Leveraging Mel-Frequency Cepstral Coefficients to Identify Machine Generated Voice Audio Through K-Means and Naive Bayes Classification

Syracuse University
College of Engineering and Computer Science
Daniel Shannon (djshanno@syr.edu)
March 20th, 2024
https://github.com/radioxeth/voice-attack/

Abstract

Here we explore the features that comprise human recorded audio and machine generated audio to train and compare binary k-means clustering and Naive Bayes classification systems. The goal is to correctly identify if the sample voice is human or machine generated. The machine generated audio samples are created using the text-to-speech model Tacotron2 *(Shen et al.)*. The recorded samples are gathered from the Common Voice repository ("Common Voice"). We process the audio waveforms through resampling, normalization, and Voice Activity Detection. Each audio sample is transformed to create a log power Mel spectrogram from which we can calculate the Mel Frequency Cepstral Coefficients (MFCC) *(Hasan et al.)*. We then pass the MFCC through a binary k-means clustering classifier and a Naive Bayes classifier to find the accuracy, construct a confusion matrix, and determine the best model and number of MFCCs to use for identifying human voices.

The Naive Bayes approach outperforms the binary k-means clustering classification system for every number of MFCCs used in the relative model. The Naive Bayes Model (NBM) with 16 MFCC principal components achieved a mean accuracy of 92.63% with a standard deviation of 2.71%. The k-means model (KMM) trained on 27 MFCC features achieved a mean accuracy of 78.32% with a standard deviation of 2.56%. The models were trained and tested on a dataset of 2,410 text-to-speech samples and 4,776 recorded samples from Mozilla Common Voice *("Common Voice")*.

Introduction

Text-to-speech (TTS) models can serve a variety of purposes, including accessibility, learning tools, and delivering messages. While TTS can be incredibly helpful, it is also possible to misuse these models in adversarial ways, such as robocalls and scams. Biometric voice authentication systems rely on the unique features of a person's voice to identify if someone is speaking and who that speaker is. We aim to extract these features and use them to determine if a voice audio sample is human generated or machine generated.

Data Generation

We start with data generation, which involves gathering a subset of data from the Common Voice corpus ("Common Voice"), and TTS generated recordings. We use the Common Voice transcripts to generate the TTS recordings through Tacotraon2 (Yang and Hira). In our experiments, we used the cv-valid-dev subset of Common Voice - 4,076 samples and generated 2,410 TTS samples, totalling 6,486 samples.

TTS models can generate wildly incorrect text, so we will use the speech-to-text (STT) model Whisper Tiny to compose the TTS back to text. By measuring the Levenshtein distance between the TTS and STT transcripts, we can examine the general quality of the TTS generated samples. The whisper-tiny model was trained using only voice recorded data (Radford et al.).

Waveform Processing

We must process the sample waveforms to ensure that we are accounting for variations in sample rate, length, amplitude, and noise. First we derive the waveform and resample to match the sample rate of 16,000 Hz, the same sample rate used in Whisper Tiny. We also normalize the waveform's amplitude to the maximum amplitude of all samples in the dataset (0.9999). After resampling and normalization, we use a Voice Activity Detection (VAD) algorithm to detect voice, trim silence, and remove noise.

Mel Frequency Cepstral Coefficients

Mel Frequency Cepstral Coefficients (MFCC) are the features we use to train and test our models. We transform the time domain waveforms into frequency domain log-power Mel spectrograms through a Discrete Fourier Transform (DFT). The Mel-frequencies are binned against time, and the logarithmic scale represents audio similar to how the human ear perceives sound. We can extract MFCCs from the Mel-frequency bins through a Discrete Cosine Transform (DCT).

Before training the model, we pad and reduce the dimensions of the MFCCs. For binary k-means clustering, we transform, reshape, and flatten the MFCCs to extract the first two features of each sample's MFCCs. With Naive Bayes, we employ Principal Component Analysis (PCA) to reduce the dimensionality of the sample to the number of MFCCs extracted.

