 8.3974371769043099, "neutral": 18.532183734859988, "sadness":
15.257194216503285, "disgust": 33.669596231121524, "anger": 7.0197330895360333, "fear":
17.123855551074854}
disgust

Input:  TorontoESS/disgust/OAF_yearn_disgust.wav
{"joy": 15.724400998563866, "neutral": 6.8453876390234196, "sadness":
26.474560883698771, "disgust": 25.993127068919087, "anger": 11.655306567662887, "fear":
13.307216842131972}
sadness

Input:  TorontoESS/disgust/OAF_witch_disgust.wav
{"joy": 7.6906997642391506, "neutral": 34.378190674422413, "sadness":
13.240547952173157, "disgust": 25.875851783652102, "anger": 9.5273953321770684, "fear":
9.2873144933361029}
neutral

Input:  TorontoESS/disgust/OAF_wire_disgust.wav
{"joy": 5.7669397712908497, "neutral": 16.987716873427431, "sadness":
18.039520597644326, "disgust": 32.942139144423571, "anger": 9.4392933729387831, "fear":
16.824390240275058}
disgust

Input:  TorontoESS/disgust/OAF_wife_disgust.wav
{"joy": 10.859798886221359, "neutral": 15.480462852122672, "sadness":
10.445683373805457, "disgust": 30.503399473728233, "anger": 20.560438022480078, "fear":
12.150217391642196}
disgust

Input:  TorontoESS/disgust/OAF_white_disgust.wav
{"joy": 11.18284421307553, "neutral": 11.372578599813489, "sadness": 19.888697520496823,
"disgust": 31.069936703951669, "anger": 9.8795803358664944, "fear": 16.606362626795981}
disgust

Input:  TorontoESS/disgust/OAF_whip_disgust.wav
{"joy": 13.757099276300806, "neutral": 17.393925487526513, "sadness":
11.968587979851506, "disgust": 34.875870518731496, "anger": 11.195115784436698, "fear":
10.809400953152977}
disgust

Input:  TorontoESS/disgust/OAF_which_disgust.wav
{"joy": 9.8838145142719043, "neutral": 9.7774426704688633, "sadness": 22.34963006587466,
"disgust": 29.39653101527086, "anger": 6.5790094724822161, "fear": 22.013572261631488}
disgust

Input:  TorontoESS/disgust/OAF_when_disgust.wav
{"joy": 14.634190392142191, "neutral": 15.955793369192852, "sadness":
22.123818640227121, "disgust": 29.295191961363702, "anger": 6.635121039661537, "fear":
11.355884597412592}
disgust

Input:  TorontoESS/disgust/OAF_wheat_disgust.wav
{"joy": 8.1954089453372401, "neutral": 11.130052351061016, "sadness":
15.283131267752038, "disgust": 38.613888231577739, "anger": 10.361990591793182, "fear":
16.415528612478781}
disgust
```

```
Input:  TorontoESS/disgust/OAF_week_disgust.wav
{"joy": 13.068227497264772, "neutral": 8.6138481810848475, "sadness":
14.540936358621162, "disgust": 39.497631027729362, "anger": 10.943792427232815, "fear":
13.335564508067042}
disgust

Input:  TorontoESS/disgust/OAF_wash_disgust.wav
{"joy": 10.88976829025718, "neutral": 16.625826974965562, "sadness": 12.80917321531302,
"disgust": 31.29576782166875, "anger": 9.5385339213161178, "fear": 18.840929776479378}
disgust

Input:  TorontoESS/disgust/OAF_walk_disgust.wav
{"joy": 16.290050026020488, "neutral": 13.636392320808858, "sadness": 14.73192403636919,
"disgust": 27.967109600448325, "anger": 6.9674007842282917, "fear": 20.407123232124849}
disgust

Input:  TorontoESS/disgust/OAF_wag_disgust.wav
{"joy": 11.517197302828112, "neutral": 14.781197326071007, "sadness": 12.45058451642358,
"disgust": 31.106660362850974, "anger": 14.2075064300337, "fear": 15.936854061792626}
disgust

Input:  TorontoESS/disgust/OAF_vote_disgust.wav
{"joy": 11.460249813788458, "neutral": 22.79442850826285, "sadness": 16.801510589657553,
"disgust": 28.178496067805781, "anger": 8.9793299714202135, "fear": 11.78598504906514}
disgust

Input:  TorontoESS/disgust/OAF_void_disgust.wav
{"joy": 12.710067997693342, "neutral": 17.344110905614059, "sadness":
11.431632972034546, "disgust": 24.512907566441008, "anger": 16.504684881680088, "fear":
17.496595676536963}
disgust

Input:  TorontoESS/disgust/OAF_voice_disgust.wav
{"joy": 14.245940542690533, "neutral": 12.669496615907036, "sadness":
8.9453062501388771, "disgust": 25.064518938072798, "anger": 23.265508441633411, "fear":
15.809229211557346}
disgust

Input:  TorontoESS/disgust/OAF_vine_disgust.wav
{"joy": 12.495212525005483, "neutral": 21.618096839705125, "sadness":
11.849720245492898, "disgust": 32.092676649667141, "anger": 6.4096536543342912, "fear":
15.534640085795058}
disgust

-----
Score List:  {"disgust": 90.0}
Total Score:  90.0
Time elapsed:  26.2436139584
```

Note: Anger and Neutral is not included because they have a 100% Accuracy

Appendix B: Experimental Test Results -KNN

This appendix shows the different results gathered from the change of features by using the best resulting classifier KNN.

B.1: Final Feature Set

```
KNN
Correct class:  anger
Input:  wow/fightclub_anger_1.wav
{"joy": 13.544984236201529, "neutral": 6.9080318050240557, "sadness":
12.437900068392445, "disgust": 12.907107415512105, "anger": 34.78133188746326, "fear":
19.420644587406613}
anger

Input:  wow/guldan_anger_1.wav
{"joy": 19.80906361492686, "neutral": 8.5922419166502753, "sadness": 23.571455579059744,
"disgust": 14.192622563164377, "anger": 18.15246646830208, "fear": 15.68214985789667}
sadness

Input:  wow/khadgar_anger_1.wav
{"joy": 16.250891067802627, "neutral": 16.705370504220451, "sadness":
16.915829396085883, "disgust": 14.420129945406678, "anger": 19.257324638083691, "fear":
16.450454448400674}
anger

Input:  wow/khadgar_anger_2.wav
{"joy": 17.21929975249159, "neutral": 12.249741201558345, "sadness": 17.802897593396661,
"disgust": 17.244101130387456, "anger": 19.97110487423614, "fear": 15.512855447929802}
anger

Input:  wow/khadgar_anger_3.wav
{"joy": 15.495881022243697, "neutral": 18.475917890807391, "sadness": 14.14919947301731,
"disgust": 18.08028575088936, "anger": 19.607516882845232, "fear": 14.19119898019701}
anger

Input:  wow/khadgar_anger_4.wav
{"joy": 15.698095307927014, "neutral": 14.985619758366724, "sadness":
16.487017965089226, "disgust": 15.708828458534819, "anger": 21.028989567876867, "fear":
16.091448942205346}
anger

Input:  wow/kiljaiden_anger_1.wav
{"joy": 20.359739754269857, "neutral": 9.3039102967587812, "sadness":
16.605155181556231, "disgust": 15.006627721067337, "anger": 23.36311303223906, "fear":
15.361454014108737}
anger

Input:  wow/kiljaiden_anger_2.wav
{"joy": 19.256113187346866, "neutral": 9.6914340577897615, "sadness":
16.822120984686308, "disgust": 13.612041118793686, "anger": 24.944458482703592, "fear":
15.673832168679793}
anger

Correct class:  joy
Input:  wow/goblin_joy_1.wav
{"joy": 23.310817880889726, "neutral": 6.6665162389682795, "sadness":
15.238271640468886, "disgust": 16.437281095727904, "anger": 22.96017989934726, "fear":
15.386933244597937}
joy

Input:  wow/khadgar_joy_1.wav
{"joy": 23.467116418233314, "neutral": 14.251058089132167, "sadness":
15.808189086120915, "disgust": 12.835144646140236, "anger": 16.543765780743946, "fear":
17.094725979629427}
```

```
joy

Input:  wow/khadgar_joy_3.wav
{"joy": 16.098360634136977, "neutral": 17.227850069376842, "sadness":
14.416841114088918, "disgust": 18.374600117238618, "anger": 17.047304906672075, "fear":
16.835043158486563}
disgust


Input:  wow/khadgar_joy_5.wav
{"joy": 16.008510245853603, "neutral": 12.444916118750198, "sadness":
17.538985911750601, "disgust": 18.316919469460597, "anger": 20.079857452348037, "fear":
15.610810801836971}
anger

Input:  wow/voldemort_joy_1.wav
{"joy": 23.298929799334971, "neutral": 14.255030324571992, "sadness":
14.607596251010728, "disgust": 10.198686485271438, "anger": 23.003846530019157, "fear":
14.635910609791729}
joy

Correct class:  sadness
Input:  wow/diggory_sadness_1.wav
{"joy": 10.327022660068371, "neutral": 15.896629735971134, "sadness":
16.318985900479824, "disgust": 19.246359001290685, "anger": 25.594945168525172, "fear":
12.616057533664819}
anger

Input:  wow/harry_sadness_1.wav
{"joy": 16.004528861054713, "neutral": 11.373905083091271, "sadness":
22.937980546900299, "disgust": 17.118074599422695, "anger": 15.26247750387166, "fear":
17.303033405659363}
sadness

Input:  wow/harry_sadness_2.wav
{"joy": 17.272842058264281, "neutral": 15.268206836784854, "sadness":
24.807880079767919, "disgust": 15.07390254602614, "anger": 9.8312107217471532, "fear":
17.745957757409659}
sadness

Input:  wow/harry_sadness_3.wav
{"joy": 15.082418503543227, "neutral": 11.5391079088479, "sadness": 22.768016928541666,
"disgust": 11.925771733158411, "anger": 21.733819489224693, "fear": 16.95086543668409}
sadness

Input:  wow/khadgar_sadness_1.wav
{"joy": 16.614191932649664, "neutral": 10.761967202273295, "sadness":
21.109157803294057, "disgust": 11.774294272173284, "anger": 22.497238116037003, "fear":
17.2431506735727}
anger

Correct class:  fear
Input:  wow/harry_fear_1.wav
{"joy": 12.894591145540778, "neutral": 13.517822458620973, "sadness":
20.451894453226085, "disgust": 17.898323583026066, "anger": 19.946507105054852, "fear":
15.290861254531269}
sadness

Input:  wow/khadgar_fear_1.wav
{"joy": 19.795277991726817, "neutral": 9.9976766097579119, "sadness":
14.331631329432806, "disgust": 16.938774453970431, "anger": 23.28092679915537, "fear":
15.655712815956662}
anger

-----
Score List:  {"anger": 87.5, "joy": 60.0, "sadness": 60.0, "fear": 0.0}
Total Score:  51.875
Time elapsed:  30.5167188644
```

B.2: Effect of exclusion of Mean Spectral Subband Centroids

```
KNN
Correct class:  anger
Input:  wow/fightclub_anger_1.wav
{"joy": 13.827171407789059, "neutral": 7.0519491342953886, "sadness":
11.655680489065864, "disgust": 13.176005486668606, "anger": 34.463952132536832, "fear":
19.825241349644248}
anger

Input:  wow/guldan_anger_1.wav
{"joy": 18.346679017212484, "neutral": 8.7712469565804891, "sadness":
24.062527570290158, "disgust": 14.488302199896969, "anger": 18.530642853058371, "fear":
15.800601402961533}
sadness

Input:  wow/khadgar_anger_1.wav
{"joy": 14.714363149277538, "neutral": 17.053399056391708, "sadness": 17.26824250850434,
"disgust": 14.720549319269317, "anger": 19.658518901377104, "fear": 16.584927065179997}
anger

Input:  wow/khadgar_anger_2.wav
{"joy": 15.702938923239797, "neutral": 12.504944143257477, "sadness":
18.173791293259093, "disgust": 17.603353237270529, "anger": 20.387169559116057, "fear":
15.627802843857046}
anger

Input:  wow/khadgar_anger_3.wav
{"joy": 13.943629584074927, "neutral": 18.860832846865879, "sadness":
14.443974462038504, "disgust": 18.456958370699557, "anger": 20.016006817904508, "fear":
14.278597918416629}
anger

Input:  wow/khadgar_anger_4.wav
{"joy": 14.150033773983118, "neutral": 15.297820169999367, "sadness":
16.830497506028589, "disgust": 16.03609571808763, "anger": 21.467093517207637, "fear":
16.218459314693671}
anger

Input:  wow/kiljaiden_anger_1.wav
{"joy": 18.908829387576439, "neutral": 9.497741761274586, "sadness": 16.951095914505316,
"disgust": 15.319265798589569, "anger": 23.849844553744031, "fear": 15.473222584310035}
anger

Input:  wow/kiljaiden_anger_2.wav
{"joy": 17.782218204688892, "neutral": 9.8933389339937161, "sadness": 17.17258183853394,
"disgust": 13.895625308768553, "anger": 25.464134701093247, "fear": 15.79210101292165}
anger

Correct class:  joy
Input:  wow/goblin_joy_1.wav
{"joy": 21.921366680146722, "neutral": 6.8054019939467851, "sadness":
15.555735632978655, "disgust": 16.779724451888903, "anger": 23.43851698058366, "fear":
15.499254260455274}
anger

Input:  wow/khadgar_joy_1.wav
{"joy": 22.080896547647725, "neutral": 14.547955132655753, "sadness":
16.137526358748435, "disgust": 13.102543492934824, "anger": 16.888427567842779, "fear":
17.242650900170485}
joy

Input:  wow/khadgar_joy_3.wav
{"joy": 14.558636727687075, "neutral": 17.586763612488863, "sadness":
14.717191970632436, "disgust": 18.757404286347757, "anger": 17.402457092227746, "fear":
16.97754631061612}
disgust
```

```
Input:  wow/khadgar_joy_5.wav
{"joy": 14.466879506746558, "neutral": 12.704185204557493, "sadness": 17.90438145157874,
"disgust": 18.698521958407692, "anger": 20.498187815938621, "fear": 15.727844062770906}
anger

Input:  wow/voldemort_joy_1.wav
{"joy": 21.90930937385005, "neutral": 14.552010123000576, "sadness": 14.911921172906785,
"disgust": 10.41115912038126, "anger": 23.48309333272789, "fear": 14.732506877133453}
anger

Correct class:  sadness
Input:  wow/diggory_sadness_1.wav
{"joy": 8.6670918095579594, "neutral": 16.2278095221372, "sadness": 16.658964773406488,
"disgust": 19.647324813817573, "anger": 26.128173192869447, "fear": 12.67063588821134}
anger

Input:  wow/harry_sadness_1.wav
{"joy": 14.462868694259095, "neutral": 11.610861438989007, "sadness":
23.415855141627389, "disgust": 17.474701153577335, "anger": 15.58044578520232, "fear":
17.455267786344855}
sadness

Input:  wow/harry_sadness_2.wav
{"joy": 15.757640414239733, "neutral": 15.586294479217871, "sadness":
25.324710914763084, "disgust": 15.387942182401686, "anger": 10.036027611783553, "fear":
17.907384397594083}
sadness

Input:  wow/harry_sadness_3.wav
{"joy": 13.521554611654446, "neutral": 11.779505990282232, "sadness":
23.242350614552951, "disgust": 12.174225310932544, "anger": 22.186607395250206, "fear":
17.095756077327604}
sadness

Input:  wow/khadgar_sadness_1.wav
{"joy": 15.085168347747292, "neutral": 10.986174852320657, "sadness":
21.548931924196015, "disgust": 12.019592069510226, "anger": 22.965930576787773, "fear":
17.394202229438033}
anger

Correct class:  fear
Input:  wow/harry_fear_1.wav
{"joy": 11.288138818821317, "neutral": 13.799443759842241, "sadness":
20.877975587668292, "disgust": 18.271205324339103, "anger": 20.362059336410159, "fear":
15.401177172918894}
sadness


Input:  wow/khadgar_fear_1.wav
{"joy": 18.332534490948227, "neutral": 10.205961539127868, "sadness":
14.630206982129323, "disgust": 17.291665588428149, "anger": 23.765946107471109, "fear":
15.773685291895326}
anger


-----
Score List:  {"anger": 87.5, "joy": 20.0, "sadness": 60.0, "fear": 0.0}
Total Score:  41.875
Time elapsed:  30.0550808907
```

B.3: Experimental Dataset, KNN, Without Delta and Delta Delta Features

```
KNN
Correct class:  anger
Input:  wow/fightclub_anger_1.wav
{"joy": 11.300455271749129, "neutral": 6.8630527144598954, "sadness":
14.249749921878283, "disgust": 9.744095274857699, "anger": 39.952512837282093, "fear":
17.890133979772905}
anger


Input:  wow/guldan_anger_1.wav
{"joy": 18.204648358049901, "neutral": 19.766108157125576, "sadness":
21.659541294897174, "disgust": 5.3249211277178103, "anger": 21.265895714223554, "fear":
13.778885347985995}
sadness


Input:  wow/khadgar_anger_1.wav
{"joy": 21.588356586298691, "neutral": 14.211644015102113, "sadness":
16.366548905088475, "disgust": 12.380485705368216, "anger": 22.051448513142482, "fear":
13.401516275000017}
anger


Input:  wow/khadgar_anger_2.wav
{"joy": 18.061778591959953, "neutral": 13.097854837448768, "sadness":
16.799641896365934, "disgust": 9.8952057181897786, "anger": 24.902056476200432, "fear":
17.243462479835134}
anger


Input:  wow/khadgar_anger_3.wav
{"joy": 14.167359766824427, "neutral": 20.865944463630058, "sadness":
12.343152838224775, "disgust": 11.563925544034111, "anger": 24.133090495800641, "fear":
16.926526891485977}
anger


Input:  wow/khadgar_anger_4.wav
{"joy": 17.059807499437149, "neutral": 17.15204014790439, "sadness": 17.249235683407953,
"disgust": 13.940128964929984, "anger": 17.710626364466638, "fear": 16.888161339853898}
anger


Input:  wow/kiljaiden_anger_1.wav
{"joy": 14.538528831275348, "neutral": 24.024433705862641, "sadness": 16.09516224690508,
"disgust": 7.2895833573620203, "anger": 21.018360528500946, "fear": 17.033931330093964}
neutral


Input:  wow/kiljaiden_anger_2.wav
{"joy": 15.422873346705822, "neutral": 21.066383009159598, "sadness":
17.346151962580578, "disgust": 4.57742137590441, "anger": 26.252555704186697, "fear":
15.334614601462889}
anger


Correct class:  joy
Input:  wow/goblin_joy_1.wav
{"joy": 15.626128964215649, "neutral": 13.853332188651081, "sadness":
17.545850554446076, "disgust": 10.664582154851415, "anger": 30.018584852463182, "fear":
12.291521285372594}
anger

Input:  wow/khadgar_joy_1.wav
{"joy": 24.821944442320714, "neutral": 17.294279191532013, "sadness":
11.326920344138834, "disgust": 17.480670821963781, "anger": 17.319747516854484, "fear":
11.756437683190176}
```

```
joy

Input:  wow/khadgar_joy_3.wav
{"joy": 20.647722832353839, "neutral": 15.859636964620258, "sadness":
10.712010833772128, "disgust": 18.082657079273648, "anger": 16.666896110980005, "fear":
18.03107617900012}
joy

Input:  wow/khadgar_joy_5.wav
{"joy": 15.711157187046167, "neutral": 15.265601049541766, "sadness":
16.430370677309778, "disgust": 15.594067252417924, "anger": 21.117312986931672, "fear":
15.881490846752698}
anger

Input:  wow/voldemort_joy_1.wav
{"joy": 16.152010546099916, "neutral": 20.202424466419618, "sadness":
16.697122871747002, "disgust": 9.7748561480420477, "anger": 23.348687627447138, "fear":
13.82489834024426}
anger

Correct class:  sadness
Input:  wow/diggory_sadness_1.wav
{"joy": 21.200731323877662, "neutral": 13.526460768877165, "sadness":
12.874473418656276, "disgust": 11.800273557644772, "anger": 24.116156789386405, "fear":
16.481904141557713}
anger

