

Client Recognition Using Text-Dependent Voice Biometrics: A Machine Learning and Deep Learning Approach

Nadia Tahseen

Electrical, Electronics, and Telecommunication Engineering, University of Engineering and Technology, Lahore New Campus, Pakistan

Muhammad Usama

Electrical, Electronics, and Telecommunication Engineering, University of Engineering and Technology, Lahore New Campus, Pakistan

Ahmer Tabassum

Electrical, Electronics, and Telecommunication Engineering, University of Engineering and Technology, Lahore New Campus, Pakistan

Wasib Ilyas

Electrical, Electronics, and Telecommunication Engineering, University of Engineering and Technology, Lahore New Campus, Pakistan

ABSTRACT

Human-machine interaction increasingly relies on natural spoken language, which makes robust and efficient speaker recognition a critical enabling technology. In this work we present a text-dependent speaker (client) recognition system built around the common Islamic greeting phrase “Assalam-Alaikum”, chosen so that the system is directly relevant to a large global user base. We collected a dataset of 500 speakers (250 male and 250 female), each contributing four utterances of the target phrase, for a total of 2000 raw audio recordings. The recordings were pre-processed to remove silence, normalized in duration, and augmented using shifting, stretching, noise injection, pitch shifting and Vocal Tract Length Perturbation (VTLP) to expand the dataset to 10,000 samples. Five acoustic feature families were extracted using Librosa: zero-crossing rate, root-mean-square energy, Mel-spectrogram, Mel-Frequency Cepstral Coefficients (MFCCs, including first- and second-order derivatives), and Chroma STFT. The resulting feature set was encoded, scaled and used to train ten classical machine-learning classifiers (Decision Tree, Gaussian Naïve Bayes, Linear Discriminant Analysis, K-Nearest Neighbors, Support Vector Machines with linear/polynomial/RBF kernels, Logistic Regression, SoftMax Regression and XGBoost) as well as a four-layer Recurrent Neural Network (RNN) implemented in Keras. Logistic Regression and SoftMax Regression achieved the best classical accuracy at 96% on the held-out test set, while the RNN model achieved 97.33% test accuracy and 96.60% accuracy on a separate, previously unseen validation set. These results indicate that MFCC-based representations, combined with data augmentation and a modest deep-learning architecture, can

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

deliver highly reliable text-dependent speaker identification even across a relatively large, class-balanced speaker population of 500 individuals.

Keywords— Speaker Recognition, Voice Biometrics, MFCC, Feature Extraction, Data Augmentation, Machine Learning, Recurrent Neural Network, Text-Dependent Speaker Identification

I. Introduction

It has become well accepted that people interact with electronic devices using natural language, whether in English or another language. With cutting-edge technologies such as Apple's Siri and Microsoft's Kinect, it has become essential for machines to understand and act upon human speech. Realizing this goal requires improving the accuracy of speech-directed applications even for the most ordinary of tasks – such as deciding whether a person is speaking at a given moment, i.e., human voice detection. Because there are more than two billion Muslims worldwide, we selected the common greeting phrase “*Assalam-Alaikum*” as the target utterance for our text-dependent recognition task, so that the resulting system would be relevant to a very large potential user base.

Human speech is a signal that carries several types of information simultaneously, including lexical content, emotion, language, and the identity of the speaker. It is the task of the receiving system to decode this signal and extract the information that is required, often using different techniques for different pieces of information. Many recognition tasks assume single-speaker input; however, a great deal of real-world audio consists of long, multi-speaker streams such as meetings and news broadcasts, and it is impractical to build hardware that records only a clean human voice with no ambient noise or overlapping speech. At the same time, real-time performance is one of the most important requirements of any human–speech interaction system. Effective, highly accurate and time-efficient methods are therefore necessary for processing large volumes of speech data.

A. Project Scope and Objectives

Speaker identification is typically performed by building a sufficiently informative model of a speaker (or speakers) and using this model to identify the speaker of an unseen test utterance, a paradigm most classically realized through Gaussian Mixture Models built on top of a speaker-independent universal background model [6, 7]. New techniques are usually justified empirically, i.e., by comparison against established baseline approaches; even when the same downstream methods are used, a new representation must demonstrate superiority in a real, or realistically simulated, system. We hypothesized that Mel-Frequency Cepstral Coefficients (MFCCs) would enhance the performance of a text-dependent speaker-related task, based on the strong performance of this representation in related segmentation tasks. This proposition was evaluated empirically using standard evaluation metrics. The specific objectives of the project were to:

- survey current state-of-the-art methods for speaker-related tasks;
- study empirical evaluation methods commonly used by the research community;
- identify the statistical metric and speaker model that is most faithful to speaker identity when using speaker-specific representations; and apply these findings to improve speaker

verification and speaker identification performance.

The project falls within the broader area of speech and digital signal processing. In addition to speaker verification and identification, it touches on related tasks such as voice activity detection. It can also be viewed through the lens of online processing (where only information from earlier parts of the stream is available) versus offline processing (where information from the entire stream can be used), i.e., the study of temporal effects; both online and offline algorithms were therefore considered relevant to the project.

II. Client Recognition Using Text-Dependent Voice

A. Dataset

The audio dataset prepared for client recognition contains 500 speakers in total, comprising 250 male and 250 female speakers so as to maintain a balanced gender ratio. Each speaker contributed 4 recordings, giving a total of 2000 raw voice samples. Audio was originally collected in .m4a format and converted to .wav, since the *Librosa* library used for loading and reading audio does not support the .m4a format. Because the lexical content of every utterance is fixed to the single phrase “Assalam-Alaikum”,

the dataset is text-dependent rather than text-independent. Sample files are named using the convention $S_x_V_y$, where x denotes the speaker number and y denotes the voice/recording number; splitting the filename on the hyphen yields usable class labels. Most recordings are approximately 1 second long, although duration varies between 1 and 3 seconds because speakers differ in speaking rate and accent.

B. Overview of the Recognition Pipeline

The complete client-recognition pipeline consists of the following stages: (1) creation of the audio dataset, (2) data visualization and preprocessing, (3) data augmentation, (4) feature extraction, (5) data splitting and scaling, (6) model training and testing, and (7) validation accuracy and classification re-reporting. The overall flow is illustrated in Figure 1.

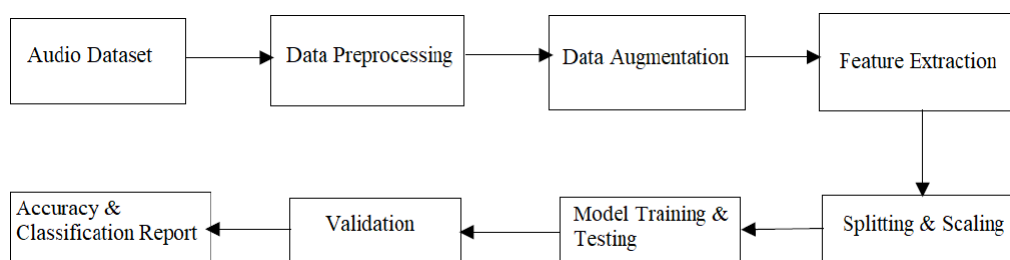


Figure 1: Flow diagram of text-dependent voice recognition.

C. Data Visualization and Preprocessing

Sound is fundamentally a series of vibrations with time-varying pressure, so visualizing sound is anal-ogous to visualizing airwaves. Data visualization renders such signals graphically, making it easier to observe trends, outliers and patterns using charts and plots. In this work, the audio data was visualized to inspect features such as amplitude and duration.

Figure 2 shows the waveform of sample $S81_V1$ in the time domain. An audio signal can also

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

be represented in the frequency domain via the Fourier Transform, which decomposes a time-domain signal into its constituent frequencies; the corresponding power spectrogram is shown in Figure 3.

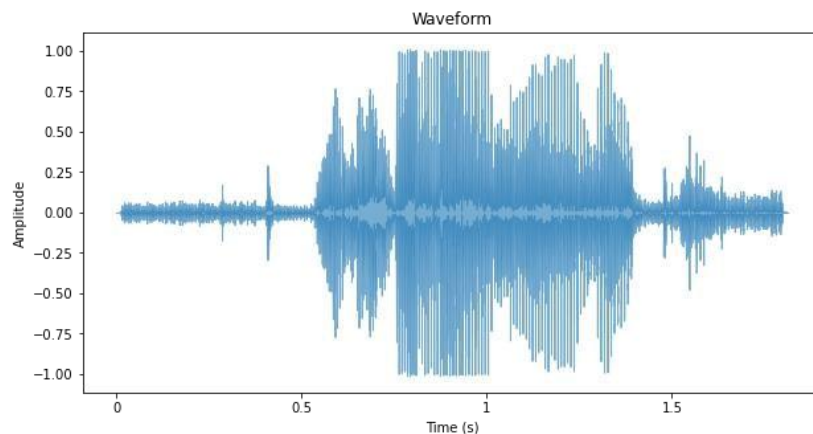


Figure 2: Audio waveform of S81_V1 in the time domain.