Training and Results

We train and test each model 10 times for each MFCC ranging from 1 to 40. Every time we train a model, the data is shuffled and then split into training and testing data sets. The results are saved for analysis to find the MFCCs which yield the highest mean and maximum accuracy.

The Naive Bayes classification model generally outperforms binary k-means clustering by approximately 12%. The most accurate model is Naive Bayes classification with 15 MFCCs at 95.61% accuracy. The highest *mean* accuracy for Naive Bayes classification was achieved with 16 MFCCs at 92.63% accuracy with a standard deviation of 2.71%. Binary k-means clustering performs best with 27 MFCCs yielding 83.91% max accuracy and a 78.32% mean accuracy with a standard deviation of 2.56%.

Previous Work

Mel Frequency Cepstral Coefficients

MFCCs have many applications, including those in industrial analysis, medical analysis and acoustic analysis (Abdul and Al-Talabani). MFCC's are used for speech analysis because the features they extract are features that contain human ear information. These features are representative of the features as humans perceive them. This is the result of starting with the Mel spectrogram, which scales the amplitude of audio to match how a human hears.

Figure 1 shows how MFCC features are extracted from waveforms. They are common features used in voice processing. Pre-emphasis improves the signal to noise ratio by accounting for the high frequencies lost in recording. Framing and windowing reduces the variation of the signal over time into bins. The DFT transforms the time domain waveform into a frequency domain Mel-Frequency filter bank. Applying a logarithm to the filter bank emphasizes the features similar to how a human ear hears. The DCT extracts the MFCCs features.

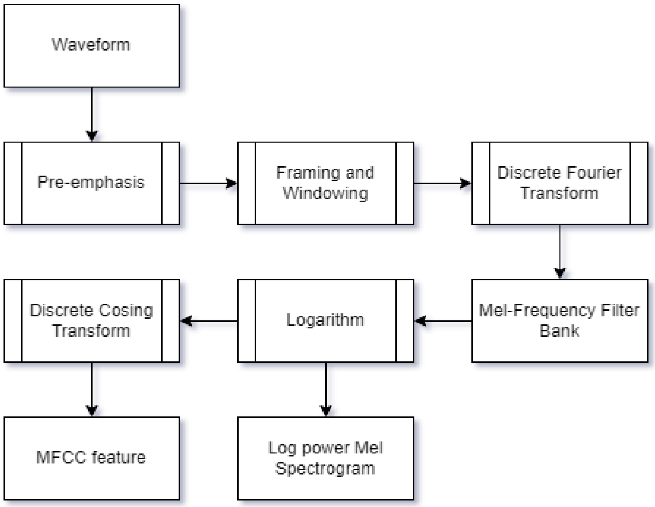
Waveform
Pre-emphasis
Framing and Windowing
Discrete Fourier Transform
Mel-Frequency Filter Bank
Logarithm
Discrete Cosing Transform
MFCC feature
Log power Mel Spectrogram

Figure 1. The process of extracting MFCC features from the time-domain waveform of an audio sample. (Abdul and Al-Talabani)

Tacotron2 Text-to-Speech

Tacotron2 is an off the shelf TTS model available in the TorchAudio python library (Yang et al.). The three main tenets of Tacotron2 are that it is GPU compatible, has automatic differentiability, and is production ready. This means that it is simple to compute TTS samples on a GPU and integrate the model seamlessly into a machine learning pipeline.

Whisper Tiny Speech-to-Text Model

Whisper is a large-scale unsupervised speech-to-text (STT). The Whisper model was trained on 680,000 hours of unsupervised human audio recordings paired with transcripts and has 5 model sizes ranging from tiny (39M parameters) to large (1550M parameters). Because machine generated transcripts have been shown to decrease performance (Abdul and Al-Talabani), Whisper used heuristics in the data grooming process to identify and remove machine generated transcripts.