Input:  wow/harry_sadness_1.wav
{"joy": 11.686400903456882, "neutral": 14.182971798916414, "sadness":
18.789602842650694, "disgust": 15.242376360103576, "anger": 20.208724762525527, "fear":
19.88992333234691}
anger

Input:  wow/harry_sadness_2.wav
{"joy": 23.808202217119074, "neutral": 22.68971065585502, "sadness": 17.464678067536116,
"disgust": 7.8564426995649539, "anger": 11.166722133019208, "fear": 17.014244226905635}
joy

Input:  wow/harry_sadness_3.wav
{"joy": 23.291947401565825, "neutral": 12.751153881301811, "sadness":
11.777260306856732, "disgust": 6.6872149316579579, "anger": 25.173073025972862, "fear":
20.319350452644809}
anger

Input:  wow/khadgar_sadness_1.wav
{"joy": 18.552496289869573, "neutral": 14.456451210480475, "sadness":
17.419034870545772, "disgust": 8.7365395984413627, "anger": 22.502182366413553, "fear":
18.333295664249256}
anger

Correct class:  fear
Input:  wow/harry_fear_1.wav
{"joy": 20.443754758726229, "neutral": 15.035697564185327, "sadness":
13.475351426558426, "disgust": 11.431763004836514, "anger": 21.377011880146764, "fear":
18.236421365546732}
anger

Input:  wow/khadgar_fear_1.wav
{"joy": 19.010876559619042, "neutral": 14.38860163146342, "sadness": 14.29554637705874,
"disgust": 10.923463801806207, "anger": 22.550437169083764, "fear": 18.831074460968825}
anger


-----
Score List:  {"anger": 75.0, "joy": 40.0, "sadness": 0.0, "fear": 0.0}
Total Score:  28.75
Time elapsed:  30.6105041504
```

B.4: Experimental Dataset, KNN, Non-normalised MFCC Features

```
KNN
Correct class:  anger
Input:  wow/fightclub_anger_1.wav
{"joy": 16.26395149070995, "neutral": 8.9452751374220103, "sadness": 12.238433022336958,
"disgust": 17.416974153107486, "anger": 26.705648795356122, "fear": 18.429717401067485}
anger

Input:  wow/guldan_anger_1.wav
{"joy": 21.937818067243612, "neutral": 10.839816496104376, "sadness":
21.996288913707637, "disgust": 9.1388284756647664, "anger": 18.865635116637666, "fear":
17.221612930641946}
sadness

Input:  wow/khadgar_anger_1.wav
{"joy": 13.969830520451124, "neutral": 17.14131144051581, "sadness": 17.141977339741178,
"disgust": 15.238874742156341, "anger": 17.146348462308307, "fear": 19.361657494827252}
fear

Input:  wow/khadgar_anger_2.wav
{"joy": 11.639653480190258, "neutral": 13.452236335943427, "sadness":
24.370419005669145, "disgust": 16.636016834855361, "anger": 19.461920280289075, "fear":
14.439754063052748}
sadness

Input:  wow/khadgar_anger_3.wav
{"joy": 12.981828712824246, "neutral": 10.496973523746304, "sadness":
14.384420252606155, "disgust": 10.307337178336839, "anger": 36.777026306260701, "fear":
15.052414026225742}
anger

Input:  wow/khadgar_anger_4.wav
{"joy": 14.068185021403679, "neutral": 14.880403620406984, "sadness":
18.076389921035158, "disgust": 15.819949932977933, "anger": 20.890599435512193, "fear":
16.264472068664066}
anger

Input:  wow/kiljaiden_anger_1.wav
{"joy": 15.319511527724691, "neutral": 10.323461662250891, "sadness":
19.115665995558622, "disgust": 11.837477074014624, "anger": 21.383612036423752, "fear":
22.020271704027412}
fear

Input:  wow/kiljaiden_anger_2.wav
{"joy": 14.634663370274044, "neutral": 9.5274257383606553, "sadness":
21.390642756946363, "disgust": 11.909045884551377, "anger": 23.087805262963972, "fear":
19.450416986903591}
anger

Correct class:  joy
Input:  wow/goblin_joy_1.wav
{"joy": 16.787292574744079, "neutral": 20.091646757965826, "sadness":
11.185147404424194, "disgust": 12.399805172769547, "anger": 27.515137921301545, "fear":
12.020970168794806}
anger

Input:  wow/khadgar_joy_1.wav
{"joy": 23.796876579792819, "neutral": 12.701150778100937, "sadness":
13.199906442994889, "disgust": 14.03095457701515, "anger": 20.795721896920764, "fear":
15.475389725175438}
joy

Input:  wow/khadgar_joy_3.wav
{"joy": 13.046224970906104, "neutral": 18.025679406706182, "sadness":
14.921484799033276, "disgust": 15.757600315173718, "anger": 19.726718968118668, "fear":
18.522291540062046}
anger
```

```
Input:  wow/khadgar_joy_5.wav
{"joy": 13.204103074091938, "neutral": 13.028740098787315, "sadness":
19.754772291254255, "disgust": 10.136374738376157, "anger": 29.099827281840909, "fear":
14.776182515649413}
anger

Input:  wow/voldemort_joy_1.wav
{"joy": 18.587239194306566, "neutral": 12.697626694700668, "sadness":
17.993214712061359, "disgust": 10.410757292188128, "anger": 26.417269525127921, "fear":
13.893892581615374}
anger

Correct class:  sadness
Input:  wow/diggory_sadness_1.wav
{"joy": 9.7337191713370874, "neutral": 16.75702074169099, "sadness": 13.131660151969832,
"disgust": 12.811068334492605, "anger": 34.590904641799121, "fear": 12.975626958710366}
anger

Input:  wow/harry_sadness_1.wav
{"joy": 13.952444324241123, "neutral": 20.805413204260805, "sadness":
19.506929525067193, "disgust": 11.327862758011568, "anger": 18.937889075967536, "fear":
15.469461112451768}
neutral

Input:  wow/harry_sadness_2.wav
{"joy": 15.935708575849915, "neutral": 15.297901912083178, "sadness":
26.510917805512872, "disgust": 11.169861087760323, "anger": 13.73167921540862, "fear":
17.353931403385108}
sadness

Input:  wow/harry_sadness_3.wav
{"joy": 27.268560672115619, "neutral": 13.603597601645106, "sadness":
10.183395939827239, "disgust": 9.9100944526022179, "anger": 23.513494343961426, "fear":
15.520856989848395}
joy

Input:  wow/khadgar_sadness_1.wav
{"joy": 11.381294542262992, "neutral": 12.019818469967795, "sadness":
25.180727300127426, "disgust": 9.4846736555560778, "anger": 25.912680210157603, "fear":
16.020805821928111}
anger

Correct class:  fear
Input:  wow/harry_fear_1.wav
{"joy": 14.123213992552792, "neutral": 14.321749270507915, "sadness":
20.035006068430729, "disgust": 12.061861107863814, "anger": 24.130828141002304, "fear":
15.327341419642449}
anger

Input:  wow/khadgar_fear_1.wav
{"joy": 16.573514036207616, "neutral": 9.840753247381361, "sadness": 15.393065802534261,
"disgust": 14.240467460623755, "anger": 24.252213611006891, "fear": 19.699985842246122}
anger

-----
Score List:  {"anger": 50.0, "joy": 20.0, "sadness": 20.0, "fear": 0.0}
Total Score:  22.5
Time elapsed:  30.9233160019
```

B.5: Effect of exclusion of max and min values of MFCC

```
KNN
Correct class:  anger
Input:  wow/fightclub_anger_1.wav
{"joy": 13.274499700958714, "neutral": 7.2019906052378451, "sadness":
12.967172411728292, "disgust": 13.456346028938153, "anger": 34.555561712369354, "fear":
18.544429540767648}
anger

Input:  wow/guldan_anger_1.wav
{"joy": 20.423021553915753, "neutral": 8.1087858015784171, "sadness":
23.516638055192338, "disgust": 14.366584067690161, "anger": 17.235495137858724, "fear":
16.349475383764613}
sadness

Input:  wow/khadgar_anger_1.wav
{"joy": 14.827970802927444, "neutral": 17.006231595390716, "sadness":
17.211448197913874, "disgust": 14.569792335937723, "anger": 19.443442505522682, "fear":
16.941114562307551}
anger

Input:  wow/khadgar_anger_2.wav
{"joy": 16.458059128565193, "neutral": 12.35307472100995, "sadness": 18.136823809800081,
"disgust": 17.113744883763783, "anger": 19.765320500508647, "fear": 16.172976956352347}
anger

Input:  wow/khadgar_anger_3.wav
{"joy": 15.077002527237124, "neutral": 18.213688697337815, "sadness":
14.322431940267471, "disgust": 18.402319677342913, "anger": 19.401835072123774, "fear":
14.582722085690916}
anger

Input:  wow/khadgar_anger_4.wav
{"joy": 14.471780880928904, "neutral": 15.207650493295761, "sadness":
16.325038871295195, "disgust": 15.917992251822152, "anger": 21.511893232543503, "fear":
16.565644270114472}
anger

Input:  wow/kiljaiden_anger_1.wav
{"joy": 21.015229526654732, "neutral": 8.6475189914956072, "sadness":
15.812911665903806, "disgust": 15.405066131838941, "anger": 23.104140775780774, "fear":
16.015132908326127}
anger

Input:  wow/kiljaiden_anger_2.wav
{"joy": 19.878762997249225, "neutral": 9.2502146248649861, "sadness": 15.63827025699147,
"disgust": 13.942902874939108, "anger": 24.949045495629473, "fear": 16.34080375032574}
anger

Correct class:  joy
Input:  wow/goblin_joy_1.wav
{"joy": 23.243383650024182, "neutral": 5.8991767907203458, "sadness":
15.467668917874363, "disgust": 16.480850033827405, "anger": 23.075702808752407, "fear":
15.833217798801291}
joy

Input:  wow/khadgar_joy_1.wav
{"joy": 22.763213488073816, "neutral": 14.236144581589151, "sadness":
16.045656445433529, "disgust": 12.922026545270359, "anger": 16.41512111997487, "fear":
17.617837819658291}
joy

Input:  wow/khadgar_joy_3.wav
{"joy": 14.00735307499542, "neutral": 17.756610862297094, "sadness": 14.179081199457942,
"disgust": 18.07893078262677, "anger": 16.723665634670883, "fear": 19.254358445951897}
fear

Input:  wow/khadgar_joy_5.wav
{"joy": 15.195825900643358, "neutral": 12.76394152033378, "sadness": 17.854145424235945,
"disgust": 18.02832425952473, "anger": 19.882662272070448, "fear": 16.275100623191733}
```

```
anger


Input:  wow/voldemort_joy_1.wav
{"joy": 21.927534124760719, "neutral": 14.228542917510234, "sadness":
15.021428165365176, "disgust": 10.63267314421916, "anger": 23.353340405104856, "fear":
14.836481243039865}
anger

Correct class:  sadness
Input:  wow/diggory_sadness_1.wav
{"joy": 8.4121128619707015, "neutral": 16.149824798990416, "sadness":
16.801971226671768, "disgust": 20.065353001345603, "anger": 25.632108013170662, "fear":
12.938630097850844}
anger

Input:  wow/harry_sadness_1.wav
{"joy": 14.774121806104874, "neutral": 11.435387573390454, "sadness":
23.264989016304167, "disgust": 17.636516478269908, "anger": 15.277517935429334, "fear":
17.611467190501273}
sadness

Input:  wow/harry_sadness_2.wav
{"joy": 16.509221231957969, "neutral": 14.651806175408558, "sadness":
24.583916741569755, "disgust": 15.715345207559171, "anger": 10.249560114161927, "fear":
18.290150529342629}
sadness

Input:  wow/harry_sadness_3.wav
{"joy": 13.594104839225828, "neutral": 12.030133777309516, "sadness":
23.525757215778988, "disgust": 11.795947567965644, "anger": 21.806544313682565, "fear":
17.247512286037459}
sadness

Input:  wow/khadgar_sadness_1.wav
{"joy": 16.270400772082194, "neutral": 11.012185331293937, "sadness": 21.78867300985355,
"disgust": 11.815403881154124, "anger": 21.341566254170097, "fear": 17.771770751446095}
sadness

Correct class:  fear
Input:  wow/harry_fear_1.wav
{"joy": 10.883425438682176, "neutral": 13.462022074449488, "sadness":
20.673670938585083, "disgust": 18.659954373793127, "anger": 20.795294641440162, "fear":
15.525632533049977}
anger

Input:  wow/khadgar_fear_1.wav
{"joy": 18.727433574004511, "neutral": 10.21904589890479, "sadness": 14.726655038462058,
"disgust": 17.20102280707232, "anger": 23.009667819139448, "fear": 16.116174862416873}
anger

-----
Score List:  {"anger": 87.5, "joy": 40.0, "sadness": 80.0, "fear": 0.0}
Total Score:  51.875
Time elapsed:  32.356194973
```

Appendix C: Experimental Test Results - SVM & DT

C.1: Decision Tree

```
DT
Correct class:  anger
Input:  wow/fightclub_anger_1.wav
{"joy": 0.16, "neutral": 0.12, "sadness": 0.16, "disgust": 0.10000000000000001, "anger":
0.22, "fear": 0.22}
anger


Input:  wow/guldan_anger_1.wav
{"joy": 0.080000000000000002, "neutral": 0.16, "sadness": 0.32000000000000001,
"disgust": 0.10000000000000001, "anger": 0.14000000000000001, "fear":
0.17999999999999999}
sadness


Input:  wow/khadgar_anger_1.wav
{"joy": 0.12, "neutral": 0.12, "sadness": 0.20000000000000001, "disgust": 0.22, "anger":
0.17999999999999999, "fear": 0.14000000000000001}
disgust


Input:  wow/khadgar_anger_2.wav
{"joy": 0.14000000000000001, "neutral": 0.16, "sadness": 0.059999999999999998,
"disgust": 0.29999999999999999, "anger": 0.22, "fear": 0.10000000000000001}
disgust


Input:  wow/khadgar_anger_3.wav
{"joy": 0.16, "neutral": 0.14000000000000001, "sadness": 0.12, "disgust":
0.14000000000000001, "anger": 0.29999999999999999, "fear": 0.12}
anger


Input:  wow/khadgar_anger_4.wav
{"joy": 0.22, "neutral": 0.10000000000000001, "sadness": 0.26000000000000001, "disgust":
0.16, "anger": 0.14000000000000001, "fear": 0.10000000000000001}
sadness


Input:  wow/kiljaiden_anger_1.wav
{"joy": 0.17999999999999999, "neutral": 0.040000000000000001, "sadness": 0.16,
"disgust": 0.29999999999999999, "anger": 0.23999999999999999, "fear":
0.059999999999999998}
disgust


Input:  wow/kiljaiden_anger_2.wav
{"joy": 0.20000000000000001, "neutral": 0.10000000000000001, "sadness":
0.10000000000000001, "disgust": 0.20000000000000001, "anger": 0.29999999999999999,
"fear": 0.080000000000000002}
anger


Correct class:  joy
Input:  wow/goblin_joy_1.wav
{"joy": 0.080000000000000002, "neutral": 0.040000000000000001, "sadness":
0.23999999999999999, "disgust": 0.12, "anger": 0.32000000000000001, "fear":
0.17999999999999999}
anger


Input:  wow/khadgar_joy_1.wav
{"joy": 0.12, "neutral": 0.080000000000000002, "sadness": 0.16, "disgust": 0.12,
"anger": 0.32000000000000001, "fear": 0.17999999999999999}
anger
```

```
Input:  wow/khadgar_joy_3.wav
{"joy": 0.23999999999999999, "neutral": 0.20000000000000001, "sadness": 0.12, "disgust":
0.10000000000000001, "anger": 0.14000000000000001, "fear": 0.17999999999999999}
joy


Input:  wow/khadgar_joy_5.wav
{"joy": 0.17999999999999999, "neutral": 0.22, "sadness": 0.12, "disgust": 0.12, "anger":
0.23999999999999999, "fear": 0.10000000000000001}
anger


Input:  wow/voldemort_joy_1.wav
{"joy": 0.20000000000000001, "neutral": 0.16, "sadness": 0.14000000000000001, "disgust":
0.22, "anger": 0.12, "fear": 0.14000000000000001}
disgust


Correct class:  sadness
Input:  wow/diggory_sadness_1.wav
{"joy": 0.12, "neutral": 0.16, "sadness": 0.12, "disgust": 0.10000000000000001, "anger":
0.32000000000000001, "fear": 0.16}
anger


Input:  wow/harry_sadness_1.wav
{"joy": 0.26000000000000001, "neutral": 0.14000000000000001, "sadness": 0.16, "disgust":
0.10000000000000001, "anger": 0.26000000000000001, "fear": 0.059999999999999998}
joy


Input:  wow/harry_sadness_2.wav
{"joy": 0.22, "neutral": 0.17999999999999999, "sadness": 0.12, "disgust": 0.22, "anger":
0.059999999999999998, "fear": 0.17999999999999999}
joy


Input:  wow/harry_sadness_3.wav
{"joy": 0.16, "neutral": 0.16, "sadness": 0.12, "disgust": 0.14000000000000001, "anger":
0.20000000000000001, "fear": 0.20000000000000001}
anger


Input:  wow/khadgar_sadness_1.wav
{"joy": 0.16, "neutral": 0.17999999999999999, "sadness": 0.16, "disgust":
0.14000000000000001, "anger": 0.14000000000000001, "fear": 0.20000000000000001}
fear


Correct class:  fear
Input:  wow/harry_fear_1.wav
{"joy": 0.059999999999999998, "neutral": 0.35999999999999999, "sadness": 0.12,
"disgust": 0.080000000000000002, "anger": 0.20000000000000001, "fear": 0.16}
neutral


Input:  wow/khadgar_fear_1.wav
{"joy": 0.16, "neutral": 0.14000000000000001, "sadness": 0.080000000000000002,
"disgust": 0.12, "anger": 0.16, "fear": 0.32000000000000001}
fear


-----
Score List:  {"anger": 37.5, "joy": 20.0, "sadness": 0.0, "fear": 50.0}
Total Score:  26.875
Time elapsed:  49.6053609848
```

C.2: SVM

```
Loaded Classifier: toronto_svm_20_crossfold
Using:  SVM
Correct class:  anger
Input:  wow/fightclub_anger_1.wav
{"joy": 16.722133185959397, "neutral": 7.2333715162788197, "sadness": 13.07484323625282,
"disgust": 14.995426771477263, "anger": 32.611581387964918, "fear": 15.362643902066791}
anger

Input:  wow/guldan_anger_1.wav
{"joy": 18.846810450970963, "neutral": 8.160584198966907, "sadness": 25.982019464223093,
"disgust": 16.997404623299794, "anger": 12.623133909402558, "fear": 17.390047353136698}
sadness

Input:  wow/khadgar_anger_1.wav
{"joy": 11.350726947714653, "neutral": 11.949345862482113, "sadness":
20.636767577467364, "disgust": 18.907094846113633, "anger": 20.522130027075779, "fear":
16.633934739146468}
sadness

Input:  wow/khadgar_anger_2.wav
{"joy": 13.747479643274131, "neutral": 10.039928172576746, "sadness":
22.184569608428863, "disgust": 23.07931489622792, "anger": 14.30853537088192, "fear":
16.640172308610421}
disgust

Input:  wow/khadgar_anger_3.wav
{"joy": 14.511373349282174, "neutral": 13.083609281994573, "sadness":
17.549476037324148, "disgust": 23.545280425555575, "anger": 15.586936420422514, "fear":
15.723324485421012}
disgust

Input:  wow/khadgar_anger_4.wav
{"joy": 18.15279979840723, "neutral": 13.418432003523483, "sadness": 16.173715748324206,
"disgust": 18.855357948515579, "anger": 14.165758200810556, "fear": 19.233936300418943}
fear

Input:  wow/kiljaiden_anger_1.wav
{"joy": 24.238550727140623, "neutral": 10.565019689100337, "sadness":
15.522945339887768, "disgust": 14.726700743870291, "anger": 15.752333045737531, "fear":
19.194450454263457}
joy

Input:  wow/kiljaiden_anger_2.wav
{"joy": 21.326964858088083, "neutral": 9.5346697156195308, "sadness":
14.056413727109909, "disgust": 14.376955817468687, "anger": 24.528164352386668, "fear":
16.176831529327124}
anger