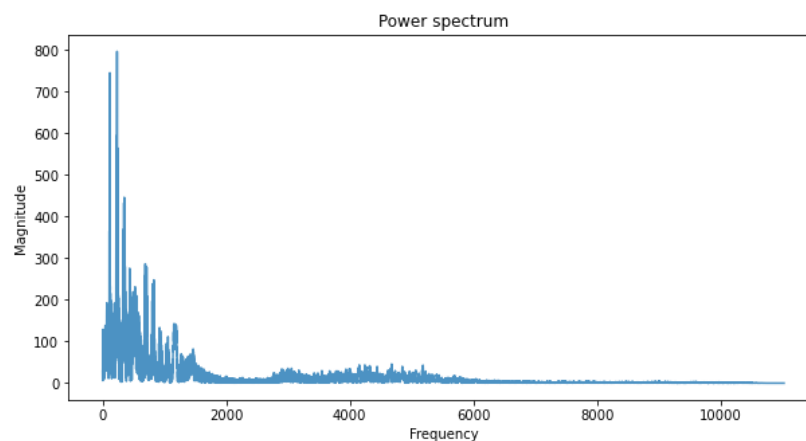


Figure 3: Audio waveform of S81_V1 in the frequency domain.

Preprocessing is a data-mining step that transforms raw data into a usable, efficient format, and generally comprises three stages: data cleansing, data transformation, and data reduction. Data cleansing

removes noise and silence, handles missing values, and deals with other data-quality issues; data transformation involves attribute selection, normalization and discretization; and data reduction involves dimensionality and numerosity reduction. Because the dataset used in this work was neither excessively large nor missing values, extensive data reduction was not required.

For feature extraction all audio clips must have the same number of frames; if all clips have equal duration they naturally have equal frame counts, but, as noted above, our recordings vary

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

in duration. One option is to pad the audio with silence up to the length of the longest recording; however, since roughly 1800 of the 2000 recordings are shorter than 2 seconds while only around 200 exceed 3 seconds, padding relative to the longest clip would introduce excessive silence and distort the dataset. Figure 4 shows the number of frames before any silence-handling was applied.

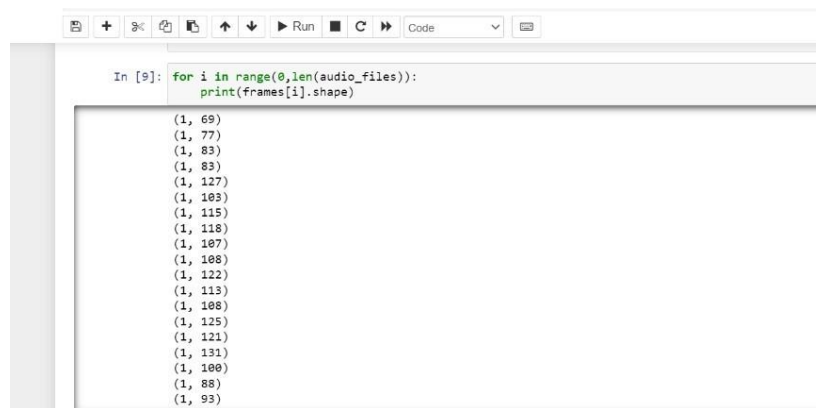


Figure 4: Number of frames before adding silence.

To address this, three helper functions were defined: *envelope*, *cut*, and *right_pad*. The *envelope* function isolates the audio portion without silence, the *cut* function truncates audio to 1.5 seconds, and any recording shorter than 1.5 seconds is zero-padded to the same length. The resulting frame counts after this preprocessing step are shown in Figure 5.

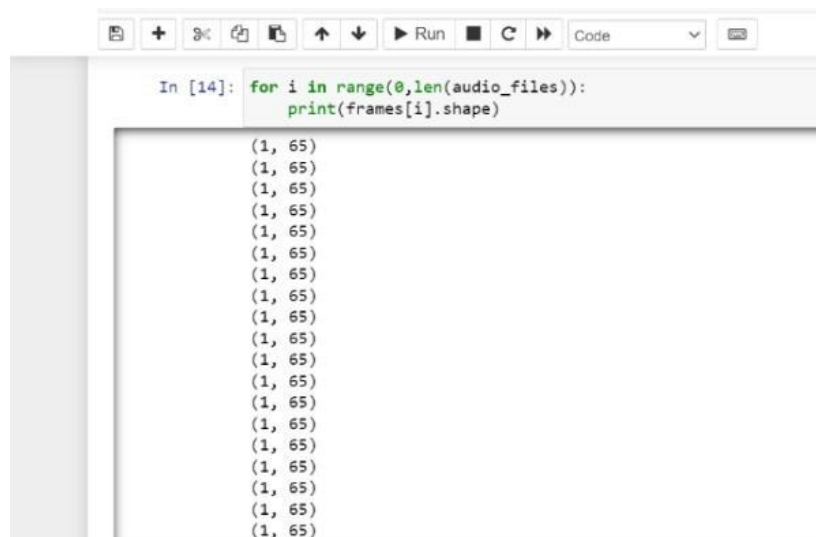


Figure 5: Number of frames after preprocessing.

An alternative solution – and the one ultimately adopted in this project – is to compute the

Online ISSN

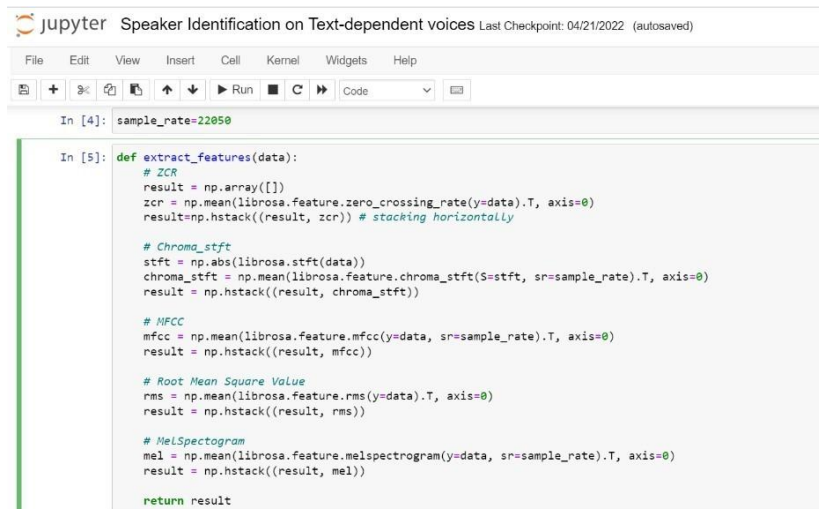
3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

mean of each extracted feature function across frames. Once every audio clip is passed through the same feature-extraction pipeline, the resulting feature vectors are of identical length regardless of the original clip duration (Figure 6).



```
In [4]: sample_rate=22050

In [5]: def extract_features(data):
# ZCR
result = np.array([])
zcr = np.mean(librosa.feature.zero_crossing_rate(y=data).T, axis=0)
result=np.hstack((result, zcr)) # stacking horizontally

# Chroma_stft
stft = np.abs(librosa.stft(data))
chroma_stft = np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).T, axis=0)
result = np.hstack((result, chroma_stft))

# MFCC
mfcc = np.mean(librosa.feature.mfcc(y=data, sr=sample_rate).T, axis=0)
result = np.hstack((result, mfcc))

# Root Mean Square Value
rms = np.mean(librosa.feature.rms(y=data).T, axis=0)
result = np.hstack((result, rms))

# MelSpectrogram
mel = np.mean(librosa.feature.melspectrogram(y=data, sr=sample_rate).T, axis=0)
result = np.hstack((result, mel))

return result
```

Figure 6: Feature extraction using the mean of each feature across frames.

D. Feature Extraction

Feature extraction converts raw data into numerical features that retain the salient information in the original signal while typically improving downstream model performance relative to using raw wave-forms directly. Extracting features that capture speaker identity from voice signals is a central challenge in speaker recognition; the goal is to transform the waveform into a parametric representation suitable for further analysis.

Feature extraction can be performed manually – by identifying and engineering task-relevant characteristics based on domain knowledge, e.g., the mean of a signal window – or automatically, using special-ized algorithms or deep networks (e.g., wavelet scattering) without manual intervention. MFCC-based representations in particular have repeatedly been shown to be effective and computationally efficient for speaker- and speech-related recognition tasks [8, 9, 11]. In this work, the automated feature-extraction capabilities of *Librosa* were used to extract the following features:

- Zero-Crossing Rate (ZCR)
- Root Mean Square (RMS)
- Mel-Spectrogram
- Mel-Frequency Cepstral Coefficients (MFCCs)
- Chroma STFT

1) Zero-Crossing Rate

A zero-crossing occurs where the sign of a signal changes from positive to negative or vice versa; the zero-crossing rate (ZCR) of an audio frame measures how frequently this occurs and is a widely used acoustic feature. ZCR can be interpreted as a crude measure of signal noise and spectral content: noisier signals tend to exhibit higher ZCR values. Because speech is inherently wideband, the average ZCR is a comparatively coarse indicator, though short-time

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

ZCR analysis can still yield useful spectral estimates. Figures 7 and 8 show, respectively, the zero-crossing rate of a sample audio clip and its frequency-time representation, computed using *librosa.display.specshow* and *matplotlib*.