The Whisper model was trained on data that is processed into 80-channel log-magnitude Mel spectrograms that has been resampled to 16,000 Hz. The mel spectrogram is computed on a 25 millisecond window with a stride of 10 milliseconds. This guides our own processing decisions as we will attempt to match the processing parameters used to generate the Mel spectrograms.

Classification Techniques

Previous studies have shown that we can use a k-means clustering classifier to identify speakers using the MFCCs and Vector Quantization (Dirman et al). Dirman et al. define the performance of the system as:

$$performance = \frac{number\ of\ speaker\ identified}{total\ number\ of\ speaker\ tested} \times 100$$

This approach achieves 100% accuracy using MFCC features with 12 coefficients in the 5th and 6th dimension.

The MFCCs are especially useful for classifying audio features such as emotion. Emotion in speech can be classified by applying a Naive Bayes classifier as demonstrated by Bhakre and Bang. Bhakre and Bang extracted 14 pitch, energy, and Zero Crossing Rate MFCC features. These features were used to identify 4 distinct emotions - anger, happy, sad, and neutral. They achieved accuracies of 81%, 78%, 76%, and 77% for anger, happy, sad, and neutral, respectively.

Experiment Design

We propose using a binary k-means clustering classifier and a Naive Bayes classifier to distinguish between recorded human speech recordings and TTS derived samples. We aim to determine which classifier achieves a higher accuracy and the number of MFCC features used to do so. The process includes data gathering and generation, waveform processing, MFCC feature extraction, dimension reduction, model training, testing, and analysis as seen in **Figure 2**. All processing is completed on a Lenovo ThinkPad with 12 Intel Core i7, 2.7 GHz CPUs and 32GB of memory. The machine also has a NVIDIA GeForce GTX 1650 Ti graphics card for GPU computations.

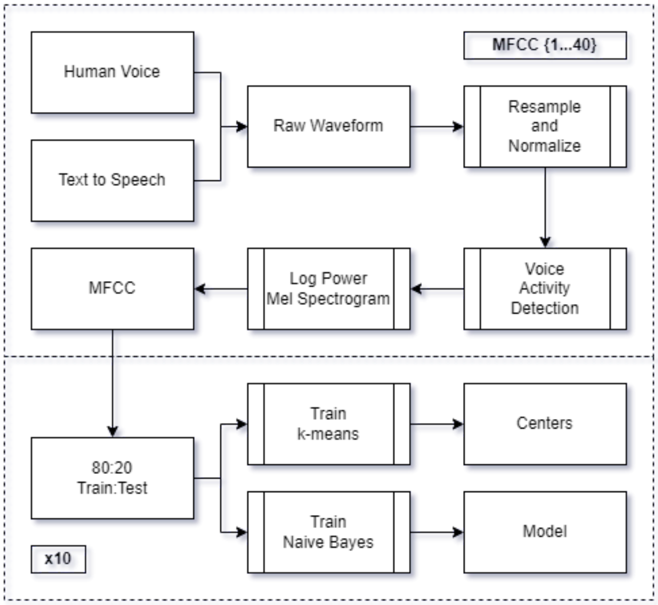

Figure 2. Experiment design for feature extraction of 40 sets of MFCCs being trained and tested 10 times each.

Data Gathering and Generation

We gathered 4,076 recorded and 2,410 machine generated samples. The recorded samples are a subset of the Mozilla Common Voice cv-valid-dev dataset available on Kaggle ("Common Voice"). The generated samples are the output of the TTS model Tacotron2.

To generate the samples with Tacotron2, we deduplicate the Common Voice transcripts to 2,410 unique transcripts. These are the transcripts used to supply the TTS model. This way the TTS model uses the same collection of phrases as the recorded dataset, eliminating the chance that the generated samples are classified due to the phrase being spoken. Tacotron2 and Whisper Tiny are both capable of computing on the GPU. It took 48 hours to generate the 2,410 TTS-STT samples.