Correct class:  joy
Input:  wow/goblin_joy_1.wav
{"joy": 27.488505411452099, "neutral": 11.04968511094622, "sadness": 15.128118866803993,
"disgust": 14.440774134244617, "anger": 17.934698244274145, "fear": 13.958218232278931}
joy

Input:  wow/khadgar_joy_1.wav
{"joy": 18.851842062951931, "neutral": 11.962982131075325, "sadness": 12.14844078379098,
"disgust": 20.839340128411482, "anger": 17.433648035048943, "fear": 18.76374685872133}
disgust

Input:  wow/khadgar_joy_3.wav
{"joy": 8.6705570018909714, "neutral": 18.496705896168606, "sadness":
20.081194779726353, "disgust": 21.544986129441451, "anger": 14.650618722632831, "fear":
16.555937470139767}
disgust

Input:  wow/khadgar_joy_5.wav
{"joy": 11.958898720605589, "neutral": 11.920236656828882, "sadness":
21.002494652492977, "disgust": 20.455137050000623, "anger": 15.478542787988529, "fear":
19.184690132083404}
```

```
sadness

Input:  wow/voldemort_joy_1.wav
{"joy": 15.001094026550462, "neutral": 12.198136257971223, "sadness":
21.272857919328604, "disgust": 18.900540480679535, "anger": 15.61101457536658, "fear":
17.016356740103596}
sadness

Correct class:  sadness
Input:  wow/diggory_sadness_1.wav
{"joy": 9.9912435555523427, "neutral": 12.547041409164263, "sadness": 23.28973653672471,
"disgust": 26.230529334558188, "anger": 13.385143331381007, "fear": 14.556305832619485}
disgust

Input:  wow/harry_sadness_1.wav
{"joy": 10.664479755155758, "neutral": 12.118164153464178, "sadness":
28.836615565949021, "disgust": 20.347446386526887, "anger": 12.080071516506255, "fear":
15.953222622397909}
sadness

Input:  wow/harry_sadness_2.wav
{"joy": 14.98770206331522, "neutral": 11.237948170794326, "sadness": 30.375620243128157,
"disgust": 17.014844679748023, "anger": 9.7068911822180439, "fear": 16.676993660796239}
sadness


Input:  wow/harry_sadness_3.wav
{"joy": 12.234519335161821, "neutral": 13.191833085411501, "sadness":
25.095263337556421, "disgust": 17.504049314542616, "anger": 15.717254697475243, "fear":
16.257080229852399}
sadness

Input:  wow/khadgar_sadness_1.wav
{"joy": 13.271101509174816, "neutral": 12.448012110851028, "sadness":
25.978494390218458, "disgust": 14.625024233032542, "anger": 13.395388607466364, "fear":
20.281979149256792}
sadness

Correct class:  fear
Input:  wow/harry_fear_1.wav
{"joy": 12.022185779579614, "neutral": 9.843178431504656, "sadness": 24.078513094697559,
"disgust": 21.612690903437187, "anger": 17.118104781012693, "fear": 15.325327009768285}
sadness

Input:  wow/khadgar_fear_1.wav
{"joy": 13.157843804558013, "neutral": 10.977140498710213, "sadness":
17.224331530721976, "disgust": 23.89248654741051, "anger": 16.888704199897067, "fear":
17.859493418702229}
disgust

-----
Score List:  {"anger": 25.0, "joy": 20.0, "sadness": 80.0, "fear": 0.0}
Total Score:  31.25
Time elapsed:  34.0987598896
```

Appendix D: Test Classification Results

D.1: Speech to Text Translation Results

```
wow/diggory_sadness_1.wav
Google Speech Recognition thinks you said the first song
wow/fightclub_anger_1.wav
Google Speech Recognition thinks you said Playa Victoria
wow/goblin_joy_1.wav
Google Speech Recognition thinks you said Peppa Pig
wow/guldan_anger_1.wav
Google Speech Recognition thinks you said I have pulled my loyalty 1000 times over
wow/harry_fear_1.wav
Google Speech Recognition thinks you said foldable
wow/harry_sadness_1.wav
Google Speech Recognition thinks you said hello
wow/harry_sadness_2.wav
Google Speech Recognition thinks you said can you back the money
wow/harry_sadness_3.wav
Google Speech Recognition could not understand audio
wow/khadgar_anger_1.wav
Google Speech Recognition thinks you said where is Eid
wow/khadgar_anger_2.wav
Google Speech Recognition thinks you said this way yo yo yo
wow/khadgar_anger_3.wav
Google Speech Recognition thinks you said 1 kg
wow/khadgar_anger_4.wav
Google Speech Recognition thinks you said give yourself a
wow/khadgar_fear_1.wav
Google Speech Recognition thinks you said it's not working
wow/khadgar_joy_1.wav
Google Speech Recognition thinks you said this will only hurt
wow/khadgar_joy_3.wav
Google Speech Recognition thinks you said this magic is extremely reliable
wow/khadgar_joy_5.wav
Google Speech Recognition thinks you said I do parties in fact I am the party
wow/khadgar_sadness_1.wav
Google Speech Recognition thinks you said IMC group
wow/kiljaiden_anger_1.wav
Google Speech Recognition thinks you said disappeared
wow/kiljaiden_anger_2.wav
Google Speech Recognition could not understand audio
wow/voldemort_joy_1.wav
Google Speech Recognition thinks you said Harry Potter is dead
-----
```

D.2: Text Classification Results

```
KNN, k = 10, 1 gram

the first song
{"anger": 11.43255709143472, "joy": 28.929786042130196, "fear": 9.6443638800994016,
"sadness": 49.993292986335689, "disgust": 0.0}
sadness

Peppa Pig
{"anger": 19.705592377728181, "joy": 0.0, "fear": 18.399663966953604, "sadness":
38.186865463202601, "disgust": 23.707878192115622}
sadness

I have pulled my loyalty 1000 times over
{"anger": 10.247996311755294, "joy": 0.0, "fear": 49.975369471659697, "sadness":
9.9013509736351448, "disgust": 29.875283242949855}
fear

foldable
{"anger": 10.0, "joy": 0.0, "fear": 30.000000000000004, "sadness": 40.0, "disgust":
20.0}
sadness

Hello
{"anger": 9.5433710543779533, "joy": 0.0, "fear": 42.739773673732287, "sadness":
28.630113163133856, "disgust": 19.086742108755907}
fear

can you back the money
{"anger": 9.5764322908193833, "joy": 39.968777324280332, "fear": 9.9936250864119369,
"sadness": 30.456918591911514, "disgust": 10.004246706576838}
joy

where is Eid
{"anger": 10.204050483070079, "joy": 19.636830540740348, "fear": 29.796516020892096,
"sadness": 10.771930203522125, "disgust": 29.590672751775344}
fear

this way yo yo yo
{"anger": 19.754054336275541, "joy": 20.612055462177711, "fear": 29.905697444937481,
"sadness": 9.8612157926116808, "disgust": 19.866976963997594}
fear

1 kg
{"anger": 9.7986659251586961, "joy": 0.0, "fear": 29.395997775476086, "sadness":
29.395997775476086, "disgust": 31.409338523889129}
disgust

give yourself a
{"anger": 30.361303620060703, "joy": 9.6841886795347367, "fear": 9.7716655841888294,
"sadness": 30.390364947266836, "disgust": 19.792477168948899}
sadness

it's not working
{"anger": 19.941241444576836, "joy": 20.531862234829195, "fear": 29.277083766374805,
"sadness": 19.44092410557904, "disgust": 10.808888448640131}
fear

this will only hurt
{"anger": 19.405645931486696, "joy": 0.0, "fear": 19.898119676986717, "sadness":
50.557436981261873, "disgust": 10.138797410264722}
sadness

this magic is extremely reliable
{"anger": 9.9900376148760781, "joy": 19.834491462740164, "fear": 30.549272300162439,
"sadness": 9.8797764463369031, "disgust": 29.746422175884408}
fear
```

```
I do parties in fact I am the party
{"anger": 19.946218522632169, "joy": 9.9822335975676637, "fear": 0.0, "sadness":
49.830008874523024, "disgust": 20.241539005277144}
sadness

IMC group
{"anger": 9.8030581411827118, "joy": 30.361371900594104, "fear": 49.719260939247619,
"sadness": 0.0, "disgust": 10.116309018975569}
fear

disappeared
{"anger": 21.10378599185405, "joy": 9.3999567892466391, "fear": 59.983926386005948,
"sadness": 0.0, "disgust": 9.51233083289336}
fear

Harry Potter is dead
{"anger": 0.0, "joy": 0.0, "fear": 30.302784885296415, "sadness": 9.7162110117065996,
"disgust": 59.981004102996984}
disgust

Playa Victoria
{"anger": 19.102629158629323, "joy": 0.0, "fear": 19.102629158629323, "sadness":
42.692112524112041, "disgust": 19.102629158629323}
sadness

Time elapsed:  2.03899002075
```

KNN,k=10,2-gram

the first song
{"anger": 9.9827481923355119, "joy": 39.66561034756284, "fear": 20.244718061074682,
"sadness": 30.106923399026968, "disgust": 0.0}
joy

Peppa Pig
{"anger": 39.29260906655486, "joy": 19.292959519085532, "fear": 9.6464797595427658,
"sadness": 20.04699035913972, "disgust": 11.72096129567713}
anger

I have pulled my loyalty 1000 times over
{"anger": 19.973015286706566, "joy": 0.0, "fear": 40.122438442025022, "sadness":
9.9121956433746163, "disgust": 29.992350627893792}
fear

foldable
{"anger": 30.0, "joy": 29.999999999999993, "fear": 9.9999999999999982, "sadness":
19.999999999999996, "disgust": 9.9999999999999982}
anger

Hello
{"anger": 29.318266423269318, "joy": 29.318266423269318, "fear": 31.590711679038257,
"sadness": 9.7727554744231053, "disgust": 0.0}
fear

can you back the money
{"anger": 10.238994495600643, "joy": 19.90315124098062, "fear": 29.866699929449304,
"sadness": 20.098876253774353, "disgust": 19.892278080195073}
fear

where is Eid
{"anger": 20.030594780118093, "joy": 9.947268974613781, "fear": 29.981612451407813,
"sadness": 10.237393450472661, "disgust": 29.80313034338765}
fear

this way yo yo yo
{"anger": 20.293310818455389, "joy": 19.85692926283776, "fear": 29.913332202381493,
"sadness": 20.132822086569661, "disgust": 9.8036056297556993}
fear

1 kg
{"anger": 29.678339203539739, "joy": 29.678339203539739, "fear": 9.8927797345132458,
"sadness": 9.8927797345132458, "disgust": 20.857762123894034}
anger

give yourself a
{"anger": 30.283405128003039, "joy": 0.0, "fear": 9.8417894980417149, "sadness":
39.983807010507142, "disgust": 19.890998363448119}
sadness

it's not working
{"anger": 10.045041862075824, "joy": 20.22477453927047, "fear": 39.813901885902318,
"sadness": 9.9040661187513415, "disgust": 20.012215594000043}
fear

this will only hurt
{"anger": 0.0, "joy": 0.0, "fear": 29.805699435425197, "sadness": 60.156437434463896,
"disgust": 10.03786313011091}
sadness

```
this magic is extremely reliable
{"anger": 10.00962818731035, "joy": 9.9660757331318415, "fear": 40.191695766395,
"sadness": 9.9323774839715835, "disgust": 29.900222829191232}
fear


I do parties in fact I am the party
{"anger": 40.063722259759167, "joy": 0.0, "fear": 0.0, "sadness": 29.724958334936375,
"disgust": 30.211319405304454}
anger


IMC group
{"anger": 9.8962886228911717, "joy": 29.934448172667615, "fear": 50.1349901212959,
"sadness": 0.0, "disgust": 10.034273083145306}
fear


disappeared
{"anger": 20.474015608793625, "joy": 19.517718828846792, "fear": 50.23504069045903,
"sadness": 0.0, "disgust": 9.773224871900549}
fear


Harry Potter is dead
{"anger": 0.0, "joy": 0.0, "fear": 30.670498803304781, "sadness": 9.7943129542071237,
"disgust": 59.535188242488104}
disgust


Playa Victoria
{"anger": 29.372240593518601, "joy": 29.372240593518601, "fear": 19.581493729012401,
"sadness": 21.674025083950404, "disgust": 0.0}
anger


Time elapsed:  2.36413192749
```

Appendix E: Hybrid Classification Results

Hybrid classification: KNN&KNN (2-gram)

```
python STE.py -s True -t True
KNN
Correct class:  anger
Input:  wow/fightclub_anger_1.wav
{"joy": 13.544984236201529, "neutral": 6.9080318050240557, "sadness":
12.437900068392445, "disgust": 12.907107415512105, "anger": 34.78133188746326, "fear":
19.420644587406613}
anger
Playa Victoria
{"anger": 29.372240593518601, "joy": 29.372240593518601, "fear": 19.581493729012401,
"sadness": 21.674025083950404, "disgust": 0.0}
anger
{"joy": 18.293161143396652, "sadness": 15.208737573059832, "neutral":
4.8356222635168384, "disgust": 9.0349751908584732, "anger": 33.158604499279861, "fear":
19.468899329888348}
anger


Input:  wow/guldan_anger_1.wav
{"joy": 19.80906361492686, "neutral": 8.5922419166502753, "sadness": 23.571455579059744,
"disgust": 14.192622563164377, "anger": 18.15246646830208, "fear": 15.68214985789667}
sadness
I have pulled my loyalty 1000 times over
{"anger": 19.973015286706566, "joy": 0.0, "fear": 40.122438442025022, "sadness":
9.9121956433746163, "disgust": 29.992350627893792}
fear
{"joy": 13.866344530448801, "sadness": 19.473677598354207, "neutral": 6.014569341655192,
"disgust": 18.932540982583198, "anger": 18.698631113823424, "fear": 23.014236433135174}
sadness


Input:  wow/khadgar_anger_1.wav
{"joy": 16.250891067802627, "neutral": 16.705370504220451, "sadness":
16.915829396085883, "disgust": 14.420129945406678, "anger": 19.257324638083691, "fear":
16.450454448400674}
anger
where is Eid
{"anger": 20.030594780118093, "joy": 9.947268974613781, "fear": 29.981612451407813,
"sadness": 10.237393450472661, "disgust": 29.80313034338765}
fear
{"joy": 14.359804439845972, "sadness": 14.912298612401916, "neutral":
11.693759352954315, "disgust": 19.035030064800971, "anger": 19.489305680694009, "fear":
20.509801849302814}
anger


Input:  wow/khadgar_anger_2.wav
{"joy": 17.21929975249159, "neutral": 12.249741201558345, "sadness": 17.802897593396661,
"disgust": 17.244101130387456, "anger": 19.97110487423614, "fear": 15.512855447929802}
anger
this way yo yo yo
{"anger": 20.293310818455389, "joy": 19.85692926283776, "fear": 29.913332202381493,
"sadness": 20.132822086569661, "disgust": 9.8036056297556993}
fear
{"joy": 18.010588605595441, "sadness": 18.501874941348561, "neutral":
8.5748188410908419, "disgust": 15.011952480197928, "anger": 20.067766657501913, "fear":
19.83299847426531}
anger


Input:  wow/khadgar_anger_3.wav
{"joy": 15.495881022243697, "neutral": 18.475917890807391, "sadness": 14.14919947301731,
"disgust": 18.08028575088936, "anger": 19.607516882845232, "fear": 14.19119898019701}
anger
1 kg
{"anger": 29.678339203539739, "joy": 29.678339203539739, "fear": 9.8927797345132458,
"sadness": 9.8927797345132458, "disgust": 20.857762123894034}
```

```
anger
{"joy": 19.750618476632511, "sadness": 12.87227355146609, "neutral": 12.933142523565174,
"disgust": 18.91352866279076, "anger": 22.628763579053583, "fear": 12.90167320649188}
anger


Input:  wow/khadgar_anger_4.wav
{"joy": 15.698095307927014, "neutral": 14.985619758366724, "sadness":
16.487017965089226, "disgust": 15.708828458534819, "anger": 21.028989567876867, "fear":
16.091448942205346}
anger
give yourself a
{"anger": 30.283405128003039, "joy": 0.0, "fear": 9.8417894980417149, "sadness":
39.983807010507142, "disgust": 19.890998363448119}
sadness
{"joy": 10.988666715548909, "sadness": 23.536054678714599, "neutral":
10.489933830856707, "disgust": 16.96347943000881, "anger": 23.805314235914715, "fear":
14.216551108956256}
anger


Input:  wow/kiljaiden_anger_1.wav
{"joy": 20.359739754269857, "neutral": 9.3039102967587812, "sadness":
16.605155181556231, "disgust": 15.006627721067337, "anger": 23.36311303223906, "fear":
15.361454014108737}
anger
disappeared
{"anger": 20.474015608793625, "joy": 19.517718828846792, "fear": 50.23504069045903,
"sadness": 0.0, "disgust": 9.773224871900549}
fear
{"joy": 20.107133476642936, "sadness": 11.623608627089361, "neutral":
6.5127372077311465, "disgust": 13.436606866317302, "anger": 22.496383805205429, "fear":
25.823530017013823}
anger


Input:  wow/kiljaiden_anger_2.wav
{"joy": 19.256113187346866, "neutral": 9.6914340577897615, "sadness":
16.822120984686308, "disgust": 13.612041118793686, "anger": 24.944458482703592, "fear":
15.673832168679793}
anger
Google Speech Recognition could not understand audio

{"anger": 30.0, "joy": 29.999999999999993, "fear": 9.9999999999999982, "sadness":
19.999999999999996, "disgust": 9.9999999999999982}
anger
{"joy": 22.479279231142804, "sadness": 17.775484689280415, "neutral":
6.7840038404528329, "disgust": 12.528428783155579, "anger": 26.461120937892513, "fear":
13.971682518075854}
anger


Correct class:  joy
Input:  wow/goblin_joy_1.wav
{"joy": 23.310817880889726, "neutral": 6.6665162389682795, "sadness":
15.238271640468886, "disgust": 16.437281095727904, "anger": 22.96017989934726, "fear":
15.386933244597937}
joy
Peppa Pig
{"anger": 39.29260906655486, "joy": 19.292959519085532, "fear": 9.6464797595427658,
"sadness": 20.04699035913972, "disgust": 11.72096129567713}
anger
{"joy": 22.105460372348468, "sadness": 16.680887256070136, "neutral":
4.6665613672777955, "disgust": 15.022385155712671, "anger": 27.859908649509538, "fear":
13.664797199081386}
joy


Input:  wow/khadgar_joy_1.wav
{"joy": 23.467116418233314, "neutral": 14.251058089132167, "sadness":
15.808189086120915, "disgust": 12.835144646140236, "anger": 16.543765780743946, "fear":
17.094725979629427}
```

```
joy
this will only hurt
{"anger": 0.0, "joy": 0.0, "fear": 29.805699435425197, "sadness": 60.156437434463896,
"disgust": 10.03786313011091}
sadness
{"joy": 16.426981492763318, "sadness": 29.112663590623811, "neutral": 9.975740662392516,
"disgust": 11.995960191331438, "anger": 11.580636046520763, "fear": 20.908018016368157}
joy


Input:  wow/khadgar_joy_3.wav
{"joy": 16.098360634136977, "neutral": 17.227850069376842, "sadness":
14.416841114088918, "disgust": 18.374600117238618, "anger": 17.047304906672075, "fear":
16.835043158486563}
disgust
this magic is extremely reliable
{"anger": 10.00962818731035, "joy": 9.9660757331318415, "fear": 40.191695766395,
"sadness": 9.9323774839715835, "disgust": 29.900222829191232}
fear
{"joy": 14.258675163835434, "sadness": 13.071502025053716, "neutral": 12.05949504856379,
"disgust": 21.832286930824402, "anger": 14.936001890863556, "fear": 23.842038940859091}
disgust


Input:  wow/khadgar_joy_5.wav
{"joy": 16.008510245853603, "neutral": 12.444916118750198, "sadness":
17.538985911750601, "disgust": 18.316919469460597, "anger": 20.079857452348037, "fear":
15.610810801836971}
anger
I do parties in fact I am the party
{"anger": 40.063722259759167, "joy": 0.0, "fear": 0.0, "sadness": 29.724958334936375,
"disgust": 30.211319405304454}
anger
{"joy": 11.205957172097522, "sadness": 21.194777638706331, "neutral":
8.7114412831251382, "disgust": 21.885239450213753, "anger": 26.075016894571377, "fear":
10.92756756128588}
anger


