Deepfake Audio Detection

Project 1 - CSCI316

Advaith Sujith - 7837525
Rhythm Shahi - 7877304
Asraa Shaikh - 7859557
Anushka Roy - 7753226

Contents

Problem Statement

As deepfake technology continues to get more advanced, the rise of deepfake audio has left people questioning the authenticity of online verbal exchanges, effectively limiting their usefulness in various fields and industries, including journalism, law enforcement, politics, and everyday personal communication. Created through AI-generated technology that uses audio recordings to create highly realistic synthesized human voices, it is nearly impossible for an untrained listener to distinguish what's real and what's not.

Originally made with beneficial purposes such as adding a human touch to audiobooks and assisting speech-impaired people through Text-to-Speech technology, deepfake audio has evolved into an ongoing cybersecurity concern, from manipulating public opinion to falsely accusing individuals (Lyu et al.,2020).

The growing prevalence of this threat is emphasized by a recent global survey conducted by Regula. Their findings revealed that 49% of companies experienced audio deepfakes, up from 37% to 49% from 2022 to 2024. This highlighted the need for enhanced detection and prevention measures (Regula.,2024).

Detecting such convincingly altered media through advanced machine learning techniques and ensuring their reliable identification is crucial in safeguarding the authenticity of online discourse and mitigating the harmful consequences of deepfake audios. This project aims to address this issue by developing and training an ensemble model utilizing various ML techniques and conducting cross-validation testing to evaluate the model's performance.

Introduction To the Dataset

We used a dataset that comprises of two key components; a collection of .wav audio files and a corresponding CSV file containing the binary truth labels.

Audio Files: The .wav audio files, derived from the Kaggle dataset called “in the wild”, constitute the raw audio data used in this project. These audio files contain recordings of both real and fake audio samples.

Metadata: Accompanying the .wav files is a CSV file that provides the binary truth labels for each audio sample. This file serves as a reference table linking each .wav file to its corresponding label, specifying whether it's real or fake, therefore providing the appropriate classifications the model learns from.

How Machine Learning Can Help

Detecting Altered Speech Patterns:

Machine learning can identify unnatural modifications to speech patterns in cloned or manipulated audio by analyzing acoustic features like articulation, pitch, and intonation. Cloned audio may exhibit unnatural transitions, reduced pitch variation, or unusual timing between phonemes, which can be flagged as indicators of manipulation.

Identifying Artifacts from Manipulation:

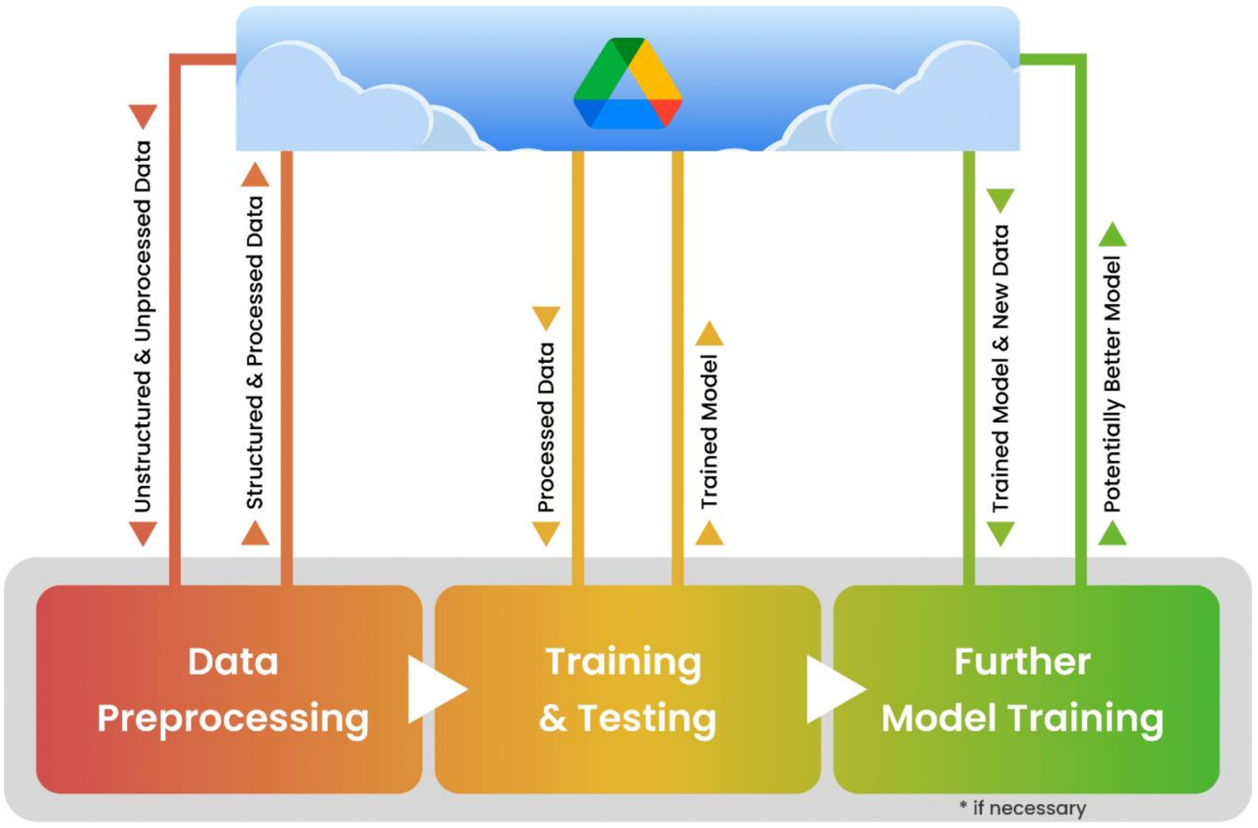
Deepfake audio often introduces artifacts during the synthesis or blending process. Machine learning models can detect these anomalies using features like:

- **Spectral Centroid Mean**: Highlights unusual frequency shifts caused by blending or editing audio sources.
- **MFCC Features**: Reveals subtle distortions in the frequency spectrum caused by splicing or altering speech.
- **Zero Crossing Rate (ZCR)**: Identifies unnatural transitions between voiced and unvoiced sounds, often resulting from imperfect splicing.

Tracking Signal Inconsistencies:

Deepfake audio manipulations can leave temporal or spatial irregularities in the audio signal, such as abrupt changes in background noise or reverberation patterns. Machine learning algorithms can detect these mismatches, which often indicate tampering.

Our Architecture



This figure represents our architecture and process in our ML Operation. Let's understand it further.

We decided to approach this problem from an MLops perspective where we used cloud storage to store our data. The service we used for cloud storage is Google Drive. Ideally, we would use a cloud service with more functionality for coding, such as Microsoft Azure or AWS. However, we were limited by financial requirements.

Our architecture involves three key components: data preprocessing, training and testing, and further model training. Unstructured and processed data are stored in Google Drive, which acts as a central hub for seamless data flow between these stages. The reason we decided to use this approach is that almost all businesses that use machine learning in their practices follow an MLops life cycle, and this architecture can also work if the static dataset were to be replaced by a real-time dataset such as an API.

Ideally, the Data Preprocessing, Training and Testing, and Further Model Training would each be implemented in separate Python scripts. However, to simplify the explanation of the project, we chose to consolidate everything into a single Colab notebook.

- First, the Unstructured and Unprocessed Data is pulled from Google Drive. The data is raw .wav files of various lengths and is a mix between both real and fake audio files.
- This data is then processed using Librosa, a Python library that can be used to extract relevant information for audio files.
- Once the relevant information is collected, processed, and using the meta csv file, is labeled, we then store the clean and structured data back into the cloud.
- Then, this data is pulled by the Training and testing script which then uses the data to train an ensemble model, which is then stored in the cloud again.
- If new data is added, we can easily update our model. Since it is saved in the cloud, we can load and prepare the new data, retrain the model, and save it back to the cloud.