Each machine generated sample is then transcribed by the Whisper Tiny STT model. The original transcription is compared to the Whisper transcription by measuring the Levenshtein distance. We can see that the TTS samples are generally close to the original transcript, within 11 edits of each other by 2 standard deviations.

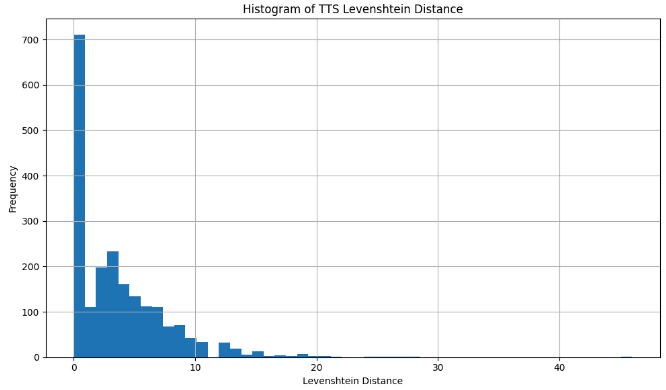

Figure 3. Histogram of the measured TTS-STT Levenshtein distances. A majority of the TTS-STT transcript pairs are within 11 edits of each other.

Waveform Processing

To prepare samples for feature extraction and classification we need to resample, normalize, and filter the waveform. The audio is loaded as a waveform, or relative amplitude vs time. We resample every waveform to 16,000 Hz, or frames per second. Next we normalize the waveform's amplitude to the maximum amplitude from all of the sample, which is 0.9999 for our dataset. Finally we pass the waveform through a Voice Activity Detection (VAD) algorithm provided by TorchAudio (Yang et al.). VAD attempts to detect speech by trimming silence and filtering noise using Cepstral Coefficients.

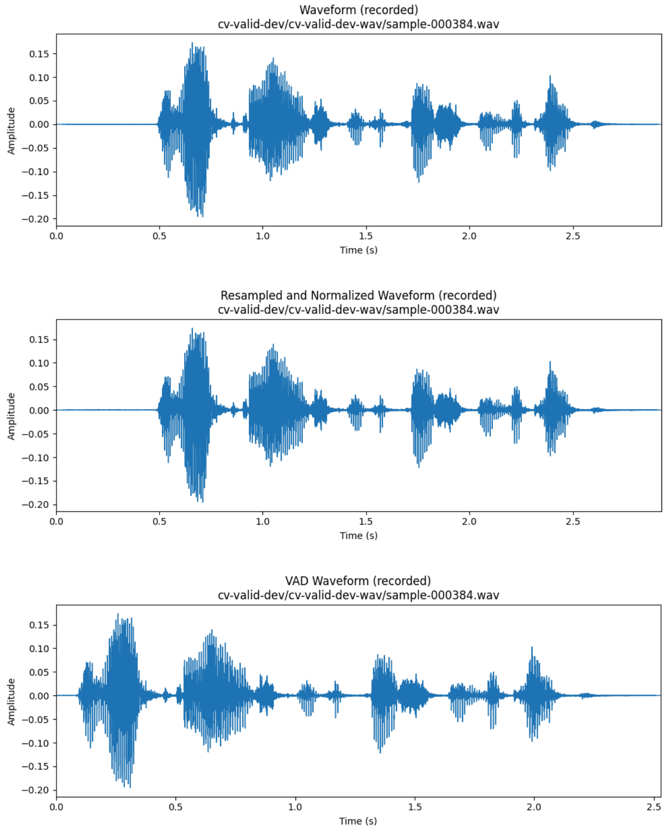

Figure 4. Raw waveform (top), resampled and normalized waveform (middle), and VAD filtered waveform (bottom) for the recorded audio sample "sample-000384.wav".