Input:  wow/voldemort_joy_1.wav
{"joy": 23.298929799334971, "neutral": 14.255030324571992, "sadness":
14.607596251010728, "disgust": 10.198686485271438, "anger": 23.003846530019157, "fear":
14.635910609791729}
joy
Harry Potter is dead
{"anger": 0.0, "joy": 0.0, "fear": 30.670498803304781, "sadness": 9.7943129542071237,
"disgust": 59.535188242488104}
disgust
{"joy": 16.30925085953448, "sadness": 13.163611261969645, "neutral": 9.9785212272003943,
"disgust": 24.999637012436438, "anger": 16.10269257101341, "fear": 19.446287067845645}
joy


Correct class:  sadness
Input:  wow/diggory_sadness_1.wav
{"joy": 10.327022660068371, "neutral": 15.896629735971134, "sadness":
16.318985900479824, "disgust": 19.246359001290685, "anger": 25.594945168525172, "fear":
12.616057533664819}
anger
the first song
{"anger": 9.9827481923355119, "joy": 39.66561034756284, "fear": 20.244718061074682,
"sadness": 30.106923399026968, "disgust": 0.0}
joy
{"joy": 19.128598966316709, "sadness": 20.455367150043966, "neutral":
11.127640815179793, "disgust": 13.472451300903479, "anger": 20.911286075668272, "fear":
14.904655691887776}
anger


Input:  wow/harry_sadness_1.wav
```

```
{"joy": 16.004528861054713, "neutral": 11.373905083091271, "sadness":
22.937980546900299, "disgust": 17.118074599422695, "anger": 15.26247750387166, "fear":
17.303033405659363}
sadness
hello
{"anger": 29.318266423269318, "joy": 29.318266423269318, "fear": 31.590711679038257,
"sadness": 9.7727554744231053, "disgust": 0.0}
fear
{"joy": 19.998650129719092, "sadness": 18.98841302515714, "neutral": 7.9617335581638891,
"disgust": 11.982652219595886, "anger": 19.479214179690956, "fear": 21.589336887673028}
sadness


Input:  wow/harry_sadness_2.wav
{"joy": 17.272842058264281, "neutral": 15.268206836784854, "sadness":
24.807880079767919, "disgust": 15.07390254602614, "anger": 9.8312107217471532, "fear":
17.745957757409659}
sadness
can you back the money
{"anger": 10.238994495600643, "joy": 19.90315124098062, "fear": 29.866699929449304,
"sadness": 20.098876253774353, "disgust": 19.892278080195073}
fear
{"joy": 18.061934813079183, "sadness": 23.395178931969848, "neutral":
10.687744785749397, "disgust": 16.519415206276818, "anger": 9.9535458539032007, "fear":
21.382180409021551}
sadness


Input:  wow/harry_sadness_3.wav
{"joy": 15.082418503543227, "neutral": 11.5391079088479, "sadness": 22.768016928541666,
"disgust": 11.925771733158411, "anger": 21.733819489224693, "fear": 16.95086543668409}
sadness
Google Speech Recognition could not understand audio

{"anger": 30.0, "joy": 29.999999999999993, "fear": 9.9999999999999982, "sadness":
19.999999999999996, "disgust": 9.9999999999999982}
anger
{"joy": 19.557692952480256, "sadness": 21.937611849979163, "neutral":
8.0773755361935287, "disgust": 11.348040213210888, "anger": 24.213673642457284, "fear":
14.865605805678863}
sadness


Input:  wow/khadgar_sadness_1.wav
{"joy": 16.614191932649664, "neutral": 10.761967202273295, "sadness":
21.109157803294057, "disgust": 11.774294272173284, "anger": 22.497238116037003, "fear":
17.2431506735727}
anger
Google Speech Recognition could not understand audio

{"anger": 30.0, "joy": 29.999999999999993, "fear": 9.9999999999999982, "sadness":
19.999999999999996, "disgust": 9.9999999999999982}
anger
{"joy": 20.629934352854761, "sadness": 20.776410462305837, "neutral":
7.5333770415913062, "disgust": 11.242005990521298, "anger": 24.7480666812259, "fear":
15.07020547150089}
anger


Correct class:  fear
Input:  wow/harry_fear_1.wav
{"joy": 12.894591145540778, "neutral": 13.517822458620973, "sadness":
20.451894453226085, "disgust": 17.898323583026066, "anger": 19.946507105054852, "fear":
15.290861254531269}
sadness
foldable
{"anger": 30.0, "joy": 29.999999999999993, "fear": 9.9999999999999982, "sadness":
19.999999999999996, "disgust": 9.9999999999999982}
anger
{"joy": 18.026213801878541, "sadness": 20.316326117258257, "neutral":
9.4624757210346804, "disgust": 15.528826508118245, "anger": 22.962554973538396, "fear":
13.703602878171887}
```

```
sadness


Input:  wow/khadgar_fear_1.wav
{"joy": 19.795277991726817, "neutral": 9.9976766097579119, "sadness":
14.331631329432806, "disgust": 16.938774453970431, "anger": 23.28092679915537, "fear":
15.655712815956662}
anger
it's not working
{"anger": 10.045041862075824, "joy": 20.22477453927047, "fear": 39.813901885902318,
"sadness": 9.9040661187513415, "disgust": 20.012215594000043}
fear
{"joy": 19.924126955989912, "sadness": 13.003361766228366, "neutral":
6.9983736268305377, "disgust": 17.860806795979315, "anger": 19.310161318031504, "fear":
22.903169536940361}
anger


-----
Score List:  {"anger": 212.5, "joy": 120.0, "sadness": 120.0, "fear": 50.0}
Total Score:  125.625
Time elapsed:  108.775345087
```

Appendix F: Design Materials

F.1: Design Materials for Web Service

Use Case: Data upload via Microphone Input

Id: UC- 1

Description
User wants to conduct an emotion classification case via microphone input

Primary Actor
User

Supporting Actors

Client

Pre-Conditions
User must have an operating microphone
User must connected the application via HTTPS
User must have a supported browser*

Post Conditions

Success end condition
Users microphone input is successfully uploaded to the instance and prepared for classification

Failure end condition:
Users data cannot be uploaded and the error view is shown. User can go back to invoke the use case again.

Main Success Scenario

1. User launches the web application via HTTPS
2. User gives access to web application to use the microphone input
3. Client confirms the access
4. User clicks the “Record” button.
5. Client starts the recording and shows the time spent since the recording has started
6. User clicks “Classify” when he wants to end the recording.
7. Client prepares the stream buffer to export and shows the indicator that the recording is processing.
8. Client requests uploads the file to the instance.
9. Use case [Preprocessing] is initiated

Extensions

In step 6, If the user marks the download checkbox, the recorded audio file is downloaded to users local machine.

Use Case: Data upload via Data Input Form

Id: UC- 2

Description
User wants to conduct an emotion classification case via using the data input form

Primary Actor
User

Supporting Actors
Client

Pre-Conditions
[1]User must connected the application via HTTPS

Post Conditions

Success end condition
Users sound input is successfully uploaded to the instance and prepared for classification

Failure end condition:
Users data cannot be uploaded and the error view is shown. User can go back to invoke the use case again.

Main Success Scenario

1. User launches the web application via HTTPS
2. User clicks the “Choose File” button and selects the file that will be classified.
3. Client sets the file to the data form
4. User clicks the “Upload” button.
5. Client requests uploads the file to the instance.
6. Use case [Audio Classification] is initiated

Extensions

In step 5, If the user uploads a file that is not a wav format, the system halts and the use case terminates.
In step 5, If the user marks the download checkbox, the uploaded audio file is downloaded to users local machine.

Assumptions

In the success case it is assumed that the user is using a valid wav file to upload.

---

[1] supported browser list can be found in the web application section in this paper.

Use Case: Audio Classification

Id: UC- 3

Description
Classifier is loaded and predicts the label of the given audio file

Primary Actor
Server

Supporting Actors
Client

Pre-Conditions
The file must be a valid wav file.

Post Conditions
Results view is routed with the probability distribution of classes

Main Success Scenario

1. Server retrieves the preprocessed export of the uploaded file
2. Server extracts the features of the audio file by using STE Feature extractor.
3. Server invokes audio classification by using KNN classifier.
4. Server loads the KNN Classifier trained with TESS dataset.
5. Server retrieves the classifier result and sends the response to the Client.
6. Client parses the response data and loads the result view.

Extensions

In step 4, if the Application can't find the saved classifier the application halts and Server loads error view.

Assumptions

The preprocessed export is only available if the user used microphone input. If the user used data form the original export is used instead.
Server is online and operating.
KNN Classifier is stored on the instance and can be loaded.

Variations:

If the User used data form for upload the server retrieves the original export

Use Case: Preprocessing

Id: UC- 4

Description
Audio file is preprocessed before classification

Primary Actor
Server

Pre-Conditions
The file must be a valid wav file.

Post Conditions
The preprocessed file will be sent for classification

Main Success Scenario

1. Server splits the wav file so that the first click sound in the first 0.5 second is removed.
2. Server initiates normalisation for the audio file
3. Server exports the audio file and changes path of the original file to the new one.
4. Use case [Audio Classification] begins.

Extensions

In step 4, if the Application can't find the saved classifier the application halts and Server loads error view.

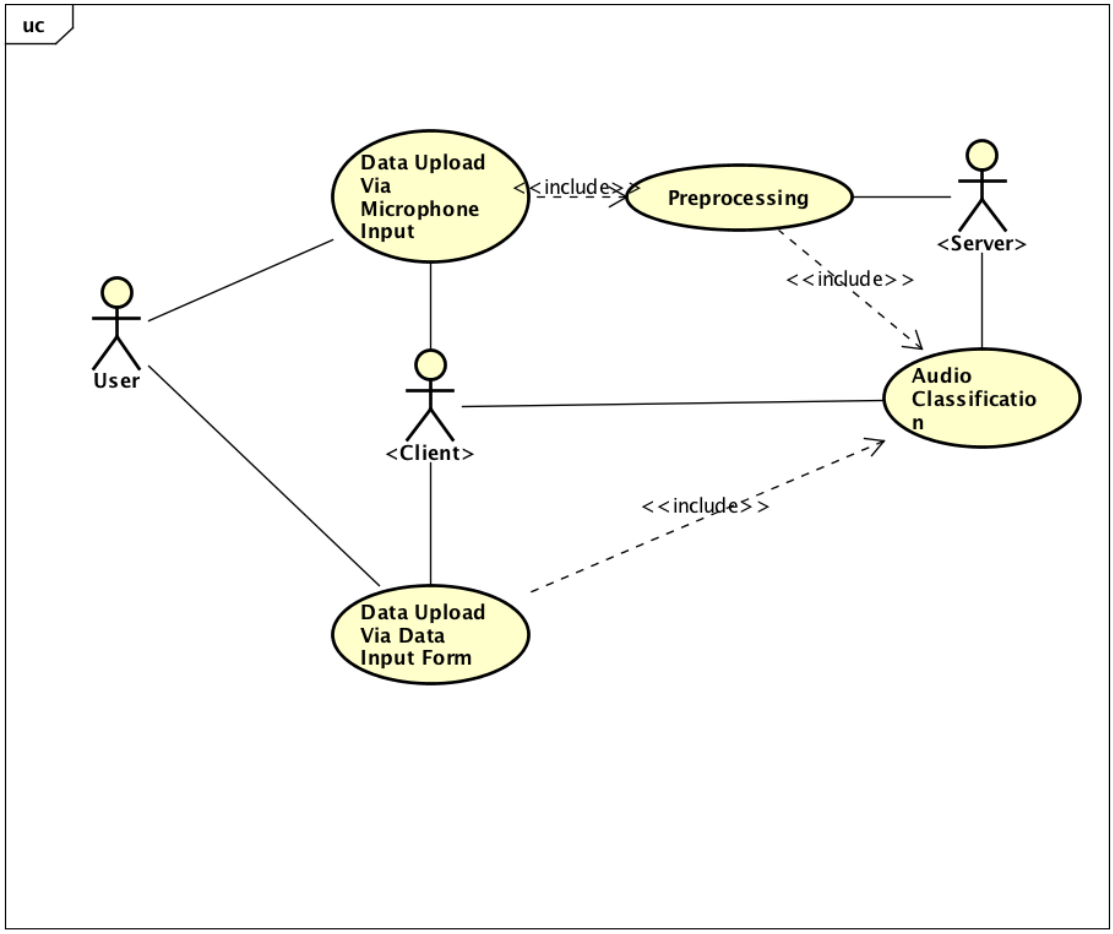
Variations

After step 2, A bandpass filter can be applied optionally. A high-pass filter with 300Hz and low-pass filter with 3400Hz is used for this process.

Assumptions

The preprocessed export is only available if the user used microphone input. If the user used data form the original export is used instead.
Server is online and operating.
The variation case is only valid for a audio file that has not been marked as professional recorded.

F.2: Use Case Diagram for Web Service Features



Appendix G: Program Manual

```
NAME
      STE - Speech to Emotion Tool
SYNOPSIS
      Python STE.py [-S] folder [-c] KNN
DESCRIPTION
      STE is a emotion classification program that is able to predict
emotions    from a speech sample. It can conduct speech, text or
hybrid classification. It can be optionally used by only one type of
classification by choosing either speech or text.

OPTIONS
     -s bool
      enable or disable speech classification parameter. Bool value
accepts True      or False

     -t bool
      enable or disable text classification parameter. Bool value
accepts True      or False
     -c classifier
      configures the classifier that will be used for classification.
KNN,SVM,DT  are supported arguments.

     -S path
      The path of the sample audio file that will be classified. It
will act recursively if the path is a directory.


BUGS
      No exception handling system if the sample path is a non-audio
file.