Data Preparation & Cleaning

To effectively train a machine learning model for deepfake audio detection, it is crucial to represent the audio data in a way that highlights the subtle differences between real and fake audio. Utilizing Librosa, a Python library used for audio and music analysis, we extracted a set of relevant features from each second of an audio file. We used PySpark to process the extraction. This significantly reduced the time to run the code due to Spark's parallelism. Our testing showed that Spark increased the time to process the data by 100%. The features of the dataset are,

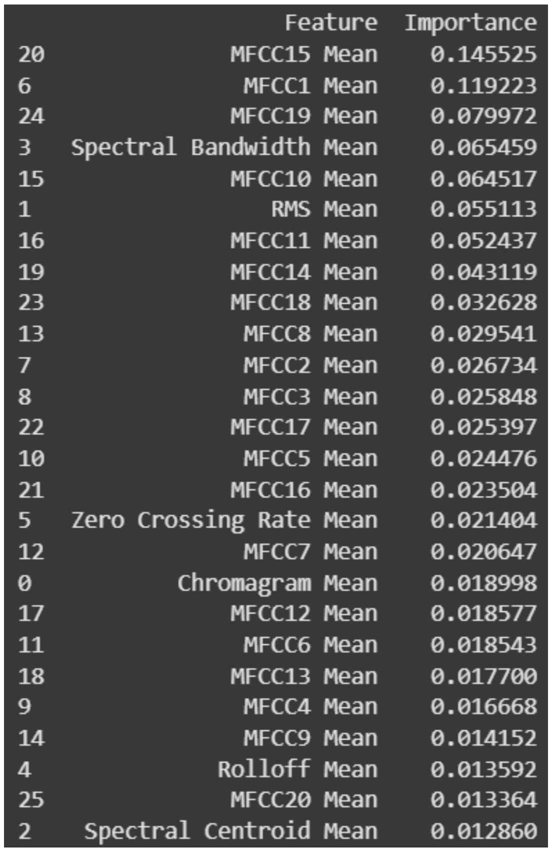
- **Chromagram Mean:** This feature refers closely to entire spectral audio information mapped into one octave that represents the tonal content of audio files. This indicates the presence of specific pitches and chords.
- **RMS Mean(Root Mean Square):** Measures the amount of continuous power an audio signal produces. It is useful in understanding the intensity and variations in loudness over time.
- **Spectral Centroid Mean:** A feature in audio signaling that represents the center of mass of the frequency spectrum, reflecting the dominant frequency content of a signal. It recognizes distinct voice characteristics and audio spoofing by measuring the weighted mean of frequencies(Sharafudeen et al.,2024).
- **RollOff Mean:** This feature represents the point below which a certain percentage of the total spectral energy is concentrated. It gives the rate of frequency at which high frequencies reduce to 0.
- **Zero Crossing Rate Mean:** The average number of times a wave signal crosses zero per unit of time across the horizontal time axis. Used primarily to recognize percussive vs pitched sounds, distinguish voiced and unvoiced speech, etc (Chauhan,2020).

- **MFCC's mean(Mel Frequency Cepstral Coefficients):** In speech processing, the Mel-frequency cepstrum is a key technique that transforms the frequency distribution of a sound to resemble human hearing, producing various coefficients that are tailored to human vocal and auditory systems. These MFCCs represent the distribution of a speaker's vocal energy across different frequencies and timeframes.

The way we extracted the data wasn't in such a way that each audio file was transformed into one record of data. We felt like we would be losing a large amount of data that way. Therefore, we decided to split each sound file into one second bits, which we then converted into a record respectively. Which means for example, if there was a 60 second audio clip, it would give us 60 records of data with each of the features above.