Feature Extraction

After processing the waveforms, we can begin to extract the MFCC features. We transform each time-domain waveform into a frequency-domain Mel spectrogram through a DFT. Taking the logarithm of the Mel spectrogram gives us a log-power Mel spectrogram of the Mel-Frequency bins. Each bin reduces the variability in the amplitude of the data in that frequency. We set the window length and hop length to 2,048 frames and 512 frames, respectively, for each number of MFCCs we extract. The window length is the size in frames of the window, and the hop length is the number of frames the window moves to smooth each bin. From the log-power Mel spectrograms, we can perform a DCT to extract the MFCC features of the waveform. The log-power Mel spectrogram and MFCC spectrogram of sample-000348.wav can be seen in **Figures 5 and 6**, respectively.

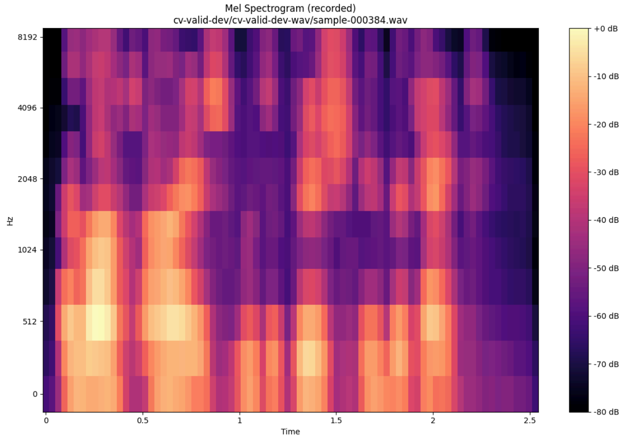

Figure 5. Log-power Mel spectrogram with 15 Mel-Frequency bins of recorded audio "sample-000348.wav".

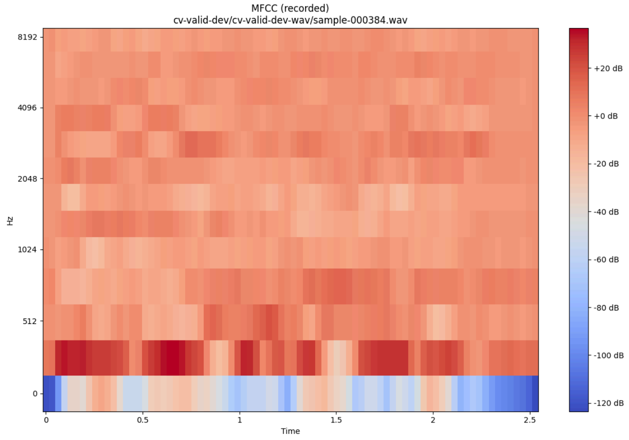

Figure 6. 15 MFCC features extracted from recorded audio "sample-000348.wav".

To identify the optimum number of MFCCs to use in the classification of machine generated voices, we extract increasing numbers of MFCCs, up to 40 coefficients. We end up with 40 different MFCC feature sets for each of the 6,486 audio samples.

Dimension Reduction

For binary k-means clustering we need two dimensions, but the MFCCs are 3 dimensional - time, frequency bin, and power. To reduce the dimensions of the MFCC features for binary k-means clustering, we pad, stack, transpose, reshape and slice the MFCC. Padding the MFCCs ensures each sample is the same size. (In future studies, we should pad the waveform during processing instead of the MFCCs.)

When the MFCCs for every sample are padded and stacked, we are left with a 3D array. Transposing and reshaping the data renders the time series data into a fixed-size feature vector. After transposing and reshaping the data, we are left with a flattened 2D array where each row is a sample, and each column is the flattened time series of each MFCC. We use only the first two features in binary k-means clustering (**Figure 6**).