2) Mel-Frequency Cepstral Coefficients (MFCCs)

MFCCs are among the most widely used features for speech-signal representation. The Mel-frequency cepstrum (MFC) represents the short-term power spectrum of a sound based on a linear cosine transform of the log power spectrum on a nonlinear Mel scale of frequency, and MFCCs are the coefficients that collectively make up this representation. Because the Mel scale more closely approximates the human

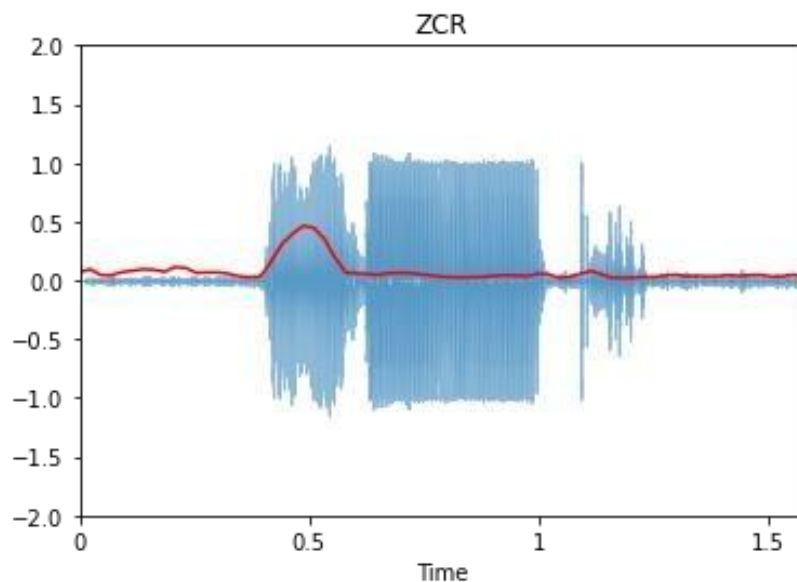


Figure 7: Zero-crossing rate (ZCR) of an audio sample.

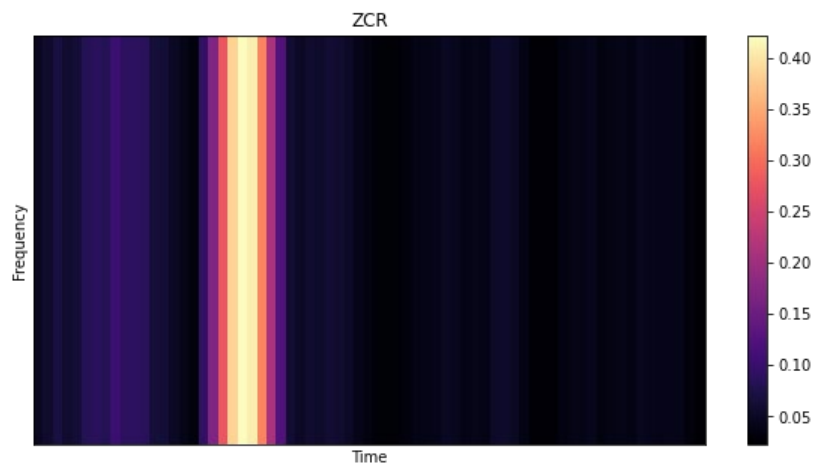


Figure 8: Frequency-time graph of the ZCR of an audio sample.

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

auditory system than a purely linear cepstral representation [1], MFCCs derived from this nonlinear representation tend to capture phonetically relevant properties of speech more effectively.

The critical bandwidths of the human ear vary with frequency, which motivates the MFCC computation: filters are spaced linearly at low frequencies and logarithmically at high frequencies to capture the phonetically relevant properties of the signal (Figure 9).

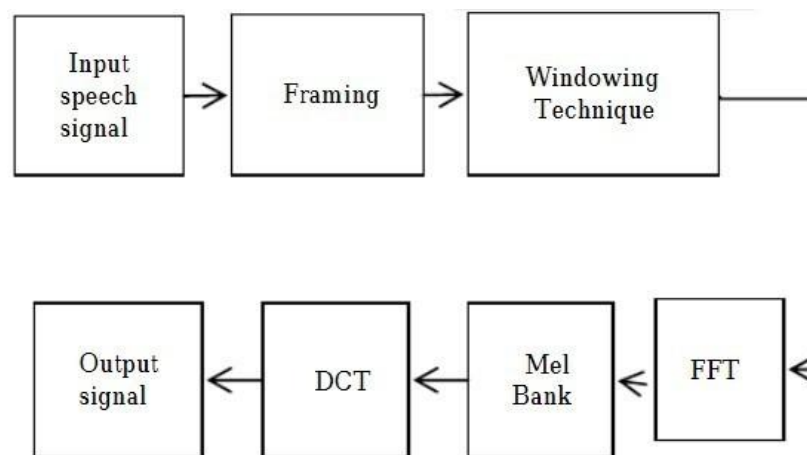


Figure 9: Block diagram of the MFCC computation pipeline.

The MFCC computation applies a Discrete Fourier Transform to convert the signal from the time domain to the frequency domain, after first applying a Hamming window to reduce spectral leakage introduced by segmentation. The resulting spectrum is warped from Hertz to the Mel scale via a filter bank, and feature vectors are finally extracted from the logarithm of the Mel-scaled power spectrum using a Discrete Cosine Transform (DCT). Thirteen MFCC coefficients were computed for speaker identification, as illustrated in Figure 10.

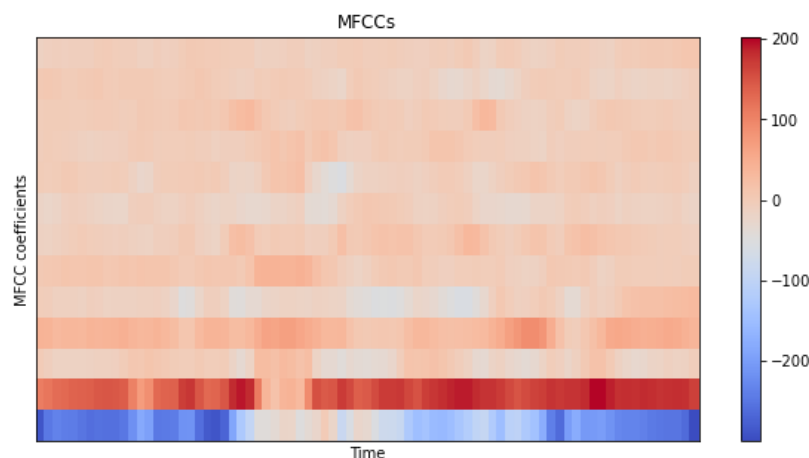


Figure 10: MFCC representation with 13 coefficients for a sample audio clip.

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

After computing the base MFCCs, their first derivative with respect to time was computed using the delta function of *Librosa*, in order to capture temporal dynamics. A second derivative (delta-delta, order 2) was then computed and concatenated with the original coefficients, yielding a final representation of 39 MFCC coefficients (13 base + 13 delta + 13 delta-delta), shown in Figure 11.

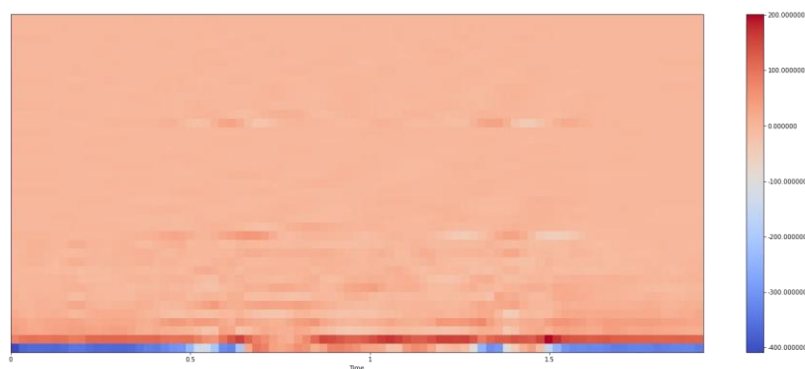


Figure 11: MFCC representation with 39 coefficients, including first and second temporal derivatives.