AUTHOR
     Ugur Bastug
```

Appendix H: Glossary and Abbreviations

STE: Speech To Emotion, the name of the application that is created.

MFCC: Mel-Frequency Cepstal Coefficients

ASR: Automatic Speech Recognition

KNN: K-Nearest Neighbours

Emo-DB: Berlin Database of Emotional Speech

TF-IDF: Term Frequency Inverse Document Frequency

SAVEE: Surrey Audio Visual Expressed Emotion

SVM: Support Vector Machines

GMM: Gaussian Mixture Models

ZCR: Zero Crossing Rates

LPCC: Linear Predictive Coding Coefficients

TESS: Toronto Emotional Speech Set

FFT: Fast Fourier Transform

SSC: Spectral Subband Centroids

FBE: Filterbank Energies

Split Dataset Testing: Dataset is divided into two parts and used one part for testing and the other part for training. In all experiments the data is splitter into 20%(testing) and 80%(training)

Cross-validation Testing: Dataset is split into N-folds. Often, 1 fold used for testing and N-1 fold used for training. The average result is calculating by doing the testing N times by cycling the test fold over all iterations.

Appendix I: Test Result Analysis

These results are obtained from unbiased experiments with the purpose of showing the different outcomes by changing the classifiers, datasets and features.

The main objective of these results tables are shown in order to prove that the task is reliable and stabile and is able to show consistent results if used with recommended set of choices.

Appendix I.1: KNN on TESS Dataset

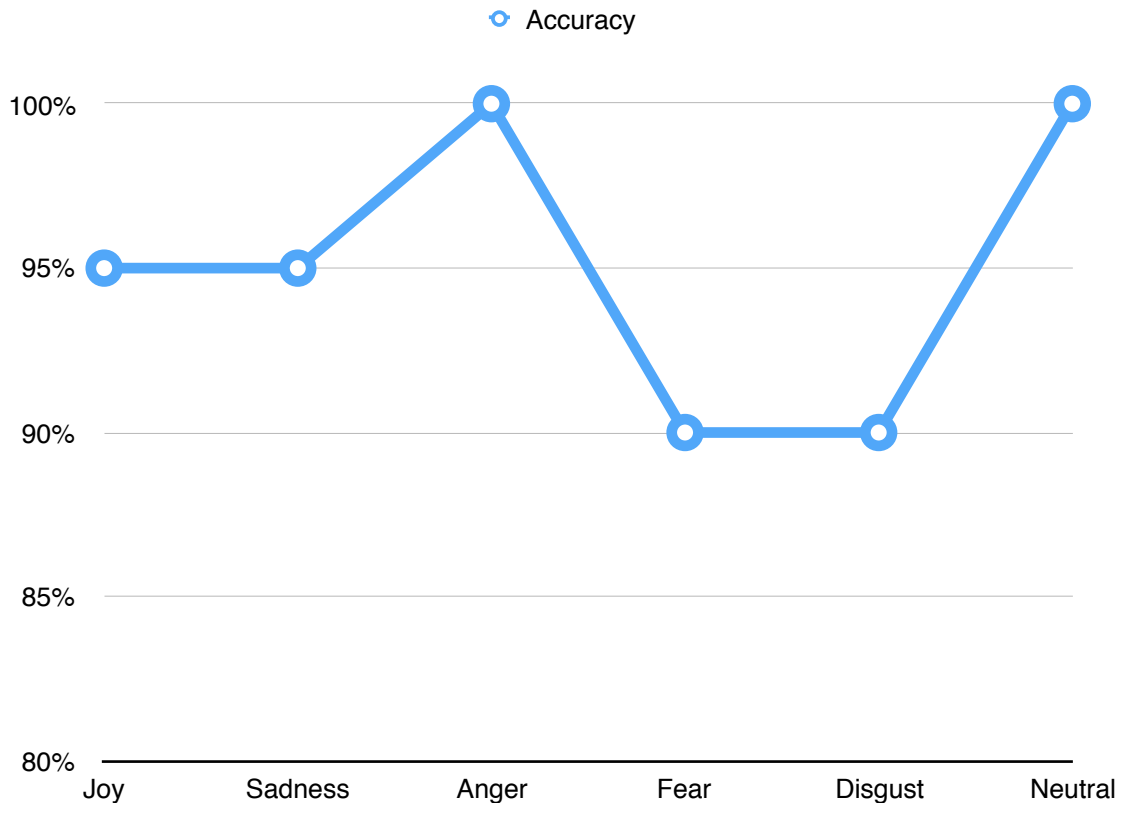
Audio

Features:

1. MFCC
2. MEAN MFCC
3. STDEV MFCC
4. MAX_MFCC
5. MIN_MFCC
6. DELTA MFCC
7. DELTA DELTA MFCC
8. MEAN FBANK
9. STD DELTA FBANK
10. FBANK
11. DELTA FBANK
12. DELTA DELTA FBANK
13. MEAN SSC

	Joy	Sadness	Anger	Fear	Disgust	Neutral
Score	**95.0**	**95.0**	**100.0**	**90.0**	**90.0**	**100.0**
Time	*24.1771 221161*	*23.4975 881577*	*24.1207 430363*	*22.5825 04034*	*24.8159 661293*	*23.8247 010708*

Description:
The results above are gathered by using K-Nearest-Neighbours classifier with Toronto Emotional Speech Dataset. Data set is split into test and training samples in order to conduct an accuracy score. K integer value is defined as 10.



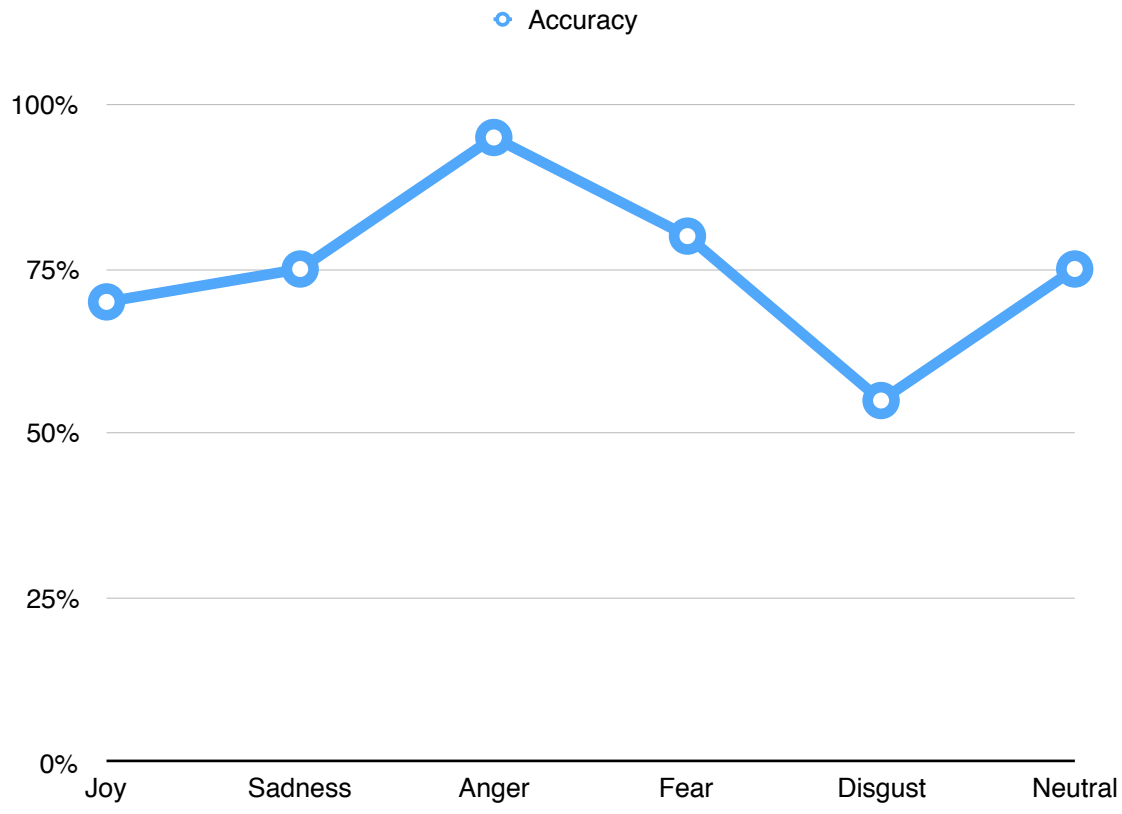
Audio

Features:

1. MFCC
2. MEAN MFCC
3. STDEV MFCC
4. MAX_MFCC
5. MIN_MFCC
6. DELTA MFCC
7. DELTA DELTA MFCC
8. MEAN FBANK
9. STD DELTA FBANK
10. FBANK
11. DELTA FBANK
12. DELTA DELTA FBANK
13. MEAN SSC

	Joy	Sadness	Anger	Fear	Disgust	Neutral
Score	**70.0**	**75.0**	**95.0**	**80.0**	**55.0**	**75**
Time	*25.7019388676*	*24.7379310131*	*24.5852019787*	*25.0476009846*	*24.7225980759*	*25.2251710892*

Description:
The results above are gathered by using SVM classifier with Toronto Emotional Speech Dataset. Data set is split into test and training samples in order to conduct an accuracy score. This classifier is using RBF kernel with C value defined as 1.0.



Appendix I.3: Linear-SVM on TESS Dataset

Audio

Features:

1. MFCC
2. MEAN MFCC
3. STDEV MFCC
4. MAX_MFCC
5. MIN_MFCC
6. DELTA MFCC
7. DELTA DELTA MFCC
8. MEAN FBANK
9. STD DELTA FBANK
10. FBANK
11. DELTA FBANK
12. DELTA DELTA FBANK
13. MEAN SSC

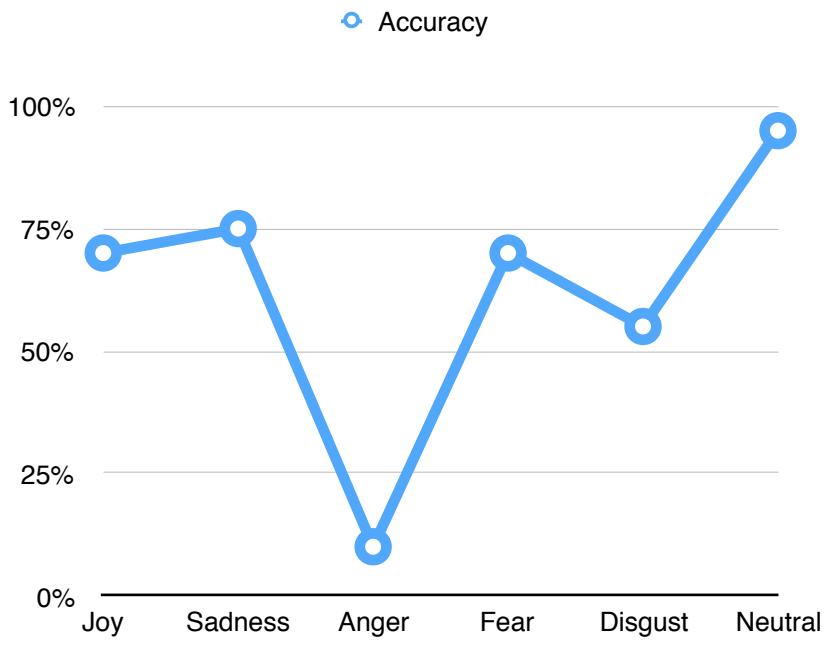
	Joy	Sadness	Anger	Fear	Disgust	Neutral
Score	**70.0**	**75.0**	**10.0**	**70.0**	**55.0**	**95.0**
Time	*25.4230 480194*	*26.2485 089302*	*26.5043 871403*	*24.8632 43103*	*26.8502 321243*	*25.2575 099468*

Description:

The results above are gathered by using SVM classifier with Toronto Emotional Speech Dataset. Data set is split into test and training samples in order to conduct an accuracy score. This classifier is using linear kernel with C value defined as 1.0.

As it is seen the linear kernel is giving less accurate results compared to RBF kernel SVM.

It can be seen that “anger” score is diminished drastically, which is interesting since all the other classifiers using TESS outputs the best accuracy for the emotion “anger”.The training of Linear SVM is significantly higher than other classifiers including RBF-SVM



Appendix I.4: Decision Tree on TESS Dataset

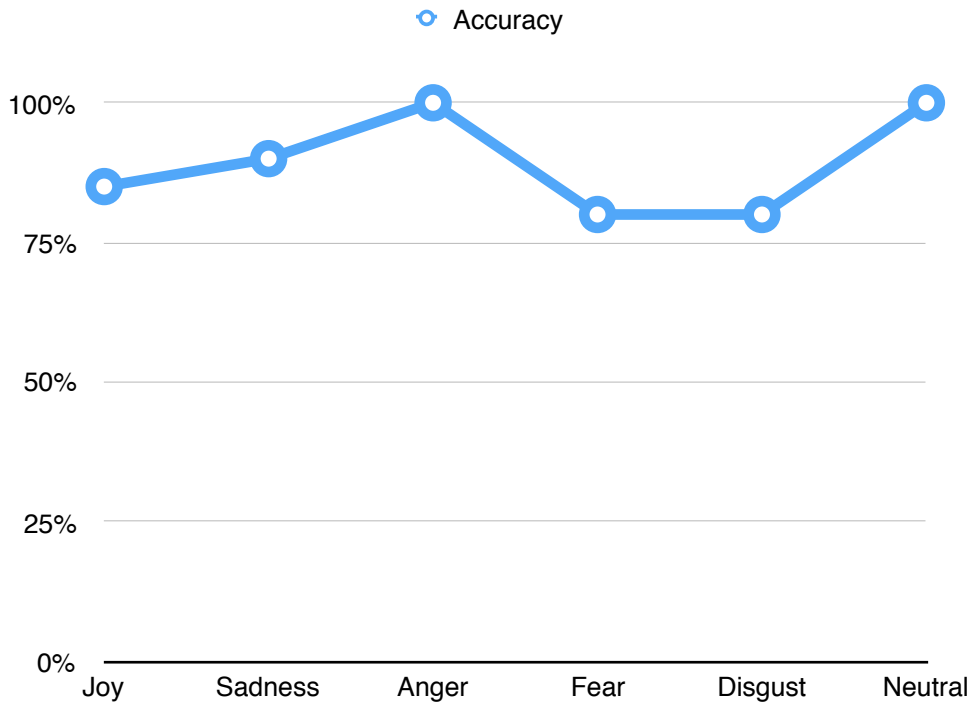
Audio

Features:

1. MFCC
2. MEAN MFCC
3. STDEV MFCC
4. MAX_MFCC
5. MIN_MFCC
6. DELTA MFCC
7. DELTA DELTA MFCC
8. MEAN FBANK
9. STD DELTA FBANK
10. FBANK
11. DELTA FBANK
12. DELTA DELTA FBANK
13. MEAN SSC

	Joy	Sadness	Anger	Fear	Disgust	Neutral
Score	**85.0**	**90.0**	**100.0**	**80.0**	**80.0**	**100.0**
Time	*38.1132860184*	*37.5558919907*	*37.272892952*	*36.8839700222*	*36.3548791409*	*36.9590959549*

Description:
The results above are gathered by using Decision Tree classifier with Toronto Emotional Speech Dataset. Data set is split into test and training samples in order to conduct an accuracy score. Classifier have 1 minimum number of samples required for a leaf node and minimum number of samples required to split an internal node. Strategy best is used for node splitting.



Appendix I.5: KNN on EmoDB Dataset

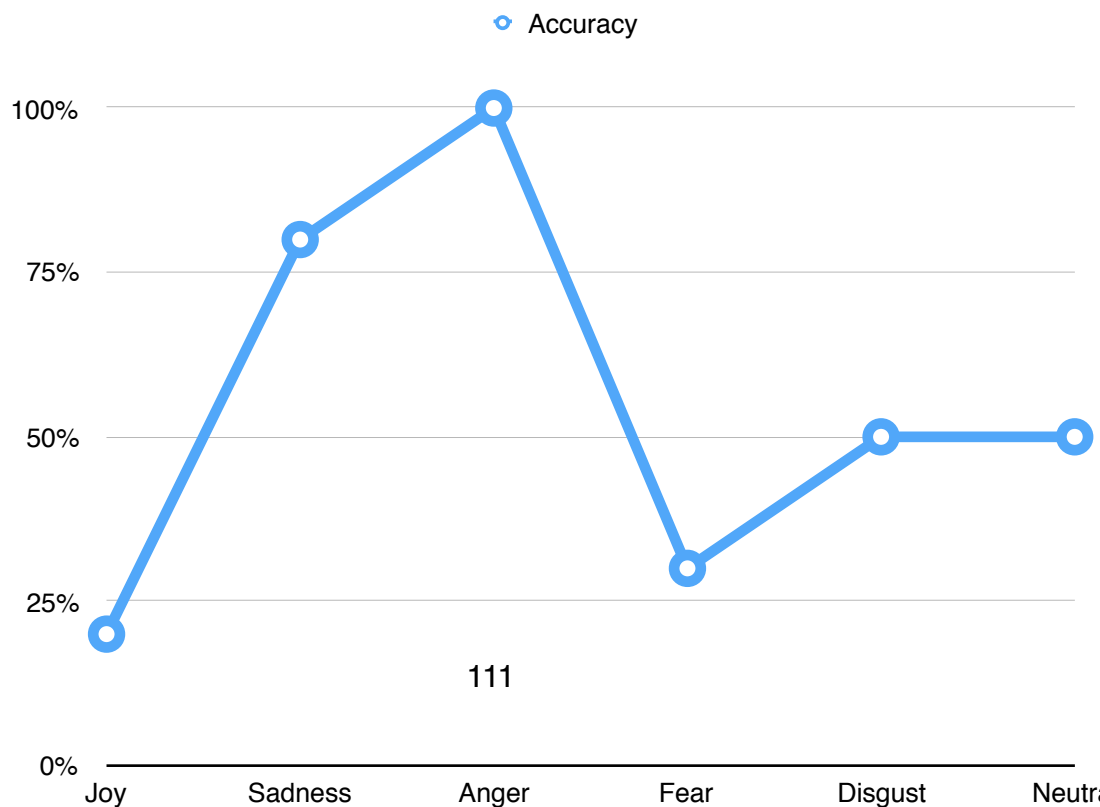
Audio

Features:

1. MFCC
2. MEAN MFCC
3. STDEV MFCC
4. MAX_MFCC
5. MIN_MFCC
6. DELTA MFCC
7. DELTA DELTA MFCC
8. MEAN FBANK
9. STD DELTA FBANK
10. FBANK
11. DELTA FBANK
12. DELTA DELTA FBANK
13. MEAN SSC

	Joy	Sadness	Anger	Fear	Disgust	Neutral
Score	**20.0**	**80.0**	**100.0**	**30.0**	**50.0**	**50.0**
Time	*9.68820714951*	*10.4355480671*	*10.2796888351*	*11.1330080032*	*9.87406992912*	*9.74673891068*

Description:
These results are gathered by using a KNN classifier with n=10 on Berlin Emotional Speech(EmoDB) dataset. It can be seen that emotions like joy and fear is less detectable in German language on the other hand emotions sadness and anger have good recognition rates. It should be also noted that the training set of this experiment is limited dramatically in order to equalise the number of samples for each emotions.



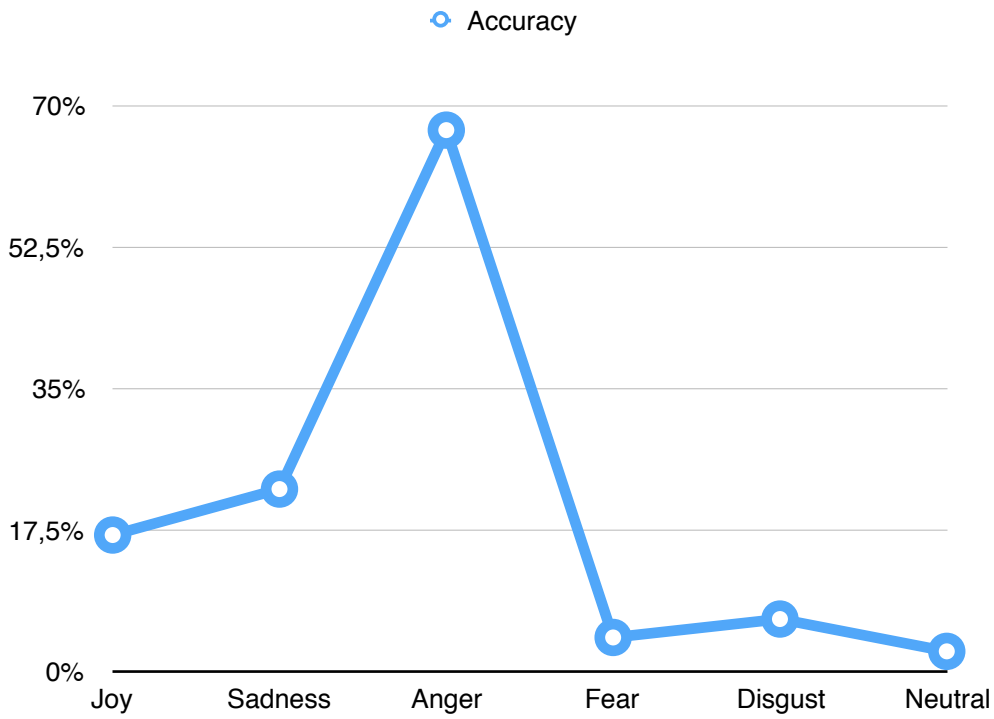
Audio

Features:

1. MFCC
2. MEAN MFCC
3. STDEV MFCC
4. MAX_MFCC
5. MIN_MFCC
6. DELTA MFCC
7. DELTA DELTA MFCC
8. MEAN FBANK
9. STD DELTA FBANK
10. FBANK
11. DELTA FBANK
12. DELTA DELTA FBANK
13. MEAN SSC

	Joy	Sadness	Anger	Fear	Disgust	Neutral
Score	**16.9014084507**	**22.5806451613**	**66.9291338583**	**4.34782608696**	**6.52173913043**	**2.53164556962**
Time	*39.2744181156*	*39.8750741482*	*37.2728 92952*	*37.3803818226*	*33.9230520725*	*40.758769989*

Description:
The results above are gathered by using KNN classifier with Toronto Emotional Speech Dataset(English) and Emo-DB(German) in order to experiment with language independent classification. The results show that this task is not viable with language independent cases.



Appendix I.7: KNN on Experimental Dataset

Audio

Features:

1. MFCC
2. MEAN MFCC
3. STDEV MFCC
4. MAX_MFCC
5. MIN_MFCC
6. DELTA MFCC
7. DELTA DELTA MFCC
8. MEAN FBANK
9. STD DELTA FBANK
10. FBANK
11. DELTA FBANK
12. DELTA DELTA FBANK
13. MEAN SSC

	Joy	Sadness	Anger	Fear	Overall
Score	60.0	60.0	87.5	0.0	51.875

See **Appendix B.1** for the predicted emotions for every sample

Effect of exclusion of Mean Spectral Subband Centroids:

	Joy	Sadness	Anger	Fear	Overall
Score	20.0	60.0	87.5	0.0	41.875

See **Appendix B.2** for the predicted emotions for every sample

Effect of exclusion of mean Delta and Delta Delta Features(MFCC and FBANK)

	Joy	Sadness	Anger	Fear	Overall
Score	40	0	75	0.0	28.75

See **Appendix B.3** for the predicted emotions for every sample

Effect of exclusion of mean Delta Delta Features (MFCC and FBANK)

	Joy	Sadness	Anger	Fear	Overall
Score	40	0	75.0	0.0	28.75

Effect of working with not normalised MFCC values

	Joy	Sadness	Anger	Fear	Overall
Score	20	20	50	0.0	22.5

The example above proves that the normalisation is essential since the non-normalised MFCC samples reduced the score dramatically.See **Appendix B.4** for the predicted emotions for every sample

Effect of exclusion of max and min values of MFCC

	Joy	Sadness	Anger	Fear	Overall
Score	40.0	80.0	87.5	0.0	51.875

See **Appendix B.5** for the predicted emotions for every sample

Appendix I.8: Translation Tests

Audio => Text

The following test cases are conducted by using Experimental Dataset with Speech Recognition library's Google Speech Recognition module. See **Appendix D.1** for full output

File name	diggory_sadness_1	goblin_joy_1	guldan_anger_1	harry_fear_1	harry_sadness_1
Expected Translation	Thats my son!	Hey back me up here!	I have proved my loyalty a thousand times over.	Voldemort is back	Hold on, hold on.
Output Translation	**the first song**	**Peppa Pig**	**I have pulled my loyalty 1000 times over**	**foldable**	**Hello**
File name	harry_sadness_2	harry_sadness_3	khadgar_anger_1	khadgar_anger_2	khadgar_anger_3
Expected Translation	In your bag, Hermione	I can't leave him	Where is he hiding?	This way, go, go, go	Everyone, get to high ground
Output Translation	**can you back the money**		**where is Eid**	**this way yo yo yo**	**1 kg**
File name	khadgar_anger_4	khadgar_fear_1	khadgar_joy_1	khadgar_joy_3	khadgar_joy_5
Expected Translation	Give yourself up	It's not working	This will only hurt	This magic is extremely reliable	I do parties, In fact I am the party.
Output Translation	**give yourself a**	**it's not working**	**this will only hurt**	**this magic is extremely reliable**	**I do parties in fact I am the party**
File name	khadgar_sadness_1	kiljaiden_anger_1	kiljaiden_anger_2	voldemort_joy_1	fightclub_anger_1
Expected Translation	I'm sorry Garum	You disobey me?	Yes, keep moving.	Harry Potter is dead	Marla you liar, you …
Output Translation	**IMC group**	**disappeared**		**Harry Potter is dead**	**Playa Victoria**

[2]Time elapsed: 90.6302218437

[2] Time is calculated only by text translation.

Appendix I.9: KNN on Experimental Dataset

Text

File name	diggory_sadness_1	goblin_joy_1	guldan_anger_1	harry_fear_1	harry_sadness_1
Output Translation	**the first song**	**Peppa Pig**	**I have pulled my loyalty 1000 times over**	**foldable**	**Hello**
1-gram	sadness	sadness	fear	sadness	fear
2-gram	**joy**	**anger**	**fear**	**anger**	**fear**
3-gram	joy	anger	fear	fear	fear
File name	harry_sadness_2	harry_sadness_3	khadgar_anger_1	khadgar_anger_2	khadgar_anger_3
Output Translation	**can you back the money**		**where is Eid**	**this way yo yo yo**	**1 kg**
1-gram	joy		fear	fear	disgust
2-gram	**fear**		**fear**	**fear**	**anger**
3-gram	fear		fear	fear	fear
File name	khadgar_anger_4	khadgar_fear_1	khadgar_joy_1	khadgar_joy_3	khadgar_joy_5
Output Translation	**give yourself a**	**it's not working**	**this will only hurt**	**this magic is extremely reliable**	**I do parties in fact I am the party**
1-gram	sadness	fear	sadness	fear	sadness
2-gram	**sadness**	**fear**	**sadness**	**fear**	**anger**
3-gram	sadness	disgust	sadness	fear	anger
File name	khadgar_sadness_1	kiljaiden_anger_1	kiljaiden_anger_2	voldemort_joy_1	fightclub_anger_1
Output Translation	**IMC group**	**disappeared**		**Harry Potter is dead**	**Playa Victoria**
1-gram	fear	fear		disgust	sadness
2-gram	**fear**	**fear**		**disgust**	**anger**
3-gram	fear	fear		disgust	fear

Appendix I.10: KNN on Experimental Dataset

Hybrid

Features:

1. MFCC
2. MEAN MFCC
3. STDEV MFCC
4. MAX_MFCC
5. MIN_MFCC
6. DELTA MFCC
7. DELTA DELTA MFCC
8. MEAN FBANK
9. STD DELTA FBANK
10. FBANK
11. DELTA FBANK
12. DELTA DELTA FBsy.ANK
13. MEAN SSC
14. TF-ID

	Joy	Sadness	Anger	Fear	Overall
Speech	60.0	60.0	87.5	0.0	51.875
Text	0.0	0.0	37.5	50.0	21.875
Hybrid	60.0	60.0	87.5	0.0	51.875

See **Appendix E** for the full output of Hybrid Classification experiments that shows the probability distributions for all classes

Appendix J: Application Usage

Installation - README.txt