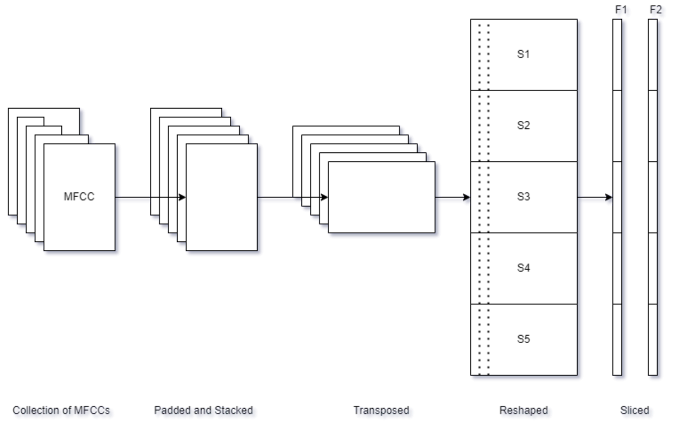

Figure 6. The dimension reduction process for binary k-means clustering samples. We end up with two features for each sample.

To reduce the dimensionality of the data for Naive Bayes classification we employ PCA against the MFCC feature sets. First the MFCCs are padded and stacked into a 3D array. Each principal component (PC) extracted from PCA correlates to the number of MFCCs used in the classification. If a sample has n MFCCs, then PCA will reduce to n PCs as seen in **Figure 7**. There is an exception for the case of 1 MFCC. We cannot perform Naive Bayesian classification on a single feature using the Scikit-learn NaiveBayes python library (Pedregosa), so the 1 MFCC feature set extracts 2 PCs.

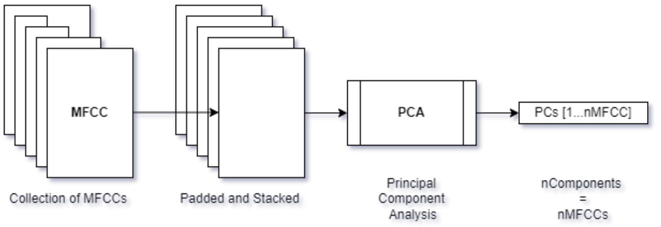

Figure 7. Principal Component Analysis of n MFCCs results in n Principal Components.

Training and Testing

We train and test each model 10 times for each number of MFCCs extracted, totaling 400 model runs for each of the binary k-means clustering and Naive Bayes classifiers (800 total model runs). Every time a model is trained, the samples are shuffled and split 80:20 into training and testing sets. Training the model several times allows us to determine the mean performance metrics and their standard deviations.

The binary k-means clustering algorithm used is imported from the python library Scikit-learn (Pedregosa et al). Training the k-means model identifies two centers. The centers are the model, and whichever center is closer to the sample is the prediction. We guess which center belongs to which class. During training, we noticed that the accuracy would either be around 80% or 20% for the binary k-means clustering model. If the accuracy on a trained set is less than 50% during testing, we swap the center's class assignment.

For training the Naive Bayes model, every sample is stacked and its dimensions are reduced to the number of MFCCs in the training set. The probabilistic model is then used to test and the predicted classes are output for analysis.

Analysis

During training and testing, we classify each sample as TTS (-1) or RECORDED (1) and specify RECORDED as the base class. After each test, we have a list of known and predicted values. These values are then used to derive the performance metrics of accuracy, precision, recall, and specificity. Since we run each model 10 times, we can examine the mean performance metrics and understand the consistency of each model.

Results

The Naive Bayes classifier outperforms the binary k-means clustering classifier in all performance metrics except for recall. The maximum mean accuracy of the Naive Bayes classifier is 92.63% with a standard deviation of 2.71% achieved with 16 MFCCs. The maximum mean accuracy of the binary k-means clustering classifier is 78.32% with a standard deviation of 2.56% achieved with 27 MFCCs. The overall maximum accuracy achieved by Naive Bayes is 95.61% with 15 MFCCs. The overall maximum accuracy achieved by binary k-means clustering is 83.91% with 27 MFCCs, respectively. We can examine the confusion matrices of the best performing classifiers in **Figures 8 and 9**.