3) *Root Mean Square (RMS)*

RMS is a metering tool that estimates the average loudness of an audio recording, typically over a 300-millisecond interval, and provides a reasonable approximation of a listener's perceived loudness. While RMS is a useful reference point for a track's overall loudness, it remains an average across the entire waveform, meaning that a very short clip may not read as loud as a longer one. RMS metering approximates human auditory perception, which is important because the human ear does not respond equally to all frequencies. Figures 12 and 13 show the RMS of a sample audio clip and its frequency-time representation.

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

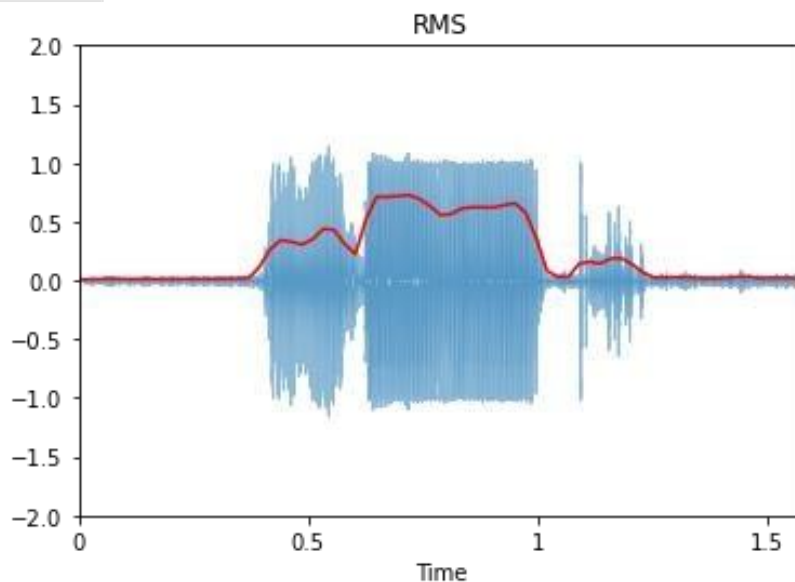


Figure 12: Root Mean Square (RMS) of an audio signal.

4) *Chroma STFT*

Chroma features were computed via a short-term Fourier transform (STFT), which captures information about pitch and signal structure. The short-time Fourier transform of the audio signal was computed and its magnitude taken using the `abs` function of *NumPy*. Figure 14 shows the resulting Chroma STFT representation for a sample audio signal.

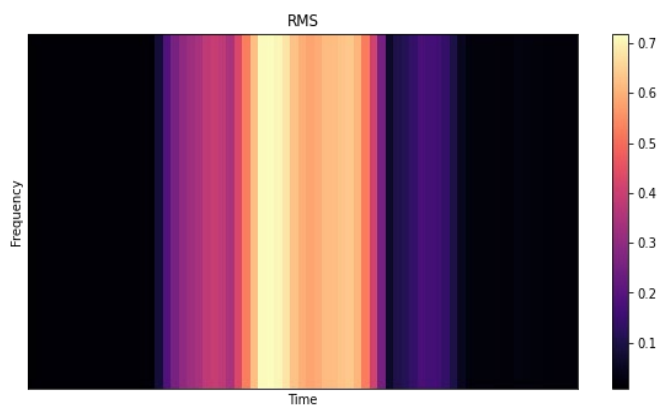


Figure 13: Frequency-time graph of the RMS of an audio signal.

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

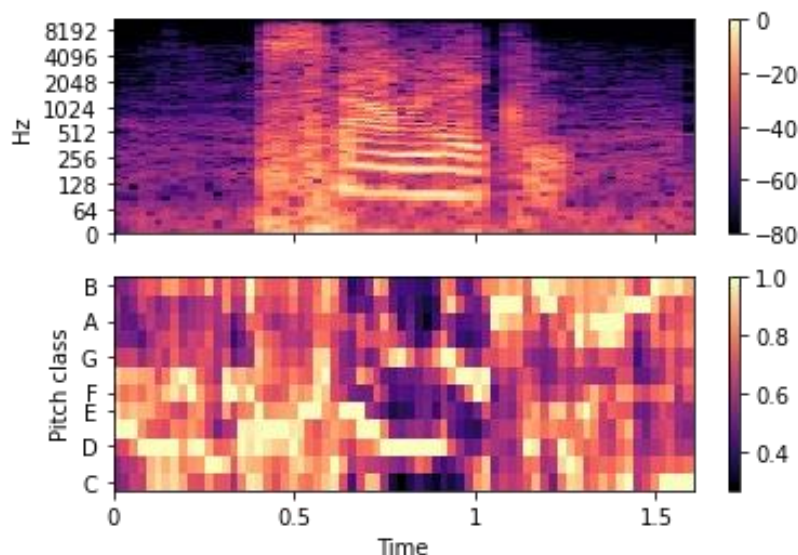


Figure 14: Chroma STFT of an audio signal.

5) *Mel-Spectrogram*

The Fast Fourier Transform (FFT) is useful for analyzing the frequency content of a signal, but most real-world audio signals – including speech and music – are non-stationary, meaning their frequency content changes over time. A spectrogram addresses this by computing the FFT over overlapping windowed segments of the signal, yielding a time-varying representation of the frequency spectrum; converting this representation to the Mel scale produces a Mel-spectrogram. Figure 15 shows a simple (non-Mel) spectrogram of an audio signal, and Figure 16 shows the same spectrogram converted to a decibel (dB), log-amplitude scale, which better reflects human auditory perception.

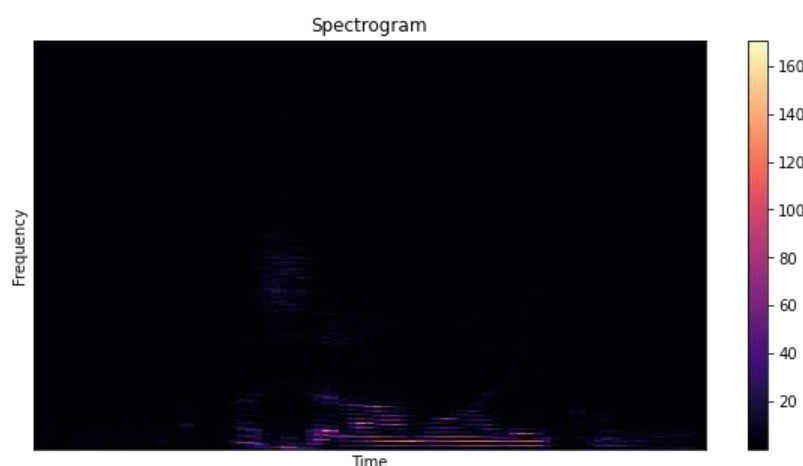


Figure 15: Spectrogram of an audio signal.

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

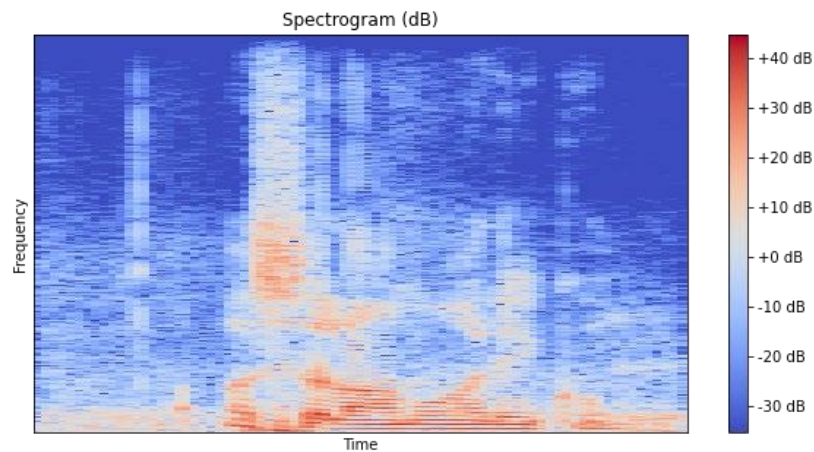
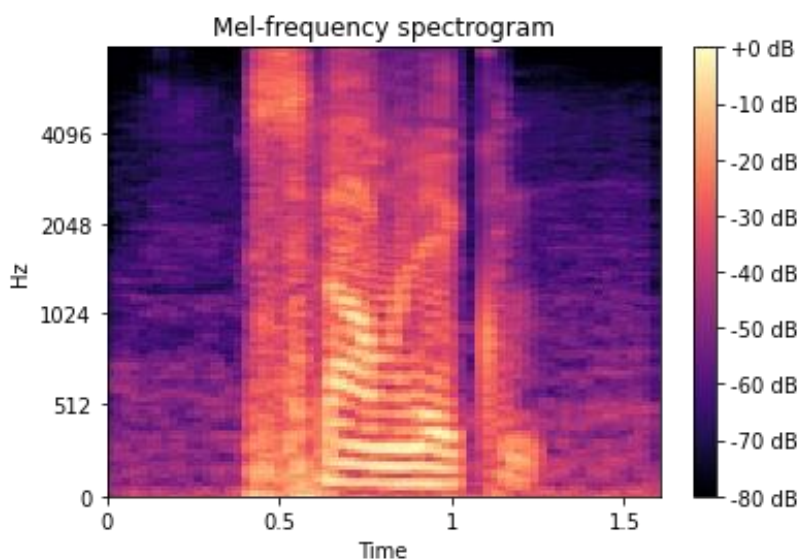


Figure 16: Spectrogram of an audio signal expressed in decibels (dB).