```
Install Virtual Environment:

$ pip install virtualenv

Create Virtual Environment Instance:
$ virtualenv venv

Activate Virtual Environment:
$ source venv/bin/activate

Install Requirements:
$ pip install -r requirements.txt

Use STE:
$ python STE.py

Default Options:
     -t False
     -s True
     -c KNN

Use of Options:

Use STE with Specific Classifier:
$ python STE.py -c SVM
$ python STE.py -c KNN
$ python STE.py -c DT

Use STE with only Text Classification:
$ python STE.py -s False -t True

Use STE with Hybrid Classification:
$ python STE.py -s True -t True

Use STE on custom audio folder:
$ python STE.py -S wow/
```

Appendix K: Project Proposal Report

UNIVERSITY OF WESTMINSTER

On Classification and Recognition of Emotions by Investigating Equivalent Speech and Textual Data

MSc. Computer Science Dissertation Proposal

Ugur Bastug, 157435854
w1574358@my.westminster.ac.uk

Supervisor: Simon Courtenage

Introduction

Overview

The public interest for applicable artificial intelligence components is increasing day by day. Many fields of intelligent systems are appealing to today's technology, including robotics, intelligent user interfaces etc. However, the general interest is mostly leaning towards the human vs computer interacted mixture applications. I believe automatic human emotion recognition is the ultimate blending point in the human vs computer interaction. This is the main reason I chose this topic as my final project.

Project Focus

Predicting emotions is a highly demanding task in the Machine Learning field. There is many research that aims to have satisfactory results in predicting emotions from both plain text and speech using state-of-art classification algorithms, yet the task itself is limited to several categorisations.
Most of the research cases are only known to people who both excelled in the subject and have a technical background knowledge of the related tasks and also in the machine learning field. A part of my project is to make this project for people who don't have technical knowledge and still experience the emotion classification task. Because of that reason, I'm additionally planning to build a web application that will be ready to use for the public by recording their speech to be classified by my application online.
I will inspect the emotion classification task by utilising both the speech and the plain text and I will aim to investigate which approaches can result in better classification. What features in the speech and text data effects the output and ultimately, what could be the possible explanation that makes speech classification superior to plain text classification or visa versa.

Project Aims

The main aim of the project is to find a feasible solution to the emotion classification problem. There are many research studies and projects conducted to obtain emotional data from various sources of human input including speech, text, visual face patterns. Since this is a research area and we are aware that there is not yet a state-of-art solution for this problem, different perspectives for this classification problem is still on debate.
The ultimate aim and the main purpose of this project is to deliver an alternative emotion classification application that takes human speech audio as an input and outputs an emotion from the input.
My projected approach will be a hybrid model that consists of both text classification results and speech classification results. The most important part of having a reliable result is to find a balancing point between these two inputs.
The second aim for this project is to create a public web environment to run the application online. The main reason for this aim is to have the opportunity of publicly testing the application by reviewing user inputs and calculate it's correctness. The alternative purpose is to give the chance to people who are not familiar with the technical concerns of the field, to experience today's technology. I'm planing to enable user microphones to record their voices and take that as an input and run the application.

Objectives

In order to achieve measurable emotion prediction results the following objectives will be followed.

Automatic Speech Recognition Translation

In obtaining textual representation of the speech audio, I will be using Google Cloud Speech API tool for speech to text conversion. The API can recognise over 80 languages and is able to handle, clean and optimise noise in the audio. By using this component I will be able to get the textual representation of a human speech audio file.

Feature Extraction

Text Data

For text data, I will be using keyword extraction and BoW approach in order to create feature vectors. The NRC Word-Emotion Association Lexicon provides a list of English words and their associations with the following emotions.

Anger
Fear
Anticipation
Trust
Surprise
Sadness
Joy
Disgust

I will divide the input text data into words in order to perform a unigram keyword comparison. The words will be processed with the NRC lexicon in order to form a feature dataset.

Speech Data

I will be using Python Speech Features for creating the feature dataset for the speech input.
The Python library provides the most essential speech features including Mel Frequency Cepstral Coefficients, Filterbank Energies and Log Filterbank Energies. In order to obtain reliable features for the speech data, the audio file must not contain noise. Google Speech API provides built-in noise removal function but since I haven't tested that functionality I may need an external noise removal library in order to get accurate feature vectors.

Acquiring Reliable Data

In order to acquire reliable data there are some important elements that needed to be considered.
As I mentioned earlier, the audio data must be clean from noise. In order to achieve that the recording should be done in an acoustic environment. Using a condenser microphone, a decent audio card and also recording in a noise perfect environment is preferable.
Secondly, the training data must have obvious emotional aspects to the human ear. Since the audio files will be labeled with emotion categories the data itself should be emotionally expressive.
Berlin Database of Emotional Speech contains about 500 samples spoken by actors in various states of emotions. The database might be feasible to use since the recordings are done by professional actors but since the audio files are recorded in German the textual representation will not be compatible with NRC Lexicon. Because of that reason, I might need to create a sample speech database by my own resources or look for alternative solutions.

Classification Algorithms

Hu and Downie (2010) state that “SVM is the most popular model with top performances”, and use therefore used SVM for their Mood classification task.
Furthermore, most of the emotion classification tasks, which will be mentioned in the Background section in this paper, used Support Vector Machines. Therefore, my current plan will also be using Support Vector Machines for both text and speech analysis.
Schuller et al. (2003) used HMM model for emotion classification by taking solely speech data, and Toivanen et al. (2004) used KNN classifier, but since those classifiers are not as popular as SVM, and because those examples were the only instances through my research process, I decided not to use them at the first stages of my project.

Understanding and recording the results

Throughout the research process the findings and results will be recorded in a precisely detailed way. Parameter changes and weighting factor alterations will be made during the whole project lifecycle in order to compare different prediction scores. It is crucial to take notes for every change in order to understand what changes effects the output.
The main objective of this stage is to maximise the hybrid score of the end product.

Create and implement the web service for public use

The second aim of this project is to migrate this project to a web instance and create a simple web application that can be easily usable online. I’m currently planning to use Amazon Web Services to host the web application. I will migrate the project on my local computer to the remote instance and design a Web Application UI that will enable users to record and send their speech for the online classification.
The results can be recorded in order to populate the testing data which will ultimately help me to evaluate the algorithm’s success rate even more accurately. The users will have an option to send their input for evaluation or not, in order to express their consent for participation.

Background and Context

The background research about the topic is crucial for research projects since the previous works are tested and had results according to their models. It is important to take their notes, and warnings and proceed with the project according to their findings.
In this section I will briefly explain the related works and their emphasised points for their tasks.

Hybrid Models

Houjeij et al.'s (2012) model of fusion emotion classification is based on previous work of Hu and Downie's (2010) music mood classification based on the combination of lyrics and audio features that combines the audio data and lyrics to predict emotions.
Besides predicting emotions solely from text and audio, the main objective of Houjeij et al.'s (2012) approach is to produce a hybrid system that combines both of the input features into one dataset and inputted into SVM classifier.
Their dataset consisted of 350 different movie quotes and their textual representations. The movie quotes were selected based on the variety of emotions. For example, for an emotion labeled as "happy", they picked comedy quotes, and for "fear" they picked quotes from a horror movie in order to acquire reliable audio data. They further observed that pitch and speaking rate changes with the emotional state, for example an angry speaker is louder than a calm speaker.
Their results for the hybrid model is superior than sole text and audio results with the outcome of 46.57 %. The model of Chuang and Wu (2004) uses the acoustic properties of the audio file with the textual representation to create a hybrid model, which inspired me to pursue this objective further.

Combining Music and Lyrics

The model of Hu and Downie (2010) for combining song audio's and lyrics is not a directly relevant topic for this project but their analysis for the text data includes very sophisticated findings. Their feature vectors for text data are content words with and without stemming and also POS tags and function words for text classification. Their audio features are also similar to my planned features, including MFCC's, Flux and Rollof. But since their classification is based on songs, it is not possible to obtain the human speech from music and their research area is not directly comparable to Houjeij et al.'s (2012) work of hybrid classification.

The work of Chuang and Wu's (2004) emotion classification from text and speech also used six labeled classification systems and used SVMs to determine emotion output. For the speech data, they stressed that the **pitch** and **energy** are the most crucial features and furthermore they imply that the pitch of speech is directly relevant with the intensity of the emotion. Because of that information I'm planning to adjust the parameters of my weighted focused on pitch and energy.

Outcomes

1. Most of the previous emotion classification tasks used SVMs for emotion classification both for speech and text data due to it's performance for relevant tasks. Because of that reason I'm planning to use SVM for my project.
2. Chuang and Wu (2004) emphasise the importance of pitch and energy vectors for the magnitude of emotions. Therefore, I will increase the weighing factor of these features for the high intensity emotions such as disgust, anger, happiness.

3. Training data must be reliable. In other words, the speech audio files should be expressive and the emotions should be recognisable to human sense. Houjeij et al.'s (2012) movie quotation data is a good pick because the actor voices are reliable material for data due to their capability of expressing emotions.

Methodology

Approach and Planning

I will being by first testing the usability of the third party tools that I will be using. The dependency control is a major task because like every project, the tools that I'm going to use must be operable since their effect on the entire project is crucial. After that, I will focus on the training data and emotion labelling. When I have the testing data and the dependent components, I will test the algorithm. The implementation process will evolve during the project lifetime and every iteration will contain the testing of the relevant component. This approach will help me to report every finding I got in the implementation process. I will take notes of every stage and how it effects my results. Since the classification task is mostly about changing parameters and feature weights, I'm planning to have vast resources to include in my dissertation paper.

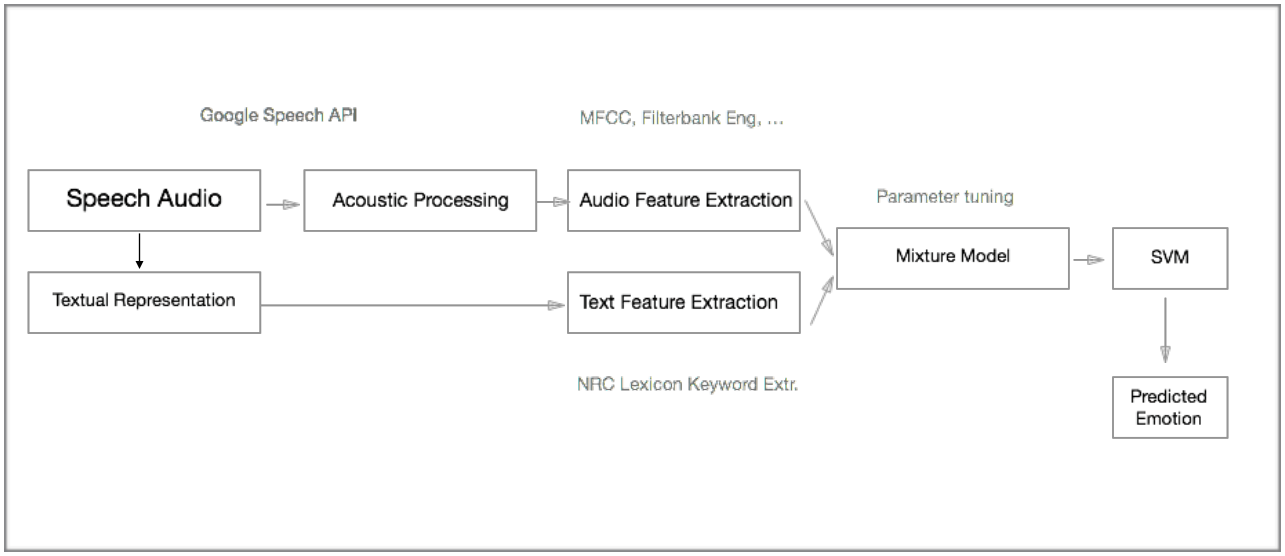
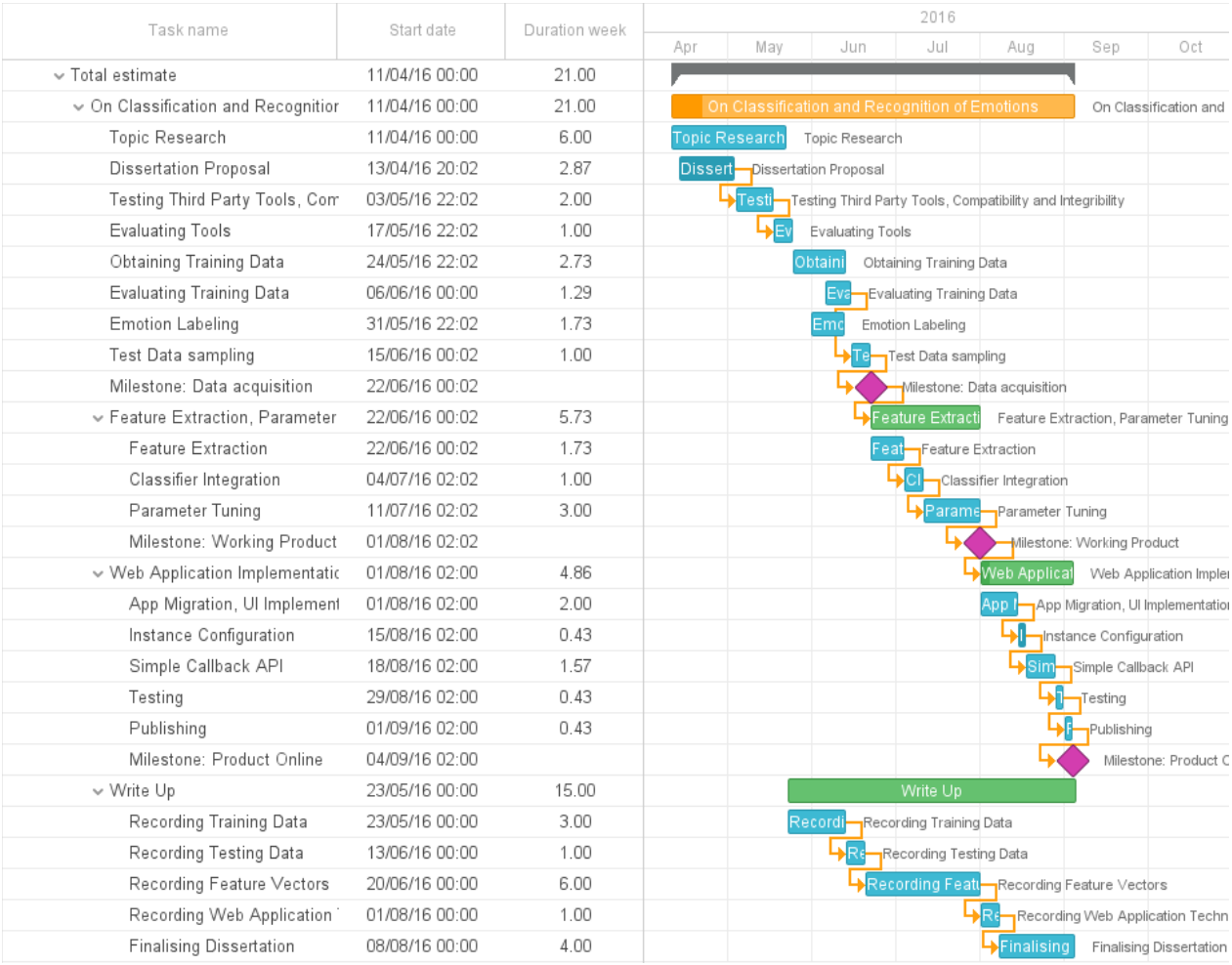


Figure 1: My Projected Model For Emotion Classification From Speech and Text Data

Task Schedule



Task name	Start date	Duration week
Total estimate	11/04/16 00:00	21.00
On Classification and Recognitior	11/04/16 00:00	21.00
Topic Research	11/04/16 00:00	6.00
Dissertation Proposal	13/04/16 20:02	2.87
Testing Third Party Tools, Con	03/05/16 22:02	2.00
Evaluating Tools	17/05/16 22:02	1.00
Obtaining Training Data	24/05/16 22:02	2.73
Evaluating Training Data	06/06/16 00:00	1.29
Emotion Labeling	31/05/16 22:02	1.73
Test Data sampling	15/06/16 00:02	1.00
Milestone: Data acquisition	22/06/16 00:02	
Feature Extraction, Parameter	22/06/16 00:02	5.73
Feature Extraction	22/06/16 00:02	1.73
Classifier Integration	04/07/16 02:02	1.00
Parameter Tuning	11/07/16 02:02	3.00
Milestone: Working Product	01/08/16 02:02	
Web Application Implementatic	01/08/16 02:00	4.86
App Migration, UI Implement	01/08/16 02:00	2.00
Instance Configuration	15/08/16 02:00	0.43
Simple Callback API	18/08/16 02:00	1.57
Testing	29/08/16 02:00	0.43
Publishing	01/09/16 02:00	0.43
Milestone: Product Online	04/09/16 02:00	
Write Up	23/05/16 00:00	15.00
Recording Training Data	23/05/16 00:00	3.00
Recording Testing Data	13/06/16 00:00	1.00
Recording Feature Vectors	20/06/16 00:00	6.00
Recording Web Application	01/08/16 00:00	1.00
Finalising Dissertation	08/08/16 00:00	4.00

Expected Outcomes

The main outcome of this project is to develop an application that can recognise human speech , convert to text, and with this text, predict the emotional state of the speaker. I'm aware that, considering real life communication speech as the input, the classification task may not be trivial. At this stage, I have little knowledge how the program will respond to sarcasm or cynicism. Since real life conversational speech evolves and barely contains emotions, the application may behave different than I expect. For that reason, I see the project as a research topic and scientific journey rather than a commercial product which leads me to see the ultimate outcome as clear notes, reports and understanding about what features and parameter changes are effecting the machine's prediction of human emotions.
A secondary outcome is to make this project publicly usable and testable. This will be achieved by porting the algorithm in a web environment with appropriate user interface and layout. This outcome can be used as a testing service for the primary outcome and can be used for further improvements for the project.

Risks and Ethics

Recording human speech for training data may be trivial but can contain ethical implications. The participants that are willing to record their voice and get tagged with emotion labels should be aware about the process and the research topic in order to satisfy an ethical understanding. It is important that the participants' recordings are kept private and are recorded with consent since we know that human emotions are one of the most private things a human can have.
Furthermore, If I choose to use movie speech data like Houjeij et al. (2012) used, I must be sure that the movie quote data is not protected legally and in public domain. The datasets must be open to use for the public or for research purposes. These two concerns are crucial in order to move on with the project smoothly.

Abbreviations and Glossary

BoW: Bag of Words.
SVM: Support Vector Machine.
NRC Lexicon: A vocabulary table that have a list of English words and their associations with emotions.
MFCC: Mel Frequency Cepstral Coefficients.
KNN: K-Nearest Neighbours.

Annotated Bibliography

Alm, C., Roth, D. and Sproat, R. (2005). Emotions from text. Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing - HLT '05.
This paper includes emotion prediction only by investigating text data. They approached sentence by sentence emotion class mapping by using a corpus of 180 children stories. They used 30 feature vectors, all in binary format. Some of the features they included are first sentence of the story, exclamation marks, WordNet emotion words, thematic story type etc. The implementation is made by using Python, which is the language that I'm going to use and followed a similar approach to mine in terms of using Emotion Words. Informative paper that might benefit my research.

Amazon Web Services, Inc. (2016). Elastic Compute Cloud (EC2) Cloud Server & Hosting – AWS. [online] Available at: http://aws.amazon.com/ec2/ [Accessed 30 Apr. 2016].
I need a web service or cloud computing engine in order to host the web application. AWS is a good option which offers free 12 month trial period.

Chuang, Z. and Wu, C. (2004). Multi-Modal Emotion Recognition from Speech and Text. Compu tational Linguistics and Chinese Language Processing, 9(2), pp. 45-62.
A hybrid model approach. The textual classification benefits from emotional keywords from emotion corpus, similar to my approach of using NRC for keyword comparison. I mentioned this paper in the background section about their underlining of the importance of pitch and energy of the speech which I will be taking into consideration while I'll be tuning my parameters. Their approach model is very similar to mine since they also converted speech to textual representation and then processed the text with emotion corpus. They also used SVM and their results for speech is higher than their results from the text input. Very useful paper for my research.

Emodb.bilderbar.info. (2016). Emo-DB. [online] Available at: http://emodb.bilderbar.info/start.html [Accessed 30 Apr. 2016].
Emotion speech corpus in German. All audio files are recorded in German in well acoustically isolated environment.

GitHub. (2016). jameslyons/python_speech_features. [online] Available at: https://github.com/jameslyons/python_speech_features [Accessed 30 Apr. 2016].
The third party library that I'll be using for determining my feature vectors for speech data. Includes MFCC, Filterbank energy extractions.

Google Cloud Platform. (2016). Speech API - Speech Recognition. [online] Available at: https:// cloud.google.com/speech/ [Accessed 30 Apr. 2016].
Google Speech API, provides automatic speech to text translation. It also includes automatic noise removal library that is abstracted and unmodifiable. Needs research about the API usage limits.

Houjeij, A., Hamieh, L., Mehdi, N. and Hajj, H. (2012). A novel approach for emotion classifica tion based on fusion of text and speech. 2012 19th International Conference on Telecommunications (ICT).

A hybrid model and up to date paper that is very similar to my project. They used noise removal technique for their speech data due to their findings about the human voice range that is between 300-3400Hz. More detailed inspection about this paper is included in the Background section in this paper.

Hu, X. and Downie, J. (2010). Improving mood classification in music digital libraries by combin ing lyrics and audio. Proceedings of the 10th annual joint conference on Digital libraries - JCDL '10.
Not a direct relevant task to my project. Text data feature extraction is more sophisticated than the other papers but their audio data processing is not applicable to my project since they are working on the music and not the human voice.

Mohammad, S. and Turney, P. (2012). CROWDSOURCING A WORD-EMOTION ASSOCIATION LEXICON. Computational Intelligence, 29(3), pp.436-465.
NRC provides list of english words and their association with emotions and also their sentiment values.

Numpy.org. (2016). NumPy — Numpy. [online] Available at: http://www.numpy.org/ [Accessed 30 Apr. 2016].
A Python library for matrix manipulation.

Schuller, B., Rigoll, G. and Lang, M. (2003). Hidden Markov model-based speech emotion recognition. 2003 IEEE International Conference on Acoustics, Speech, and Signal Pro cessing, 2003. Proceedings. (ICASSP ’03).pp 577-80.
Solely works on speech classification. They used HMM, SVM, KNN and GMM. Their audio feature extraction is highly complicated compared to other tasks and because of that it is not directly possible to observe the comparisons between the classifier results without bias. Initially selected this paper in order to compare classifiers and pick a suitable classifier for my task but because of the reason above, I decided to move on with more hybrid approach oriented papers.

Scikit-learn.org. (2016). scikit-learn: machine learning in Python — scikit-learn 0.17.1 documen tation. [online] Available at: http://scikit-learn.org/stable/ [Accessed 30 Apr. 2016].
A Machine learning library for Python which includes numerous classifiers that are easy to use and integrate with projects.

Toivanen, J., Vayrynen, E. and Seppanen, T. (2004). Automatic Discrimination of Emotion from Spoken Finnish. Language and Speech, 47(4), pp.383-412.
Points that human discrimination of emotions varies between 55%-65% on most of the emotions but some emotions such as joy, sadness and anger can be classified in humans with results between 80%-96%. That makes me believe Chuang and Wu’s theory about pitch and energy. They used a Finnish Speech corpus for their data modelling, therefore their technical process is not directly applicable for my task.

Appendix L: Code

STE.py