K-Means MFCC: 27 Round: 5	**Predicted Recorded**	**Predicted Generated**
Actual Recorded	477	6
Actual Generated	203	613

Figure 8. The confusion matrix for the best performing binary k-means clustering classification.

Naive Bayes MFCC: 15 Round: 3	**Predicted Recorded**	**Predicted Generated**
Actual Recorded	463	19
Actual Generated	38	778

Figure 9. The confusion matrix for the best performing Naive Bayes classification.

The best performing binary k-means clustering classification was achieved in the 5th round of 27 MFCCs. The precision, recall, and specificity are 70.15%, 98.76%, and 75.12%, respectively. Only the recall performance metric is higher than the performance metrics of the Naive Bayes classifier. The precision, recall, and specificity of the best performing Naive Bayes classifier are 92.42%, 96.06%, and 95.34%, respectively.

The number of MFFCs used in the binary k-means clustering classification appears to have an insignificant effect on the algorithm's performance. The performance metrics are stable across all MFCCs for binary k-means clustering. The Naive Bayes model with 1 or 2 MFCCs has similar accuracy to the binary k-means clustering model. The accuracy of the Naive Bayes classifier improves as the number of MFCCs increases and levels out at 7 MFCCs as seen in **Figure 11**. We can also examine the Receiver Operating Characteristic (ROC) curves of each classifier (**Figure 12**). In the ROC curve chart comparing the most accurate models, we can see that the Naive Bayes has more area under the curve. The Naive Bayes curve also has a better false positive rate to true positive rate ratio at the elbow of the curve.

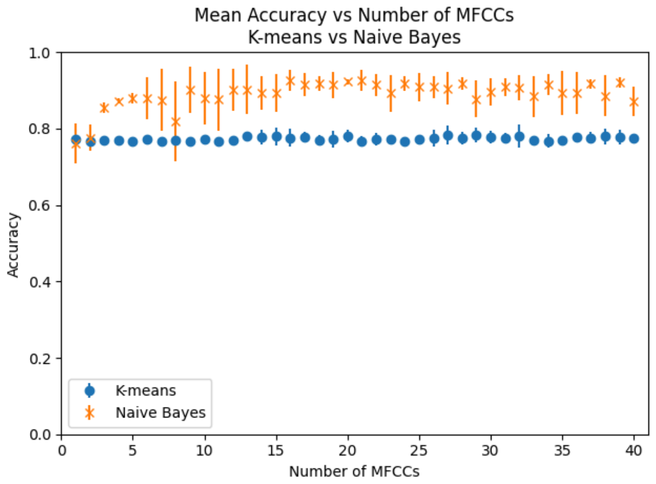

Figure 11. The mean accuracy for each number of MFCCs used to train the model. The Naive Bayes is generally more accurate than the binary k-means clustering classifier.

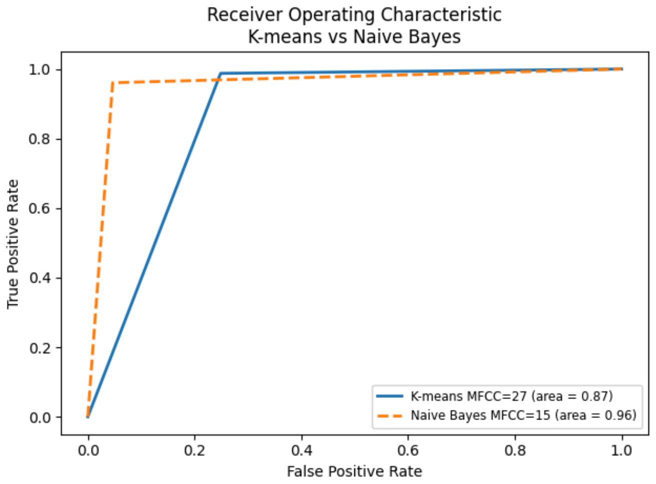

Figure 12. The ROC curves of the best performing binary k-means clustering and Naive Bayes classification models.