Humans do not perceive frequency linearly: lower-frequency differences are easier to detect than equally-sized differences at higher frequencies (e.g., 500 Hz vs. 1000 Hz is easily distinguished, whereas 10,000 Hz vs. 10,500 Hz is not, despite an identical 500 Hz gap). Stevens, Volkmann and Newman proposed the Mel scale in 1937 specifically to make equal perceptual pitch distances correspond to equal scale distances; frequencies are converted to Mel values via a standard mathematical transform, and the resulting Mel-scaled spectrogram is shown in Figure 17.

E. Data Augmentation

Data augmentation generates synthetic training samples by applying controlled perturbations to existing samples, expanding the effective size of the dataset while also providing varied instances of the same



Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

Figure 17: Mel-spectrogram of an audio signal.

underlying sample; this helps the trained model generalize better and reduces overfitting. In general, augmentation can be applied offline (ahead of time, expanding the physical dataset by a factor equal to the number of transformations applied – well suited to smaller datasets) or online / “on the fly” (applied to mini-batches during training – better suited to larger datasets, since it avoids an unmanageable increase in stored data volume) [3]. It has also been shown that synthetic augmentation of MFCC features via transfer learning from a character-level RNN can improve speaker recognition from spoken utterances [4].

The following augmentation techniques were applied in this project:

- Time Shifting
- Time Stretching
- Noise Addition
- Pitch Shifting
- Vocal Tract Length Perturbation (VTLP)

An un-augmented reference audio signal is shown in Figure 18.

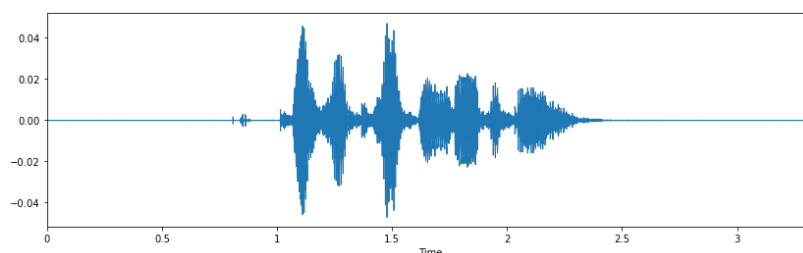


Figure 18: Audio signal without augmentation.

1) *Vocal Tract Length Perturbation (VTLP)*

VTLP perturbs a spectrogram by warping its frequency axis using a randomly generated warp

factor, mapping an original frequency f to a new frequency f' via a warping function. Although the resulting transformation is usually imperceptible to human listeners, its effect is clearly visible when plotting power against frequency. VTLP was implemented using the `VtlpAug` function from the *nlpaug* Python package; for the example audio used here, the boundary (cut-off) frequency is approximately 12 kHz.

Figure 19 illustrates the effect of VTLP on a sample signal.

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

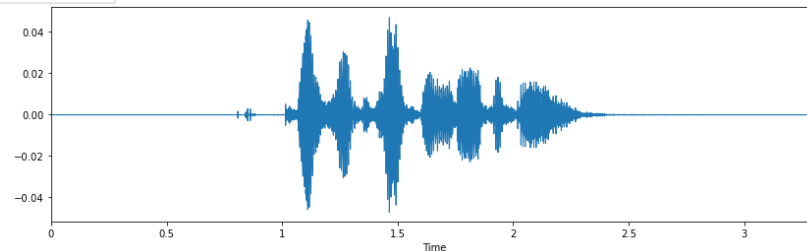


Figure 19: VTLP applied to an audio signal.

2) *Time Shifting*

Time shifting displaces the signal along the time axis by a specified number of periods, implemented via Librosa's shift function; this is a common augmentation for time-series data (Figure 20).

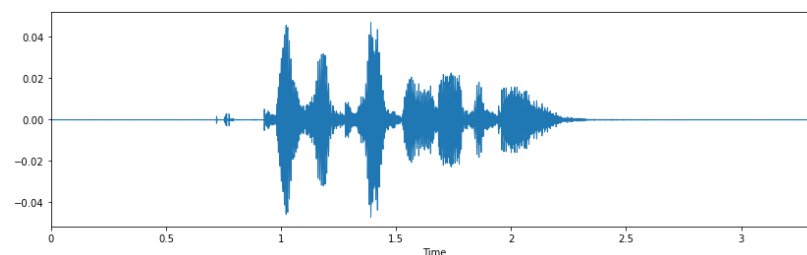


Figure 20: Time-shifted version of an audio signal.

3) *Time Stretching*

Time stretching adjusts the speed/duration of an audio clip without altering its pitch, implemented using Librosa's time_stretch function, which accepts the waveform samples and a stretch factor. A stretch factor of 0.4 was used, chosen to introduce a modest but meaningful variation relative to the original sample (Figure 21).

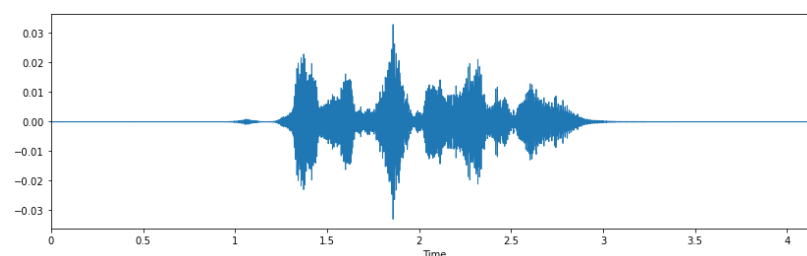


Figure 21: Time-stretched version of an audio signal.

4) *Noise Addition*

White noise – random samples drawn from a distribution with mean 0 and standard deviation 1 – was added to each sample using NumPy's normal-distribution sampling function, combined

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

With the original waveform (Figure 22).

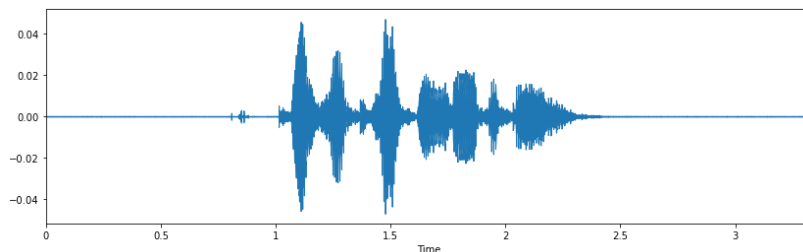


Figure 22: Effect of noise addition on an audio signal.

5) Pitch Shifting

Pitch shifting alters the pitch of a sound without changing its speed or duration, and is commonly used in musical instrument processing. This was implemented using Librosa's pitch-shift function, which takes as input the waveform samples, the sampling rate, and the number of semitone steps by which to shift the pitch (Figure 23).

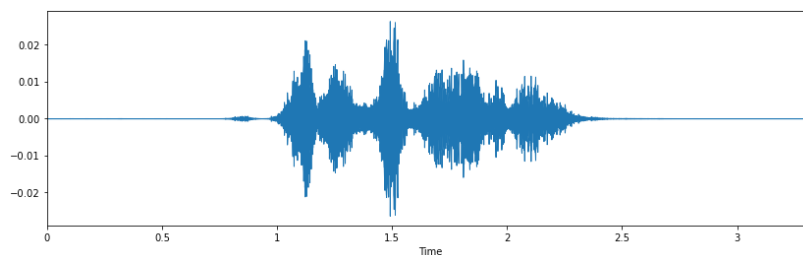


Figure 23: Pitch-shifted version of an audio signal.

6) Feature Dataset Construction

After preprocessing, augmentation and feature extraction, all audio data was converted into a single numeric dataset. Each raw audio clip was first expanded to 10,000 samples using the five augmentation techniques described above; each resulting sample was then passed through all feature-extraction functions, with the extracted features stacked horizontally into a single feature vector. Augmented samples inherited the class label of their original (un-augmented) source clip. The complete feature matrix was exported to a comma-separated

values file, `audio_features.csv`, using pandas' `.to_csv` function (Figure 24).

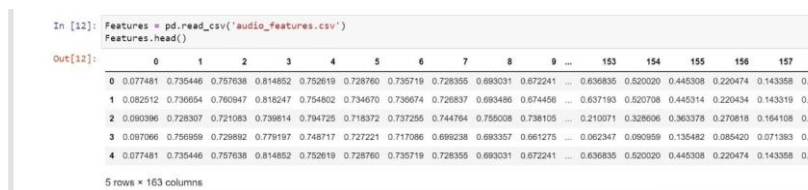


Figure 24: Structure of the extracted audio-features CSV file.