Unsupervised Learning & Data Analysis

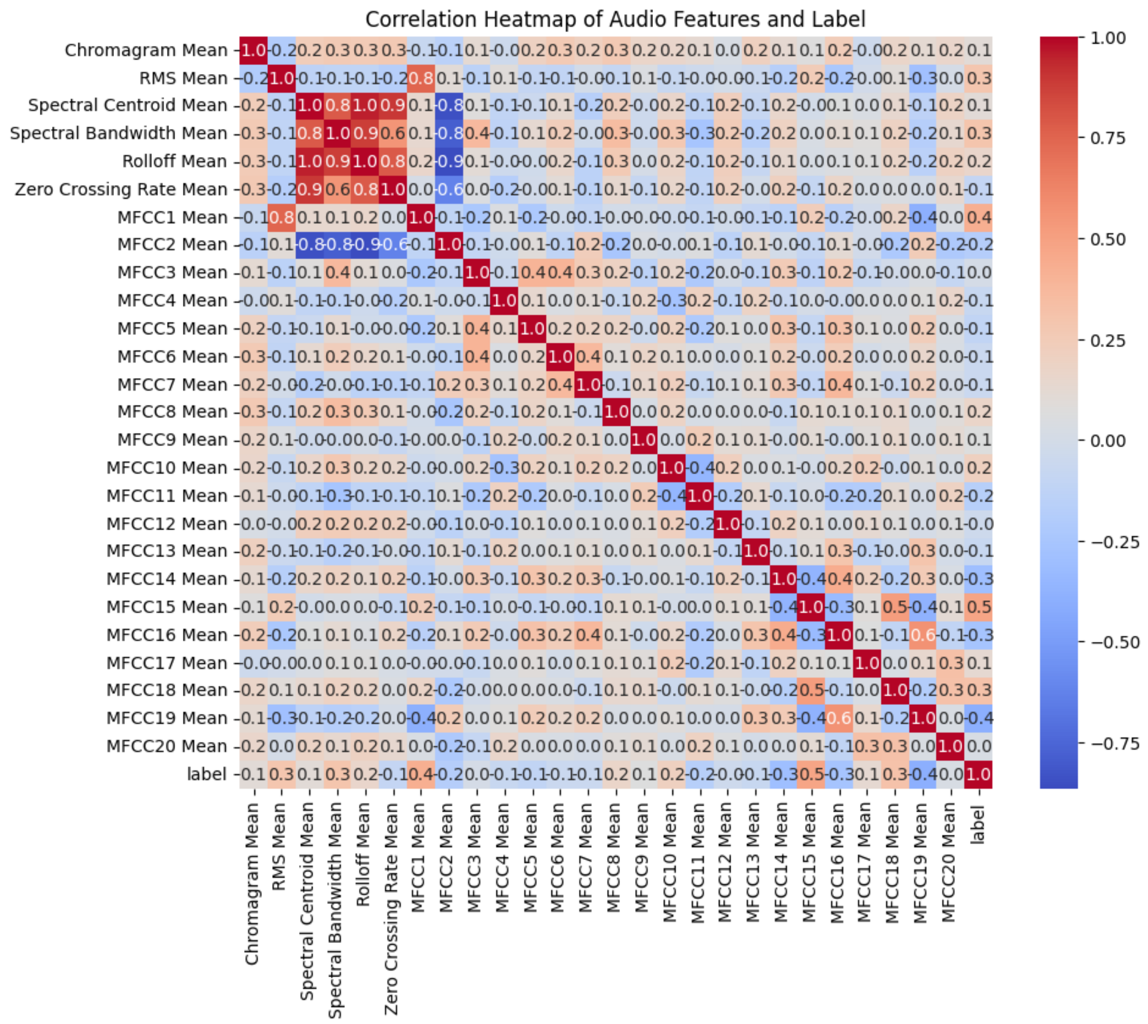
After we got a clean dataset, we decided to do some unsupervised learning and data analysis on our dataset to understand trends and how our features affect the final label. We decided to start off with Feature Importance Analysis.



```
                     Feature  Importance
20               MFCC15 Mean    0.145525
6                 MFCC1 Mean    0.119223
24               MFCC19 Mean    0.079972
3    Spectral Bandwidth Mean    0.065459
15               MFCC10 Mean    0.064517
1                   RMS Mean    0.055113
16               MFCC11 Mean    0.052437
19               MFCC14 Mean    0.043119
23               MFCC18 Mean    0.032628
13                MFCC8 Mean    0.029541
7                 MFCC2 Mean    0.026734
8                 MFCC3 Mean    0.025848
22               MFCC17 Mean    0.025397
10                MFCC5 Mean    0.024476
21               MFCC16 Mean    0.023504
5    Zero Crossing Rate Mean    0.021404
12                MFCC7 Mean    0.020647
0            Chromagram Mean    0.018998
17               MFCC12 Mean    0.018577
11                MFCC6 Mean    0.018543
18               MFCC13 Mean    0.017700
9                 MFCC4 Mean    0.016668
14                MFCC9 Mean    0.014152
4              Rolloff Mean    0.013592
25               MFCC20 Mean    0.013364
2     Spectral Centroid Mean    0.012860
```

We could see that some features were not as strong as the others, so we decided to plot a Correlation Heatmap to further understand them.