Conclusion and Future Work

In conclusion, the Naive Bayes classifier outperforms the binary k-means clustering classifier in all performance metrics except for recall. The Naive Bayes classifier performs best with 15 MFCCs reduced to 15 PCs and is able to achieve a 95.61% accuracy. The binary k-means clustering classifier achieved an 83.91% accuracy with 27 MFCCs.

Future work includes determining other features to use in k-means clustering. We only used the first two features in k-means clustering. Additionally, we can work to generate a more diverse TTS dataset. Each Tacotron2 sample has a similar “voice”. We may be able to improve our model by including samples from more TTS models.

We should also expand this technique to classify deep fake audio. Deep fakes are more of a threat than general TTS audio because deep fakes can be used in sophisticated scams and to spread misinformation. Being able to detect and classify deep fakes will prove necessary as they become more common. The literature shows promising results using neural networks to identify deep fake audio samples (Wang et al).

Naive Bayes is a good starting point for understanding how MFCCs and PCA affect the classification of recorded and generated audio samples. The future of this field is promising and we will see ample opportunities to identify machine generated voice using machine learning and pattern recognition.

Works Cited

Abdul, Z. K., and A. K. Al-Talabani. “Mel Frequency Cepstral Coefficient and its Applications: A Review.” *IEEE Access*, vol. 10, 2022, pp. 122136-122158, doi: 10.1109/ACCESS.2022.3223444.

Bhakre, S K, and A. Bang. “Emotion recognition on the basis of audio signal using Naive Bayes classifier.” *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2016, pp. 2363-2367, doi: 10.1109/ICACCI.2016.7732408.

“Common Voice.” *Kaggle*, Mozilla, https://www.kaggle.com/datasets/mozillaorg/common-voice. Accessed 19 March 2024.

Dirman, Hanafi, et al. “Speaker Identification Using K-means Method Based on Mel Frequency Cepstral Coefficients.” *ProQuest*, vol. 1, no. 1, 2012, pp. 29-28.

Hasan, Md Rashidul, et al. “Speaker Identification Using Mel Frequency Cepstral Coefficients.” *3rd International Conference on Electrical & Computer Engineering*, 2004.

“openai/whisper-tiny · Hugging Face.” *Hugging Face*, 13 September 2023, https://huggingface.co/openai/whisper-tiny. Accessed 5 March 2024.

Pedregosa. “Scikit-learn: Machine Learning in Python.” *JMLR*, vol. 12, 2011, pp. 2825-2830.

Radford, Alec, et al. “[2212.04356] Robust Speech Recognition via Large-Scale Weak Supervision.” *arXiv*, 6 December 2022, https://arxiv.org/abs/2212.04356. Accessed 5 March 2024.

Shannon, Daniel. “Voice Attack.” 2024, https://github.com/radioxeth/voice-attack/.

Shen, Johnathan, et al. “Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions.” *arXivLabs*, 2017, arXiv:1712.05884.

Wang, et al. “DeepSonar: Towards Effective and Robust Detection of AI-Synthesized Fake Voices.” *arXiv*, 2020.

Yang, Y., et al. “. TorchAudio: Building Blocks for Audio and Speech Processing.” *arXiv preprint*, 2021, arXiv:2110.15018.

Yang, Yao, and Moto Hira. “Text-to-Speech with Tacotron2 — Torchaudio 2.2.0 documentation.” *PyTorch*, https://pytorch.org/audio/stable/tutorials/tacotron2_pipeline_tutorial.html. Accessed 5 March 2024.