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

F. Encoding, Splitting and Scaling

1) *Categorical Encoding*

After constructing the CSV file, it was loaded using pandas, with the extracted features stored in a matrix X and the corresponding class labels stored in a vector Y . Two categorical encoding strategies were considered: Label Encoding and One-Hot Encoding.

One-Hot Encoding. While label encoding is simple, it risks causing learning algorithms to misinterpret purely nominal numeric class labels as having an inherent order or hierarchy. One-hot encoding avoids this by representing each category as a separate binary column (Table 1) [5].

Table 1: Representation of a categorical variable (traffic safety level) using One-Hot Encoding.

Safety Level	None	Low	Medium	High
None	1	0	0	0
Low	0	1	0	0
Medium	0	0	1	0
High	0	0	0	1

Label Encoding. Label encoding converts each categorical value directly to an integer code (Table 2); this method is simple but, as noted above, can introduce an artificial ordinal relationship among classes that do not actually possess one.

Table 2: Representation of a categorical variable (traffic safety level) using Label Encoding.

Safety Level	Encoded Category
None	0
Low	1
Medium	2
High	3

Because speaker identification is inherently a multi-class classification problem, One-Hot Encoding was applied to the label vector Y prior to model training.

2) *Splitting*

The dataset was split into 70% training and 30% held-out data using `train_test_split`, with the 30% further divided equally (50:50) into testing and validation subsets. In absolute terms, 7000 samples were used for training, 1500 for testing, and 1500 for validation (Figure 25).

The `stratify` parameter was used to ensure a uniform class distribution across all three subsets, and a fixed `random_state` was used to guarantee reproducibility across runs.

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

```
SEED = 12345

x_train, x, y_train, y = train_test_split(X, Y, test_size=0.3, random_state = SEED , shuffle=True, stratify=Y)
x_train.shape, y_train.shape, x.shape, y.shape
((7000, 162), (7000, 500), (3000, 162), (3000, 500))

x_test, x_valid, y_test, y_valid=train_test_split(x, y, test_size=0.5, random_state=SEED, shuffle=True, stratify=y)
x_test.shape, y_test.shape, x_valid.shape, y_valid.shape
((1500, 162), (1500, 500), (1500, 162), (1500, 500))
```

Figure 25: Data splitting scheme for training, testing and validation.

3) Feature Scaling

Feature scaling rescales numerical features onto a common range, preventing supervised models from becoming biased toward features with a naturally larger numeric range. For example, an unscaled linear-regression model may allow certain features to dominate the prediction simply due to their magnitude, distorting the learned relationships; scaling is not required, however, for tree-based methods such as random forests and decision trees. Two scaling strategies were used in this work.

Min–Max Scaler. The MinMaxScaler (from sklearn.preprocessing) rescales each feature independently so that it lies within a specified range, typically [0, 1] (Figure 26).

```
# scaling our data with sklearn's Standard scaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
x_valid=scaler.transform(x_valid)
x_train.shape, y_train.shape, x_test.shape, y_test.shape, x_valid.shape, y_valid.shape
((7000, 162), (7000, 500), (1500, 162), (1500, 500), (1500, 162), (1500, 500))
```

Figure 26: Feature scaling using scikit-learn's MinMaxScaler.

Standard Scaler. The StandardScaler removes the mean and scales each feature to unit variance on a feature-by-feature basis. Many learning algorithms – e.g., the RBF kernel in SVMs, or L1/L2 regularization in linear models – implicitly assume that features are centered around zero with comparable variance; if one feature has an order-of-magnitude larger variance than the others, it can dominate the objective function and impair learning from the remaining features (Figure 27).

```
# scaling our data with sklearn's Standard scaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
x_valid=scaler.transform(x_valid)
x_train.shape, y_train.shape, x_test.shape, y_test.shape, x_valid.shape, y_valid.shape
((7000, 162), (7000, 500), (1500, 162), (1500, 500), (1500, 162), (1500, 500))
```

Figure 27: Feature scaling using scikit-learn's StandardScaler.

4) Dimensionality Reduction

Principal Component Analysis (PCA) was additionally used to reduce the dimensionality of

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

the feature set by summarizing a large collection of (possibly correlated) variables into a

smaller set of components that retain most of the information in the original data. PCA improves the efficiency and, in some cases, the accuracy of downstream models by removing correlated variables that do not contribute meaningfully to classification, thereby reducing overfitting and improving visualization while also reducing training time.

III. Model Training and Evaluation

Model training aims to find the combination of weights and biases that minimizes a chosen loss function over the training data. Model development proceeds in two broad stages: training/validation, in which algorithms learn patterns between input features and target labels for data with known outcomes, and testing, in which the learned model is evaluated on unseen data. Two categories of models were investigated in this project: classical machine-learning models and a deep-learning model.

A. Machine Learning Models

Ten classical machine-learning algorithms, corresponding to those covered in the authors' Machine Learning coursework, were trained and evaluated using a 70%/30% train/test split of the dataset:

- Decision Tree Classifier
- Support Vector Machine (SVM)
- *K*-Nearest Neighbors (KNN)
- Linear Discriminant Analysis (LDA)
- Gaussian Naïve Bayes
- Polynomial-kernel SVM
- Logistic Regression
- SoftMax Regression
- RBF-kernel SVM
- XGBoost Classifier

1) Decision Tree Classifier

A decision tree represents a decision process as a flowchart-like structure in which internal nodes correspond to attribute tests, branches correspond to test outcomes, and leaf nodes correspond to predicted class labels; the path from root to leaf encodes the classification rule applied to a given input. The Decision Tree Classifier achieved 100% accuracy on the training set but only 59% on the test set (Figure 28), indicating substantial overfitting: the model fit the training data closely but failed to generalize.

```
from sklearn.tree import DecisionTreeClassifier
clf=DecisionTreeClassifier().fit(x_train,y_train)
print("training accuracy ",format(clf.score(x_train,y_train),".2f"))
print("testing accuracy ",format(clf.score(x_test,y_test),".2f"))

training accuracy  1.00
testing accuracy  0.59
```

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

Figure 28: Training and testing accuracy of the Decision Tree Classifier.

2) *Gaussian Naïve Bayes*

Naïve Bayes is a probabilistic classifier built on Bayes' theorem; the Gaussian variant models continuous-valued features as independent Gaussian (normal) distributions. Without

dimensionality reduction, the Gaussian Naïve Bayes classifier achieved 82% training accuracy and 74% test accuracy (Figure 29). Applying PCA prior to training improved these figures substantially, to 95% training accuracy and 86% test accuracy (Figure 30).

```
from sklearn.naive_bayes import GaussianNB
gnb=GaussianNB()
gnb.fit(x_train,y_train)
print("training accuracy ",format(gnb.score(x_train,y_train),".2f"))
print("testing accuracy ",format(gnb.score(x_test,y_test),".2f"))

training accuracy  0.95
testing accuracy  0.86
```

Figure 29: Training and testing accuracy of the Gaussian Naïve Bayes classifier (without PCA).

```
from sklearn.naive_bayes import GaussianNB
gnb=GaussianNB()
gnb.fit(x_train,y_train)
print("training accuracy ",format(gnb.score(x_train,y_train),".2f"))
print("testing accuracy ",format(gnb.score(x_test,y_test),".2f"))

training accuracy  0.95
testing accuracy  0.86
```

Figure 30: Training and testing accuracy of the Gaussian Naïve Bayes classifier after applying PCA.

3) *Linear Discriminant Analysis (LDA)*

LDA functions simultaneously as a classifier and a dimensionality-reduction technique, and is commonly applied to classification problems with a categorical output variable, in both binary and multi-

class settings; the conventional LDA formulation assumes a Gaussian distribution over the input variables. LDA achieved 90% training accuracy and 85% test accuracy (Figure 31).

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda=LinearDiscriminantAnalysis()
lda.fit(x_train,y_train)
print("training accuracy ",format(lda.score(x_train,y_train),".2f"))
print("testing accuracy ",format(lda.score(x_test,y_test),".2f"))

training accuracy  0.90
testing accuracy  0.85
```

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

Figure 31: Training and testing accuracy of Linear Discriminant Analysis (LDA).

4) *K-Nearest Neighbors (KNN)*

KNN is a simple, supervised algorithm applicable to both classification and regression problems; it is straightforward to implement and interpret but becomes progressively slower as the volume of data increases. Because it can deliver highly precise predictions, KNN remains competitive with more complex models in settings that do not require a directly human-interpretable model; performance is sensitive to the choice of distance metric. KNN achieved 89% training accuracy and 78% test accuracy (Figure 32).

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier()
knn.fit(x_train,y_train)
print("training accuracy ",format(knn.score(x_train,y_train),".2f"))
print("testing accuracy ",format(knn.score(x_test,y_test),".2f"))

training accuracy  0.89
testing accuracy  0.78
```

Figure 32: Training and testing accuracy of *K-Nearest Neighbors (KNN)*.