```
#!/usr/bin/env python

#Essentials
import numpy as np
import sys
import io
import os
import pickle
import speech_recognition as sr
import time
import optparse
from optparse import OptionParser
import json
import operator
#Audio, MP3-Wav Conversion
import wave
#STE-Classes
import ste_feature_extract as sf
import ste_text_classification as ste_text
import ste_decision_tree as ste_dt
import ste_svm
import ste_knn
import ste_dataset_loader as dload
import ste_hybrid
#Google Speech Recognition Translator
from ste_translator import translator
#In development
import ste_preprocess as preproc
#Development only
import pdb as debugger
from collections import defaultdict as dd
#Text
from nltk.stem.porter import *

from pydub import AudioSegment
from sklearn.neighbors import NearestNeighbors

#Deprecated in Early July
#import ste_audio_fix

r = sr.Recognizer()
encoding = 'utf-8'
experimentalTest="wow/"
session = dd(int)
test = dd(list)

allFeatures = False
def main():
      start = time.time()
      #BEGIN: For split testing enable crossvaltest and specift
label
      crossvaltest=True
      test_label = 'joy'
      if not crossvaltest:
```

```
            splitSize = 0
      else:
            splitSize = 20
      #END
      parser = optparse.OptionParser()
      parser.add_option('-s', '--speech',
                        dest="enableSpeech",
                        default='True',
                        )
      parser.add_option('-t', '--text',
                        dest="enableText",
                        default='False',
                        )
      parser.add_option('-S', '--sample',
                        dest="testPath",
                        )
      parser.add_option('-c', '--classifier',
                        dest="classifier",
                        )
      options, remainder = parser.parse_args()

      #TestSet
      #full_exp = preproc.preprocess(full_exp)
      #prepareCustomTest(full_exp,test_label)

      if(options.testPath):
            test =
prepareCustomTest(options.testPath,'Unknown',os.path.isdir(optio
ns.testPath))
      else:
            if crossvaltest:
                  #Do split data testing with TESS
                  test =
prepareTest('TorontoESS',test_label,splitSize)
            else:
                  #Do testing with Toronto vs Experimental Data
                  test = dload.loadWowData(allFeatures)

      #Take the first 10 sample from the dataset

      #For berlin TRAINING data set crossfold to 0 for all
samples to train with.
      if(options.classifier):
            chosenClassifier=options.classifier
      else:
            chosenClassifier='KNN'

      print 'Using: ',chosenClassifier


classify(chosenClassifier,options.enableSpeech,options.enableTex
t,dload.loadTorontoData(splitSize,allFeatures),test)
      #manuallyPreparedText()
      print('-----')
      if crossvaltest:
```

```
            print "Prediction score:
",float(session[test_label])*100/len(test[test_label]),"%"
      else:
            try:
                  #print "Prediction score:
",float(session[test_label])*100/sum(session.values()),"%"
                  score = calculateTestScore(test)
                  print 'Score List: ', formatJSON(score)
                  print 'Total Score: ', sum(score.values())/
len(score)
            except Exception, e:
                  pass
      end = time.time()
      print 'Time elapsed: ',(end - start)


def calculateTestScore(test):
      score = dd(float)
      for label,predictionCount in test.iteritems():
            try:
                  score[label] = float(session[label])*100/
len(test[label])
            except Exception, e:
                  print 'Label match found skipping: ',label
      return score
def prepareCustomTest(path,target,direc):
      if direc:
            for wavFile in os.listdir(path)[::-1]:
                  wavFile = path+'/'+wavFile
                  wavFile = preproc.preprocess(wavFile)

test[target].append((sf.getFeatures(wavFile,allFeatures),wavFile
))
      else:

test[target].append((sf.getFeatures(path,allFeatures),path))
      return test
def prepareTest(dataset,target,crossfold):
      #Preparing the test data with a dataset
      #Target is the directory working only for Toronto ESS
      #Crossfold is the test sample size
      test = dd(list)
      if (dataset=='TorontoESS'):
            dataset='TorontoESS'
            dir=dataset+"/"+target+'/'
            for wavFile in os.listdir(dir)[::-1][:crossfold]:
                  fullPath = dir+wavFile

test[target].append((sf.getFeatures(fullPath,allFeatures),fullPa
th))
      elif(dataset=='Berlin'):
            dir='spr03/wav/'
            for wavFile in os.listdir(dir)[::-1]:
                  fullPath = dir+wavFile
                  germanlabel=wavFile[-6]
```

```
                if germanToEnglish(germanlabel) == target and
len(test[target])<crossfold:

test[target].append((sf.getFeatures(fullPath,allFeatures),fullPa
th))
      return test

def germanToEnglish(character):
      if character == 'A':
            return 'fear'
      elif character == 'E':
            return 'disgust'
      elif character == 'F':
            return 'joy'
      elif character == 'L':
            return 'boredom'
      elif character == 'T':
            return 'sadness'
      elif character =='W':
            return 'anger'
      elif character=='N':
            return 'neutral'

def classify(clfoption,speech,text,training,test):
      #Classify with options. clfoption=classifier, speech = use
speech features, text = use text features
      #training = the training data we are going to use
      ps = dd(list)
      pt = dd(list)
      translator_instance = translator()
      for label,data in test.iteritems():
            #print "Input" + path
            print 'Correct class: ',label
            for sample,path in data:
                  #For each sample tagged with the same label
                  print "Input: ",path
                  #Classifier options: test sample, use the
previously saved classifier
                  #and the training data set we are going to use
for classification.
                  if speech == 'True':
                        if(clfoption=='KNN'):
                              #The Boolean Argument is for reading
pickled Classifier
                              #Third argument is not needed if the
prior is True
                              ps =
ste_knn.prepareKNN(sample,False,training)
                        elif(clfoption=='SVM'):
                              ps =
ste_svm.prepareSVM(sample,True,training)
                        elif(clfoption=='DT'):
                              ps =
ste_dt.prepareDT(sample,False,training)
                        print formatJSON(ps)
                        print getCorrectResult(ps,label)
```

```
                if text == 'True':
                        content =
translator_instance.translate(path)
                        print content
                        pt = ste_text.predictText(content)
                        try:
                                print formatJSON(pt)
                                print getCorrectResult(pt,label)
                        except AttributeError:
                                print 'Skipping Text Classification'
                if text == 'True' and speech == 'True':
                        ph = ste_hybrid.runHybrid(ps,pt)
                        print formatJSON(ph)
                        print getCorrectResult(ps,label)
                print '\n'

def formatJSON(prob):
        return json.dumps(prob, ensure_ascii=False)
def getCorrectResult(prob,label):
        #the second argument is needed only for calculating the
score
        result = max(prob.iteritems(), key=operator.itemgetter(1))
[0]
        if result == label:
                #increase the session prediction rate if the result
is correct for that particular sample.
                session[result]+=1
        return result



if __name__ == '__main__':
        main()
        return result
```

ste_audio_fix.py

```
#This module is not included in STE and currently in
development.
#Alpha
def porterStemLine(line):
      #Text Classification
      arr = []
      stemmer = PorterStemmer()
      for word in line.split():
            stemtest = str(stemmer.stem(word))
            arr.append(stemtest)
      return arr

#In order to accept and process MP3
def convertMP3toWav(path):
      #In order to accept as an MP3 input file for testing.
      sound = AudioSegment.from_mp3(path)
      sound.export(path+".wav", format="wav")

#Alpha
def printWavParams(path):
      #Debugging purposesfwave
      #Returns a tuple (nchannels, sampwidth, framerate,
nframes, comptype, compname), equivalent to output of the get*()
methods.
      waveFile = wave.open(path,"r")
      print(waveFile.getparams())

#Alpha
def returnFrames(path):
      waveFile = wave.open(path,"r")
      return waveFile.getnframes()

#Alpha
def changeBitDepth(path,desiredDepth):
      waveFile = wave.open(path,"w")

#Alpha
def fixWaveFile(path):
      #In order to fix all the wav files with the same sample
rate and width
      sampleRate = 16
      numChans=1
      sWidth=2
      nframes = 114688
      waveFile = wave.open(path,"w")
      waveFile.setparams((numChans, sWidth, sampleRate, nframes,
'NONE', 'not compressed'))
      output = '\0' * nframes * numChans * sWidth
      waveFile.writeframes(output)
      waveFile.close
      #waveFile.setnchannels(1)
      #waveFile.setsampwidth(2)
      #waveFile.setframerate(sampleRate)
def speakLines(translation):
```

```
#For OSX systems to speack the predicted translation.
os.system("say '"+translation+"'")
```

ste_dataset_loader.py

```
from collections import defaultdict as dd
import os
import ste_feature_extract as sf
import pdb as debugger
import ste_preprocess
#Training Data Construction

dir="TorontoESS/"

def loadTorontoData(crossfold,allFeatures):
      #IMPORTANT
      toronto_paths=dd(list)
      toronto_data=dd(list)
      happyDir = dir+'joy/'
      sadDir = dir+'sadness/'
      angryDir = dir+'anger/'
      neutralDir = dir+'neutral/'
      fearDir = dir+'fear/'
      disgustDir = dir+'disgust/'
      totalSamples = len(os.listdir(sadDir))
      #IN DEV
      #crossfold = (totalSamples * crossfold)/100
      #IN DEV
      for wavFile in os.listdir(angryDir)[::-1][crossfold:]:
            fullPath = angryDir+wavFile

toronto_data["anger"].append(sf.getFeatures(fullPath,allFeatures
))
      for wavFile in os.listdir(neutralDir)[::-1][crossfold:]:
            fullPath = neutralDir+wavFile

toronto_data["neutral"].append(sf.getFeatures(fullPath,allFeatur
es))
      for wavFile in os.listdir(happyDir)[::-1][crossfold:]:
            fullPath = happyDir+wavFile

toronto_data["joy"].append(sf.getFeatures(fullPath,allFeatures))
      for wavFile in os.listdir(sadDir)[::-1][crossfold:]:
            fullPath = sadDir+wavFile

toronto_data["sadness"].append(sf.getFeatures(fullPath,allFeatur
es))
      for wavFile in os.listdir(fearDir)[::-1][crossfold:]:
            fullPath = fearDir+wavFile

toronto_data["fear"].append(sf.getFeatures(fullPath,allFeatures)
)
      for wavFile in os.listdir(disgustDir)[::-1][crossfold:]:
            fullPath = disgustDir+wavFile

toronto_data["disgust"].append(sf.getFeatures(fullPath,allFeatur
es))
      return toronto_data
def loadBerlinData(crossfold,allFeatures):
```

```
	berlin_paths=dd(list)
	berlin_data=dd(list)
	rootDir = 'spr03/wav/'
	allwavs = os.listdir(rootDir)[::-1]
	for wavFile in allwavs:
		german_label=wavFile[-6]
		#print german_label
		#german_label = wavFile.split('.wav')[0][-2:-1]
		if german_label == 'A':
			berlin_paths['fear'].append(wavFile)
		elif german_label == 'E':
			berlin_paths['disgust'].append(wavFile)
		elif german_label == "F":
			berlin_paths['joy'].append(wavFile)
		elif german_label == "L":
			pass
			#berlin_paths['boredom'].append(wavFile)
		elif german_label == 'N':
			berlin_paths['neutral'].append(wavFile)
		elif german_label == "T":
			berlin_paths['sadness'].append(wavFile)
		elif german_label == 'W':
			berlin_paths['anger'].append(wavFile)
		else:
			print german_label
			print('Write good code.')
	for label, data in berlin_paths.iteritems():
		for sample in data[0:46][crossfold:]:
			#Disgust have 46 samples therefore we limit the
other samples for avoiding biased results
			berlin_data[label].append(sf.getFeatures(rootDir
+sample,allFeatures))
	#debugger.set_trace()
	return berlin_data

def loadWowData(allFeatures):
	rootDir = 'wow/'
	samples = os.listdir(rootDir)
	data = dd(list)
	for sample in samples:
		fullpath = rootDir+sample
		fullpath = ste_preprocess.preprocess(fullpath)
		label = sample.split('_')[1]

data[label].append((sf.getFeatures(fullpath,allFeatures),fullpat
h))
	return data
```

ste_decision_tree.py

```
from sklearn import tree
import numpy as np
from collections import defaultdict as dd
import json

def prepareDT(test,withLoad,training):
      C=1.0
      kernels = ['linear','rbf','poly']
      label_samples = dd(list)
      prob = dd(float)
      feature_len_s=50
      x=[]
      y=[]
      for label, data in training.iteritems():
            for sample in data:
                  #feature = (#frames,13)
                  if(len(label_samples[label])==0):
                        label_samples[label] = sample
                  else:
                        label_samples[label] =
np.concatenate((label_samples[label],sample))
            y.extend([label] * len(label_samples[label]))
            #debugger.set_trace()
            if(len(x)==0):
                  x = label_samples[label]
            else:
                  x = np.concatenate((x,label_samples[label]))

      clf = tree.DecisionTreeClassifier()
      clf = clf.fit(x, y)

      Z = clf.predict_proba(test)
      label_order =  clf.classes_
      for labelindex in xrange(len(label_order)):
            prob[label_order[labelindex]]=sum(Z[:,labelindex])/
feature_len_s
      return prob

def printProbs(prob):
      print prob.keys(),prob.values()
```

ste_hybrid.py

```
import pdb as debugger
import json
from collections import defaultdict as dd
def runHybrid(p_sp,p_te):
     #Weight according to the average accuracy of the
classifiers
     s = 51.875
     t = 21.875
     s = 100*s/(s+t)
     t = 100*t/(s+t)
     hybrid=dd(list)
     for s_targets,s_data in p_sp.iteritems():
          try:
               hybrid[s_targets]= s*p_sp[s_targets]
+t*p_te[s_targets]
          except TypeError:
               hybrid[s_targets]= p_sp[s_targets]
     return hybrid
```

ste_feature_extract.py

```
from features import mfcc
from features import logfbank
from features import ssc
from features import fbank
from features import delta

import scipy.io.wavfile as wav
import numpy as np
import pdb as debugger

from scipy.fftpack import fft
from scipy.io import wavfile # get the api
import math
import wave
from scipy import signal



import sys
#Dev mode only
#import matplotlib.pyplot as plt
#from matplotlib.mlab import find
from audiolazy.lazy_lpc import lpc
#import matplotlib.pyplot as plt




slice=13

'''
Slice 26: 100 classification with cross validation anger.
Filter Bank Energy Normalisation reduces the score.
'''
'''
def getF0(signal,samplerate):
      #http://stackoverflow.com/questions/9082431/frequency-
analysis-in-python
      signal = np.fromstring(signal, 'Int16');
      crossing = [math.copysign(1.0, s) for s in signal]
      index = find(np.diff(crossing));
      f0=round(len(index) *samplerate /(2*np.prod(len(signal))))
      return f0;

def convertToHz(freq,rate):
    return abs(freq * rate)

def getSpectrogram(sig,rate):
      f,t,sxx = signal.spectrogram(sig,rate)
      plt.pcolormesh(t,f,sxx)
      plt.ylabel('Frequency [Hz]')
```

```
      plt.xlabel('Time [sec]')
      plt.show()
      debugger.set_trace()

def frequencyFeatures():
      #freq testing
      frate = wav_file.getframerate()
      wav_file.close()

      FFT = np.fft.fft(sig)
      N=len(FFT)
      freqs = np.fft.fftfreq(len(FFT))