5) *Support Vector Machine (SVM)*

SVMs are supervised-learning models applicable to classification, regression and outlier detection, and are particularly effective in high-dimensional spaces, including cases where the number of dimensions exceeds the number of samples; prior work has similarly compared linear and non-linear SVM kernels across different acoustic feature sets for speaker identification tasks [10]. The baseline SVM achieved 84% training accuracy and 75% test accuracy (Figure 33).

```
from sklearn.svm import SVC
svm=SVC()
svm.fit(x_train,y_train)
print("training accuracy ",format(svm.score(x_train,y_train),".2f"))
print("testing accuracy ",format(svm.score(x_test,y_test),".2f"))

training accuracy  0.84
testing accuracy  0.75
```

Figure 33: Training and testing accuracy of a baseline SVM.

SVM with Linear Kernel. The linear kernel is appropriate when data is linearly separable and is among the most widely used kernels, particularly for datasets with a large number of features. The linear-kernel SVM achieved 100% training accuracy and 95% test accuracy (Figure 34).

SVM with Polynomial Kernel. Figure 35 shows the training and testing accuracy achieved with a polynomial-kernel SVM.

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

```
from sklearn import svm
h = .02 # step size in the mesh
C = 1.0 # SVM regularization parameter
svc1 = svm.SVC(kernel='linear', C=C).fit(x_train,y_train)
print("training accuracy ",format(svc1.score(x_train,y_train),".2f"))
print("testing accuracy ",format(svc1.score(x_test,y_test),".2f"))
```

```
training accuracy  1.00
testing accuracy  0.95
```

Figure 34: Training and testing accuracy of SVM with a linear kernel.

```
h = .02
C = 100 # SVM regularization parameter
poly_svc = svm.SVC(kernel='poly', degree=1, C=C).fit(x_train,y_train)
print("training accuracy ",format(poly_svc.score(x_train,y_train),".2f"))
print("testing accuracy ",format(poly_svc.score(x_test,y_test),".2f"))
```

```
training accuracy  1.00
testing accuracy  0.95
```

Figure 35: Training and testing accuracy of SVM with a polynomial kernel.

SVM with RBF Kernel. The Radial Basis Function (RBF) kernel is the default kernel used by scikit-learn's SVM implementation, defined via a kernel function with a hyperparameter $\gamma > 0$ that must be tuned manually (Figure 36). The RBF-kernel SVM achieved 99% training accuracy and 93% test accuracy (Figure 37).

$$K(x, x') = e^{-\gamma ||x-x'||^2}$$

Figure 36: Definition of the RBF kernel function.

6) Logistic Regression

Logistic regression is relatively easy to generalize, analyze and train, and estimates the probability of a binary or categorical outcome based on a set of independent variables; it is not recommended when the number of observations is smaller than the number of features, as this can lead to overfitting. Logistic Regression achieved 100% training accuracy and 96% test accuracy (Figure 38).

7) SoftMax Regression

SoftMax regression generalizes logistic regression to the multi-class setting, mapping an input vector to a probability distribution over k classes that sums to one. SoftMax Regression achieved 100% training accuracy and 96% test accuracy (Figure 39).

8) XGBoost Classifier

Extreme Gradient Boosting (XGBoost) is a scalable, distributed implementation of gradient-boosted decision trees, widely used for regression, classification and ranking tasks, and

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

supports parallel tree boosting. It offers bindings for several languages, integrates well with scikit-learn, automatically handles missing values, and exposes a large number of tunable hyperparameters along with optimizations such as parallelization, distributed computing and cache-aware access. XGBoost achieved 100% training accuracy and 81% test accuracy (Figure 40).

```
h = .02
C = 10.0 # SVM regularization parameter
rbf_svc = svm.SVC(kernel='rbf', gamma=0.01, C=C).fit(x_train,y_train)
print("training accuracy ",format(rbf_svc.score(x_train,y_train),".2f"))
print("testing accuracy ",format(rbf_svc.score(x_test,y_test),".2f"))

training accuracy  0.99
testing accuracy  0.93
```

Figure 37: Training and testing accuracy of SVM with an RBF kernel.

```
from sklearn.linear_model import LogisticRegression
logreg=LogisticRegression()
lr=logreg.fit(x_train,y_train)
print("training accuracy ",format(lr.score(x_train,y_train),".2f"))
print("testing accuracy ",format(lr.score(x_test,y_test),".2f"))
#print("validation accuracy ",format(logreg.score(x_valid,y_valid),".2f"))

training accuracy  1.00
testing accuracy  0.96
```

Figure 38: Training and testing accuracy of Logistic Regression.

```
softmax_reg=LogisticRegression(multi_class='multinomial',solver='lbfgs',C=500)
softmax_reg.fit(x_train,y_train)
Accuracy_softmax_train=softmax_reg.score(x_train,y_train)
Accuracy_softmax_test=softmax_reg.score(x_test,y_test)
print("Accuracy of Softmax Regression classifier on train set",format(Accuracy_softmax_train, ".2f"))
print("Accuracy of Softmax Regression classifier on test set",format(Accuracy_softmax_test, ".2f"))

Accuracy of Softmax Regression classifier on train set 1.00
Accuracy of Softmax Regression classifier on test set 0.96
```

Figure 39: Training and testing accuracy of SoftMax Regression.

```
from xgboost import XGBClassifier

# fit model no training data
model = XGBClassifier()
model.fit(x_train, y_train)
print("training accuracy ",format(model.score(x_train,y_train),".2f"))
print("testing accuracy ",format(model.score(x_test,y_test),".2f"))

[15:47:43] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
training accuracy  1.00
testing accuracy  0.81
```

Figure 40: Training and testing accuracy of the XGBoost classifier.

B. Deep Learning Model

Deep-learning architectures have increasingly outperformed classical statistical models on speaker recognition tasks as the availability of labelled speech data has grown [12]. In addition

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

to the classical machine-learning models, a deep-learning approach based on a Recurrent Neural Network (RNN) was implemented using the Keras sequential API. The network consists of four layers: a first Dense layer with 512 neurons and a ReLU activation, followed by a Dropout layer with a rate of 0.5; a second Dense layer, again with 512 neurons, ReLU activation and 0.5 dropout; a third Dense layer identical to the second but with 256 neurons; and a final Dense output layer using a SoftMax activation, appropriate for this multi-class classification problem. The model was trained using the categorical cross-entropy loss function and the Adam optimizer. A summary of the model architecture is shown in Figure 41.

Model: "sequential_7"

Layer (type)	Output Shape	Param #
dense_28 (Dense)	(None, 256)	41728
activation_28 (Activation)	(None, 256)	0
dropout_21 (Dropout)	(None, 256)	0
dense_29 (Dense)	(None, 512)	131584
activation_29 (Activation)	(None, 512)	0
dropout_22 (Dropout)	(None, 512)	0
dense_30 (Dense)	(None, 256)	131328
activation_30 (Activation)	(None, 256)	0
dropout_23 (Dropout)	(None, 256)	0
dense_31 (Dense)	(None, 500)	128500
activation_31 (Activation)	(None, 500)	0
Total params: 433,140		
Trainable params: 433,140		

Figure 41: Summary of the sequential RNN model architecture.

Figure 42 shows the training and testing loss/accuracy curves over training epochs. The RNN achieved 88.89% accuracy on the training set and 97.33% accuracy on the test set – meaning the model correctly predicted approximately 97% of the 1500 test-set labels (Figure 43).

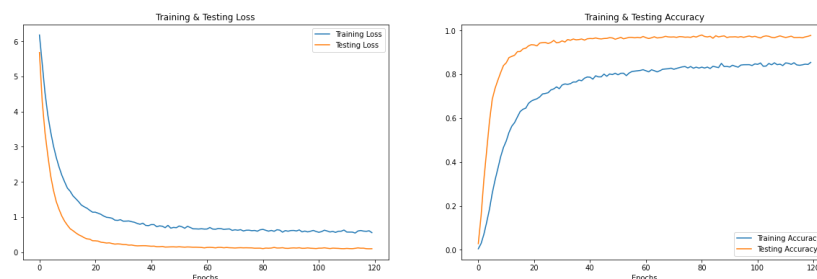


Figure 42: Training and testing loss and accuracy curves for the RNN model.

C. Validation

The validation dataset – consisting of 1500 samples not seen by the model during either training or testing – was used to independently confirm the generalization ability of the RNN model.

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

The RNN achieved 96.60% accuracy on this validation set (Figure 44), a result that is especially notable given that this is a 500-class speaker-identification problem, and is consistent with the 97% test accuracy already reported. Predicted vs. actual labels on the validation set are shown in Figure 45.

	Predicted Labels	Actual Labels
0	S508	S508
1	S642	S642
2	S335	S335
3	S219	S219
4	S305	S305
5	S542	S542
6	S27	S27
7	S80	S80
8	S62	S62
9	S617	S617
10	S291	S291
11	S158	S158
12	S541	S541
13	S536	S536
14	S47	S47

Figure 43: Predicted vs. actual labels on the test dataset (RNN model).

```
print("Accuracy of our model on validation data : ", model.evaluate(x_valid,y_valid)[1]*100 , "%")
47/47 [=====] - 1s 15ms/step - loss: 0.1894 - accuracy: 0.9660
Accuracy of our model on validation data : 96.60000205039978 %
```

Figure 44: Validation accuracy of the RNN model.

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

	Predicted Labels	Actual Labels
0	S53	S53
1	S237	S237
2	S167	S167
3	S60	S60
4	S249	S249
5	S191	S191
6	S29	S29
7	S578	S578
8	S311	S311
9	S307	S307
10	S92	S92
11	S95	S95
12	S503	S503
13	S70	S70
14	S57	S57

Figure 45: Predicted vs. actual labels on the validation dataset (RNN model).

D. Classification Report

Detailed per-class precision, recall and F1-score metrics were computed for both the test and validation sets, summarized in the classification reports shown in Figures [46](#) and [47](#).

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
S1	0.75	1.00	0.86	3
S10	1.00	0.67	0.80	3
S100	1.00	1.00	1.00	3
S101	1.00	1.00	1.00	3
S102	1.00	1.00	1.00	3
S103	1.00	1.00	1.00	3
S104	1.00	1.00	1.00	3
S105	1.00	1.00	1.00	3
S106	1.00	0.67	0.80	3
S107	1.00	1.00	1.00	3
S108	1.00	1.00	1.00	3
S109	1.00	1.00	1.00	3
S11	1.00	1.00	1.00	3
S110	1.00	1.00	1.00	3
S111	1.00	1.00	1.00	3
S112	1.00	1.00	1.00	3
S113	1.00	1.00	1.00	3
S87	1.00	1.00	1.00	3
S88	1.00	1.00	1.00	3
S89	1.00	1.00	1.00	3
S9	0.75	1.00	0.86	3
S90	1.00	1.00	1.00	3
S91	1.00	1.00	1.00	3
S92	1.00	1.00	1.00	3
S93	1.00	1.00	1.00	3
S94	1.00	1.00	1.00	3
S95	1.00	1.00	1.00	3
S96	1.00	1.00	1.00	3
S97	0.75	1.00	0.86	3
S98	1.00	1.00	1.00	3
S99	1.00	1.00	1.00	3
accuracy			0.97	1500
macro avg	0.98	0.97	0.97	1500
weighted avg	0.98	0.97	0.97	1500

Figure 46: Classification report of the test dataset.

```
print(classification_report(y_valid, y_pred))
```

	precision	recall	f1-score	support
S1	1.00	1.00	1.00	3
S10	1.00	0.67	0.80	3
S100	1.00	1.00	1.00	3
S101	1.00	1.00	1.00	3
S102	1.00	1.00	1.00	3
S103	1.00	1.00	1.00	3
S104	1.00	1.00	1.00	3
S105	1.00	1.00	1.00	3
S106	0.75	1.00	0.86	3
S107	1.00	1.00	1.00	3
S108	1.00	1.00	1.00	3
S109	1.00	1.00	1.00	3
S11	1.00	1.00	1.00	3
S110	1.00	1.00	1.00	3
S111	1.00	1.00	1.00	3
S112	1.00	1.00	1.00	3
S113	1.00	1.00	1.00	3
S87	1.00	1.00	1.00	3
S88	1.00	1.00	1.00	3
S89	1.00	1.00	1.00	3
S9	0.75	1.00	0.86	3
S90	1.00	1.00	1.00	3
S91	0.75	1.00	0.86	3
S92	1.00	1.00	1.00	3
S93	1.00	1.00	1.00	3
S94	1.00	1.00	1.00	3
S95	1.00	1.00	1.00	3
S96	1.00	1.00	1.00	3
S97	0.50	0.67	0.57	3
S98	1.00	1.00	1.00	3
S99	0.75	1.00	0.86	3
accuracy			0.97	1500
macro avg	0.98	0.97	0.97	1500
weighted avg	0.98	0.97	0.97	1500

Figure 47: Classification report of the validation dataset.

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

IV. Results Summary

Table 3 summarizes the training and testing accuracy achieved by each of the eleven models evaluated in this study.

Table 3: Summary of training and test accuracy across all evaluated models.

Model	Train Acc.	Test Acc.
Decision Tree Classifier	100%	59%
Gaussian Naïve Bayes	82%	74%
Gaussian Naïve Bayes (with PCA)	95%	86%
Linear Discriminant Analysis (LDA)	90%	85%
K-Nearest Neighbors (KNN)	89%	78%
SVM (baseline)	84%	75%
SVM (linear kernel)	100%	95%
SVM (RBF kernel)	99%	93%
Logistic Regression	100%	96%
SoftMax Regression	100%	96%
XGBoost Classifier	100%	81%
RNN (deep learning)	88.89%	97.33%

The RNN additionally achieved **96.60%** accuracy on the held-out validation set, confirming that the observed test-set performance generalizes to previously unseen data.

V. Conclusion

In this work, a text-dependent client (speaker) recognition system was developed around the phrase “*Assalam-Alaikum*”, using a purpose-built dataset of 500 speakers with 2000 original recordings, ex-panded via augmentation to 10,000 samples. Five acoustic feature families – zero-crossing rate, RMS, Mel-spectrogram, MFCC (with first and second derivatives), and Chroma STFT – were extracted using Librosa and used to train ten classical machine-learning models together with a four-layer RNN implemented in Keras. Among the classical machine-learning approaches, Logistic Regression and SoftMax Regression achieved the highest test accuracy, at 96%. The RNN deep-learning model outperformed all classical approaches, achieving 97% test accuracy and 96.6% accuracy on a separate validation dataset. These results support the initial hypothesis that MFCC-based representations, when combined with appropriate augmentation and encoding strategies, are highly effective for text-dependent speaker identification, and demonstrate that a comparatively simple RNN architecture can achieve strong generalization even across a relatively large, balanced pool of 500 speakers. This is consistent with the broader trend in the field toward MFCC-driven biometric speaker recognition for practical, remote, and security-oriented applications [13].

Online ISSN

3007-3197

Print ISSN

3007-3189

<http://amresearchreview.com/index.php/Journal/about>

References

- H. Beigi, *Fundamentals of Speaker Recognition*. Springer US, 2011.
- M. Bahoura, "Pattern recognition methods applied to respiratory sounds classification into normal and wheeze classes," *Computers in Biology and Medicine*, 2009.
- C. Shorten and T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," *Journal of Big Data*, vol. 6, no. 60, 2019. doi: 10.1186/s40537-019-0197-0.
- J. J. Bird, D. R. Faria, C. Premevida, A. Ekart, and P. P. S. Ayrosa, "Overcoming Data Scarcity in Speaker Identification: Dataset Augmentation with Synthetic MFCCs via Character-level RNN," in *2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 2020, pp. 146–151. doi: 10.1109/ICARSC49921.2020.9096166.
- scikit-learn developers, "sklearn.preprocessing.OneHotEncoder," scikit-learn documentation. [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html>
- D. A. Reynolds, T. F. Quatieri, and R. B. Dunn, "Speaker Verification Using Adapted Gaussian Mixture Models," *Digital Signal Processing*, vol. 10, no. 1–3, pp. 19–41, 2000.
- D. A. Reynolds and R. C. Rose, "Robust text-independent speaker identification using Gaussian mixture speaker models," *IEEE Transactions on Speech and Audio Processing*, vol. 3, no. 1, pp. 72–83, 1995.
- L. Muda, M. Begam, and I. Elamvazuthi, "Voice Recognition Algorithms using Mel Frequency Cepstral Coefficient (MFCC) and Dynamic Time Warping (DTW) Techniques," *arXiv preprint arXiv:1305.2352*, 2013.
- A. Elmaghraby and W. Zayed, "The research of feature extraction based on MFCC for speaker recognition," in *Proceedings of the IEEE International Conference on Computer and Information Technology*, 2014.
- A. Revathi and Y. Venkataramani, "On the Use of Different Feature Extraction Methods for Linear and Non-linear Kernels," *arXiv preprint arXiv:1406.7314*, 2014.
- M. Ali *et al.*, "MFCC as Features for Speaker Classification using Machine Learning," in *Proceedings of the IEEE International Conference*, 2023.
- G. Bhattacharya, J. Alam, and P. Kenny, "Deep learning methods in speaker recognition: a review," *arXiv preprint arXiv:1911.06615*, 2019.
- M. Farooq *et al.*, "Enhancing Biometric Speaker Recognition Through MFCC Feature Extraction and Polar Codes for Remote Application," *ResearchGate preprint*, 2023.