      #Find the peak in the coefficients
      idx = np.argmax(np.abs(FFT))
      freq = freqs[idx]
      peak_freq_in_hertz = abs(freq * frate)

      feat_1=np.array([mfcc_mean,mfcc_stdev])
      feat_2=np.array([fbank_mean,fbank_stdev])
      print path
      print peak_freq_in_hertz
      plt.plot(freqs,FFT)
      debugger.set_trace()
      plt.show()

def getFreqOfPeak(sig,rate):
      #mono = sig[:,0]
      FFT = np.fft.fft(sig)
      freqs = np.fft.fftfreq(len(sig))
      index = np.argmax(np.abs(FFT))
      freq = freqs[index]
      freq_in_hertz = abs(freq*rate)
      #frequency of maximum power.
      mean_freq = abs(np.mean(freqs) *rate)
      return freq_in_hertz
'''
def normalise(feats):
      return (np.subtract(feats,np.mean(feats)))/np.std(feats)

def getZeroCrossingsRate(sig):
      zero_crossings = np.where(np.diff(np.sign(sig)))[0]
      return len(zero_crossings)


def getEnergy(frame):
    return np.sum(frame ** 2) / np.float64(len(frame))
def getSpectralRollOff():
      energyTotal = numpy.sum(frame**2)

def prosodicFeatureSet(path):
      (rate,sig) = wav.read(path)

      wavfile = wave.open (path, "r")
      wav_params = wavfile.getparams()
      frame_length = wav_params[3]
```

```
        frames = wavfile.readframes(frame_length)
        frames = np.fromstring(frames, np.short).byteswap()
        #for frame in frames:
        #    print frame

        mfcc_feat = mfcc(sig,rate)
        fbank_feat = logfbank(sig,rate)
        ssc_feat = ssc(sig,rate)
        mfcc_mean = np.mean(mfcc_feat,axis=0)
        mfcc_stdev = np.std(mfcc_feat, axis=0)
        fbank_mean = np.mean(fbank_feat,axis=0)
        fbank_stdev = np.std(fbank_feat, axis=0)
        energy_mean = np.mean(fbank(sig,rate)[1])

        arr =
[getZeroCrossingsRate(sig),getFreqOfPeak(sig,rate),energy_mean]
        arr.extend(mfcc_mean)
        arr.extend(mfcc_stdev)
        arr.extend(fbank_mean)
        arr.extend(fbank_stdev)

        na = np.array(arr)
        na = na.reshape(27,3)
        return na

def deltaReshape(deltas):
        y=np.array([np.array(d) for d in deltas])
        return y
def calculateRow(matrix,feature_column_size):
        return matrix.size/feature_column_size
def applySlice(feat):
        return feat[0:slice,:]

def mfccOnly(rate,sig):
        mfcc_feat = mfcc(sig,rate)
        d_mfcc_feat = deltaReshape(delta(mfcc_feat, 2))
        d_d_mfcc_feat = deltaReshape(delta(d_mfcc_feat, 2))
        #mfcc_mean = np.mean(mfcc_feat,0)
        #mfcc_std = np.std(mfcc_feat,0)
        #mfcc_max = np.max(mfcc_feat,0)
        #mfcc_min = np.min(mfcc_feat,0)
        feat = np.concatenate((mfcc_feat,d_mfcc_feat),1)
        feat = np.concatenate((feat,d_d_mfcc_feat),1)
        return feat


def getFeatures(path,allFeatures):
        order=slice
        feature_column_size = 13
        LPC_feats = False
        sliceEnergy = False
        (rate,sig) = wav.read(path)

        mfcc_feat = mfcc(sig,rate)
        fbank_feat = logfbank(sig,rate)
        ssc_feat = ssc(sig,rate)
```

```
	d_mfcc_feat = deltaReshape(delta(mfcc_feat, 2))
	d_d_mfcc_feat = deltaReshape(delta(d_mfcc_feat, 2))

	mfcc_mean = np.mean(mfcc_feat,0)
	mfcc_std = np.std(mfcc_feat,0)
	mfcc_max = np.max(mfcc_feat,0)
	mfcc_min = np.min(mfcc_feat,0)

	if not allFeatures:
		mfcc_feat=applySlice(mfcc_feat)
		#ssc_feat = applySlice(ssc_feat)
		d_mfcc_feat = applySlice(d_mfcc_feat)
		d_d_mfcc_feat = applySlice(d_d_mfcc_feat)

	#data structure purposes

	row = calculateRow(fbank_feat,feature_column_size)
	fbank_feat = fbank_feat.reshape(row,feature_column_size)
	ssc_feat = ssc_feat.reshape(row,feature_column_size)
	if sliceEnergy:
		fbank_feat = applySlice(fbank_feat)

	mfcc_feat = normalise(mfcc_feat)

	#Do not normalise D's and DD's, they are calculated over
normalised MFFCs

	d_fbank_feat = deltaReshape(delta(fbank_feat,2))
	d_d_fbank_feat = deltaReshape(delta(d_fbank_feat,2))

	#Fbank deltas and delta deltas increases the results.

	#Better results without SSC

	feat = mfcc_feat
	feat = np.vstack((feat,mfcc_mean))
	feat = np.vstack((feat,mfcc_std))
	feat = np.vstack((feat,mfcc_max))
	feat = np.vstack((feat,mfcc_min))
	feat = np.vstack((feat,d_mfcc_feat))
	feat = np.vstack((feat,d_d_mfcc_feat))
	feat = np.vstack((feat,np.mean(fbank_feat,0)))
	feat = np.vstack((feat,fbank_feat[0]))

	#Pick one set
	feat = np.vstack((feat,d_fbank_feat[0]))
	feat = np.vstack((feat,d_d_fbank_feat[0]))

	#feat = np.vstack((feat,np.mean(d_fbank_feat,0)))
	#feat = np.vstack((feat,np.mean(d_d_fbank_feat,0)))



	#begin: This are also good features. Previous results
excluded below and enabled d_fbank_feat[0]
```

```
        feat = np.vstack((feat,np.std(d_fbank_feat,0)))
        #feat = np.vstack((feat,np.max(fbank_feat,0)))
        #feat = np.vstack((feat,np.min(fbank_feat,0)))
        #end
        feat = np.vstack((feat,np.mean(ssc_feat,0)))
        #debugger.set_trace()
        if LPC_feats:
              filt = lpc.kautocor(sig,feature_column_size-1)
              lpc_coef = np.array(filt.numerator)
              #lpc_feats = np.array(lpc(sig,feature_column_size)
[2])
              #lpc_feats = lpc(sig)
              #print lpc_feats
              feat = np.vstack((feat,lpc_coef))
        #print path, ps
        return feat
```

ste_knn.py

```
import numpy as np
from collections import defaultdict as dd
import pickle
from sklearn.neighbors import KNeighborsClassifier
from sklearn import neighbors
import pdb as debugger
from collections import Counter

def prepareKNN(test,withLoad,training):
      n_neighbors = 10
      label_samples = dd(list)
      prob = dd(float)
      #sample = every wav file feature
      x=[]
      y=[]
      if(withLoad):
            clf=loadClassifier('KNN_12_Sept_Toronto_All')
      else:
            for label, data in training.iteritems():
                  for sample in data:
                        #feature = (#frames,13)
                        if(len(label_samples[label])==0):
                              label_samples[label] = sample
                        else:
                              label_samples[label] =
np.concatenate((label_samples[label],sample))
                  y.extend([label] * len(label_samples[label]))
                  #debugger.set_trace()
                  if(len(x)==0):
                        x = label_samples[label]
                  else:
                        x =
np.concatenate((x,label_samples[label]))
            clf =
neighbors.KNeighborsClassifier(n_neighbors=n_neighbors,
weights='distance')
            clf.fit(x, y)
            #saveClassifier(clf,'KNN_12_Sept_Toronto_All')
      Z = clf.predict_proba(test)
      tags = clf.predict(test)
      label_order =  clf.classes_
      prob = resultParsingNew(Z,label_order,prob,tags)
      return prob
def resultParsingNew(Z,label_order,prob,tags):
      #Counter Analysis for unbiased solutions
      #counter = Counter(tags)
      #counter_prediction = counter.most_common()[0][0]
      #print counter_prediction
      #probability yields more fined tuned results
      result = sum(Z)
      totalscore = sum(result)
      result = result*100/totalscore
      for i in xrange(len(label_order)):
            prob[label_order[i]]=result[i]
```

```
      return prob
def printProbs(prob):
      print prob.keys(),prob.values()

def saveClassifier(classifier,name):
      #30m for 2 classes with full set
      f = open(name+'.pickle', 'wb')
      pickle.dump(classifier, f)
      f.close()

def loadClassifier(name):
      f = open(name+'.pickle', 'rb')
      classifier = pickle.load(f)
      f.close()
      return classifier
```

ste_preprocess.py

```
from pydub import AudioSegment
import pdb as debugger
from scipy import signal
recordedFromWeb=False
overwrite=True
def preprocess(path):
      if len(path.split('preprocessed')) < 2:
            sound = AudioSegment.from_file(path)
            #sound = sound.set_channels(1)
            #sound = sound.split_to_mono()[0]
            if not sound.sample_width == 2:
                  sound = sound.set_sample_width(2)
            #sound = sound.set_frame_rate(44100)
            if recordedFromWeb:
                  #There usually exists a click sound when
recorded from web. Slicing helps to remove the irrelevant start
noise.
                  sliced = sound[500:]
                  sound = sliced
                  #sound = sound.high_pass_filter(300)
                  #sound = sound.low_pass_filter(3400)
                  sound = sound.normalize()
                  #new_path = path.split('.wav')
[0]+'_preprocessed.wav'
            if overwrite:
                  file_handle = sound.export(path, format="wav")
            return path
      else:
            return path
```

ste_svm.py

```
from sklearn import tree
import numpy as np
from collections import defaultdict as dd
from sklearn.svm import SVC
import pickle
import json
import ste_result_parser as rparser
def prepareSVM(test,withLoad,training):
      C=1.0
      feature_len_s=50
      kernels = ['linear','rbf','poly']
      label_samples = dd(list)
      prob = dd(float)
      #sample = every wav file feature
      x=[]
      y=[]
      if(withLoad):
            clf=loadClassifier('toronto_svm_20_crossfold')
      else:
            for label, data in training.iteritems():
                  for sample in data:
                        #feature = (#frames,13)
                        if(len(label_samples[label])==0):
                              label_samples[label] = sample
                        else:
                              label_samples[label] =
np.concatenate((label_samples[label],sample))
                  y.extend([label] * len(label_samples[label]))
                  #debugger.set_trace()
                  if(len(x)==0):
                        x = label_samples[label]
                  else:
                        x =
np.concatenate((x,label_samples[label]))
            clf = SVC(probability=True,C=C,kernel='rbf')
            clf.fit(x, y)
            print 'Saving SVM'

#saveClassifier(clf,'toronto_svm_20_crossfold_linear')
      Z = clf.predict_proba(test)
      tags = clf.predict(test)
      label_order =  clf.classes_
      prob = rparser.resultParsingNew(Z,label_order,prob,tags)
      return prob

def printProbs(prob):
      print prob.keys(),prob.values()

def saveClassifier(classifier,name):
      #30m for 2 classes with full set
      f = open(name+'.pickle', 'wb')
      pickle.dump(classifier, f)
      f.close()
```

```
def loadClassifier(name):
      f = open(name+'.pickle', 'rb')
      classifier = pickle.load(f)
      f.close()
      return classifier
```

ste_result_parser.py

```
def resultParsingNew(Z,label_order,prob,tags):
      #Counter Analysis for unbiased solutions
      #counter = Counter(tags)
      #counter_prediction = counter.most_common()[0][0]
      #print counter_prediction
      #probability yields more fined tuned results
      result = sum(Z)
      totalscore = sum(result)
      result = result*100/totalscore
      for i in xrange(len(label_order)):
            prob[label_order[i]]=result[i]
      return prob
```

ste_text_classification.py

```
import nltk
import pdb as debugger
import os
import csv
from collections import defaultdict as dd
from nltk.tokenize import word_tokenize # or use some other
tokenizer
import pickle
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn import neighbors
C=1.0

def buildDataset():
      emotionDict = dd(list)
      with open('text_features/isear.csv', 'rb') as csvfile:
            reader = csv.reader(csvfile)
            for line in reader:
                  tag = line[0]
                  if applyFilter(tag):
                        rl = line[1].replace('\n','')
                        content = str(rl)
                        emotionDict[tag].append(content)
      return emotionDict
def applyFilter(tag):
      if tag != 'shame' and tag != 'guilt':
            return True
      else:
            return False
def predictText(test):
      #SVM Load: SVM_Text_12_Sept_Check
      willLoad=False
      ngram=2
      #look scikit-docs
      count_vect = CountVectorizer(ngram_range=(1,ngram))
      tfidf_transformer = TfidfTransformer()

      try:
            option='SVM'

            emotionDict=buildDataset()
            X=[]
            Y=[]
            for label, data in emotionDict.iteritems():
                  for sample in data:
                        X.append(sample)
                        Y.append(label)
            #Ngram optional
            X_train_counts = count_vect.fit_transform(X)
            X_train_tfidf =
tfidf_transformer.fit_transform(X_train_counts)
```

```
            if(option=="SVM"):
                  clf = SVC(probability=True,C=C)
                  #clf=loadClassifier("SVM_Text_Multiclass")
                  #1 gram RBF-SVM
                  clf=loadClassifier("SVM_Text_12_Sept_Check")
                  #2 gram RBF-SVM
            elif(option=='NB'):
                  clf = MultinomialNB()
            elif(option=='KNN'):
                  clf =
neighbors.KNeighborsClassifier(n_neighbors=10,
weights='distance')
            else:
                  print "Classifier not found."

            if not willLoad:
                  clf.fit(X_train_tfidf,Y)
            X_test_counts = count_vect.transform([test])
            X_test_tfidf =
tfidf_transformer.transform(X_test_counts)
            predicted = clf.predict_proba(X_test_tfidf)
            probdict = dd(int)
            for i in xrange(len(clf.classes_)):
                  probdict[clf.classes_[i]]=predicted[0][i]*100
            return probdict
      except AttributeError, e:
            print("Audio cannot be translated skipping text
classficiation")
            print e
def saveClassifier(classifier,name):
      f = open(name+'.pickle', 'wb')
      pickle.dump(classifier, f)
      f.close()

def loadClassifier(name):
      f = open(name+'.pickle', 'rb')
      classifier = pickle.load(f)
      f.close()
      return classifier
```

ste_translator.py

```
#!/usr/bin/env python3
#Source
#https://github.com/Uberi/speech_recognition/blob/master/
examples/audio_transcribe.py
import speech_recognition as sr
from os import path
import pdb as debugger
r = sr.Recognizer()
class translator:
     def translate(self,path):
          # obtain path to "english.wav" in the same folder as
this script
          AUDIO_FILE = path
          #AUDIO_FILE =
path.join(path.dirname(path.realpath(__file__)), "french.aiff")
          #AUDIO_FILE =
path.join(path.dirname(path.realpath(__file__)), "chinese.flac")
          # use the audio file as the audio source
          with sr.WavFile(AUDIO_FILE) as source:
              audio = r.record(source) # read the entire audio
file
          # recognize speech using Sphinx
          content = ''
               # recognize speech using Google Speech
Recognition
          try:
              # for testing purposes, we're just using the
default API key
              # to use another API key, use
`r.recognize_google(audio,
key="GOOGLE_SPEECH_RECOGNITION_API_KEY")`
              # instead of `r.recognize_google(audio)`
               content = r.recognize_google(audio)
              #print("Google Speech Recognition thinks you said
" + r.recognize_google(audio))
              #debugger.set_trace()
          except sr.UnknownValueError:
              print("Google Speech Recognition could not
understand audio")
          except sr.RequestError as e:
              print("Could not request results from Google
Speech Recognition service; {0}".format(e))
          return content
```

Web Application

```
import sys
pyvocoder_dir = 'bin/pyvocoder-master'
sys.path.append(pyvocoder_dir)

from flask import Flask
import pdb as debugger
import os, sys
parent = os.path.dirname(os.path.abspath(__file__))
superparent =
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
sys.path.append(superparent)
sys.path.append(parent)
import bin.STE
from flask import render_template
from flask import request
from flask import url_for
from flask import request
from flask import redirect
from werkzeug.utils import secure_filename
from flask import send_from_directory
from collections import defaultdict as dd
import scipy.io.wavfile as wavfile
import numpy as np

#from Flask Docs
UPLOAD_FOLDER = 'uploads'
ALLOWED_EXTENSIONS = set(['wav','mp3'])
app = Flask(__name__, static_url_path='/static')
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
app.config['PREFERRED_URL_SCHEME']='https'
local=False
@app.route("/process/<file>")
def index(file):
     #result = runSTE()
     print file

@app.route('/')
def hello(name=None):
     name = 'Ugur'
     return render_template('index.html', name=name)

@app.route('/information')
def information(name=None):
     return render_template('information.html')


if __name__ == "__main__":
    app.run()

@app.route('/process/<filename>')
def processFile(filename):
     instance = STE.webRun(filename)

@app.route('/user/<username>')
def show_user_profile(username):
    # show the user profile for that user
```

```
    return 'User %s' % username

@app.route('/uploads/<filename>')
def uploaded_file(filename):
     #pdb.set_trace()
     #sample =
wavfile.read(open('uploads/'+filename.encode('UTF-8'), 'r'))
     response = bin.STE.webRun(filename,True)
     response = response[0]
     return render_template('result.html', response=response)
    #return
send_from_directory(app.config['UPLOAD_FOLDER'],filename)

def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1] in ALLOWED_EXTENSIONS

@app.route('/upload', methods=['GET', 'POST'])
def upload_file():
     applyPP=False
     if request.form.get('applyPP')==u'true':
          applyPP=True

     want_https=False
     app.logger.debug(request.files['the_file'])
     if request.method == 'POST':
          # check if the post request has the file part
          if 'the_file' not in request.files:
               flash('No file part')
               return redirect(request.url)
          print request.files
          file = request.files['the_file']
          # if user does not select file, browser also
          # submit a empty part without filename
          print(file.filename)
          if file.filename == '':
               flash('No selected file')
               return redirect(request.url)
          if file and allowed_file(file.filename):
               filename = secure_filename(file.filename)

file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
               if local:
                    return
redirect(url_for('uploaded_file',filename=filename))
               else:
                    return
redirect(url_for('uploaded_file',filename=filename,
_external=True, _scheme='https'))
               #return
redirect(url_for('uploaded_file',filename=filename))

               #return
redirect(url_for('processFile',filename=filename))

def shutdown_server():
```

```
      func = request.environ.get('werkzeug.server.shutdown')
      if func is None:
            raise RuntimeError('Not running with the Werkzeug
Server')
      func()
@app.route('/shutdown', methods=['POST'])
def shutdown():
      shutdown_server()
      return 'Server shutting down...'
```

STE.py

```
#Essentials
import numpy as np
import sys
import io
import os
import pickle
#STE-Classes
import speech_recognition as sr
import mfcc as sf
import ste_knn
import ste_preprocess as preproc
#Development only
import pdb as debugger
from collections import defaultdict as dd
#Text
from nltk.stem.porter import *
#Audio, MP3-Wav Conversion
import wave
from pydub import AudioSegment
#Classifiers
from sklearn.neighbors import NearestNeighbors
import json
import operator

r = sr.Recognizer()
encoding = 'utf-8'
experimentalTest="exp/"
allFeatures = False
def webRun(fileurl,applyPreprocessing):
      print 'Running from web'
      fileurl = 'uploads/'+fileurl.encode('utf-8')
      if applyPreprocessing:
            fileurl = preproc.preprocess(fileurl)
            print "Preprocessing applied"
      test = dd(list)
      appendExperimentalTestData(test,fileurl,'test_sample')
      return classify('KNN',True,False,None,test)

def appendExperimentalTestData(testset,samplepath,target):
      #testset: previously existing test set
      #samplepath: new test data's path
      #target: expected target

testset[target].append((sf.getFeatures(samplepath,allFeatures),s
amplepath))

def classify(clfoption,speech,text,training,test):
      response = []
      #Classify with options. clfoption=classifier, speech = use
speech features, text = use text features
      #training = the training data we are going to use
      for label,data in test.iteritems():
            #print "Input" + path
```

```
            print 'Correct class: ',label
            for sample,path in data:
                  print "Input: ",path
                  #Classifier options: test sample, use the
previously saved classifier
                  #and the training data set we are going to use
for classification.
                  ps = {}
                  print 'Using KNN Classifier with k =',10
                  ps =  ste_knn.prepareKNN(sample,True,training)
                  response.append(formatJSON(ps))
                  print formatJSON(ps)
                  print getCorrectResult(ps)
                  print '\n'
      return response
def formatJSON(prob):
      return json.dumps(prob, ensure_ascii=False)
def getCorrectResult(prob):
      result = max(prob.iteritems(), key=operator.itemgetter(1))
[0]
      return result

if __name__ == '__main__':
      main()
```