Correlation Heatmap-Matrix:

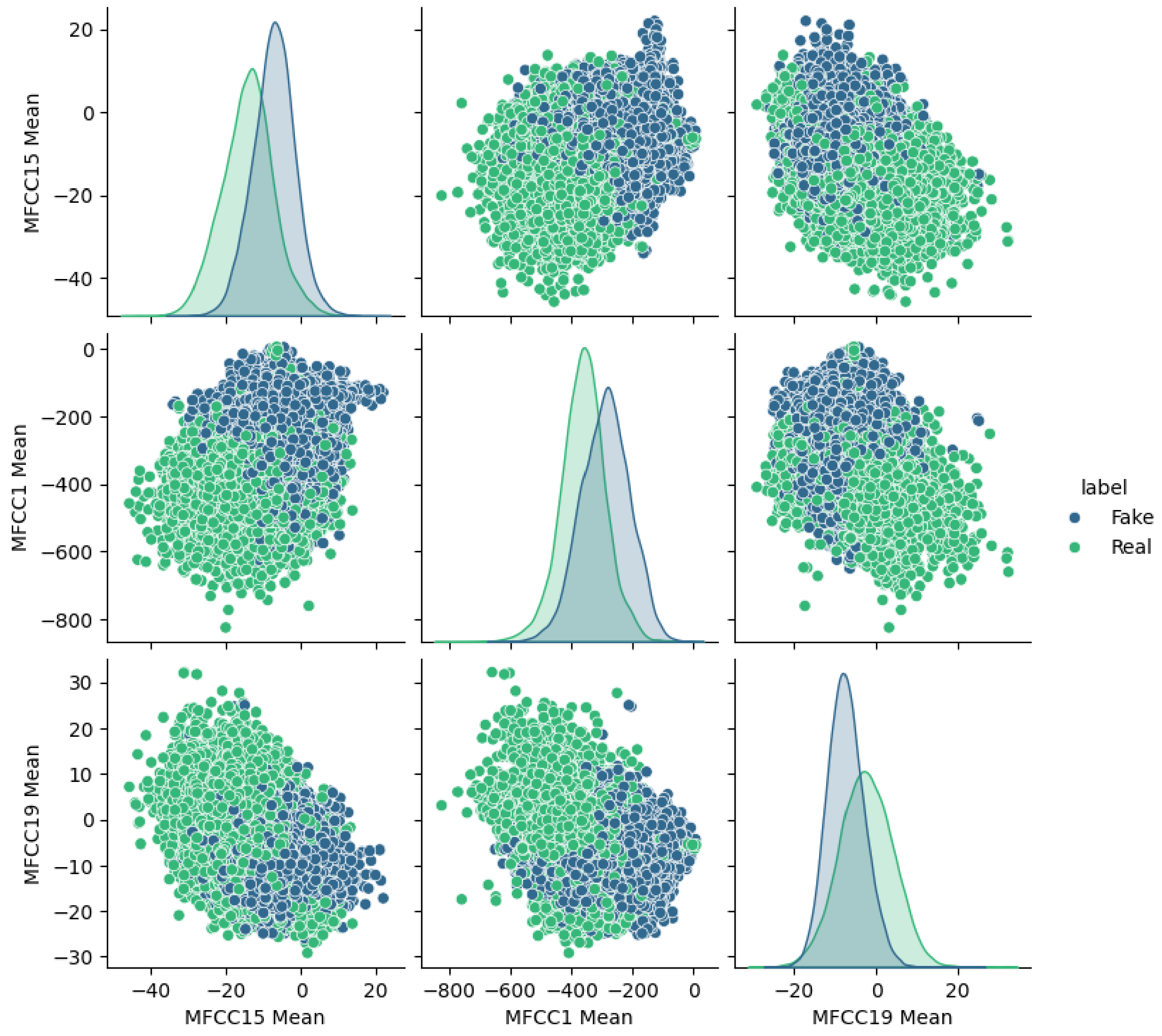


Immediately we can notice a strong set of relationships between spectral centroid mean, spectral bandwidth mean, rolloff mean, and zero crossing rate mean. Implying that if one of them goes up, the other one goes up as well (given how strong the correlation coefficient is). However, something interesting is these same features have a negative correlation coefficient with the MFCC2 Mean.

Moving on to the Label, we do see some strong relations between some of the features, and some of them have a correlation coefficient of 0. However, we decided to not drop these columns yet, as correlation matrices only look for linear relationships between features. If there were more complex relationships between these features the matrix wouldn't show it.

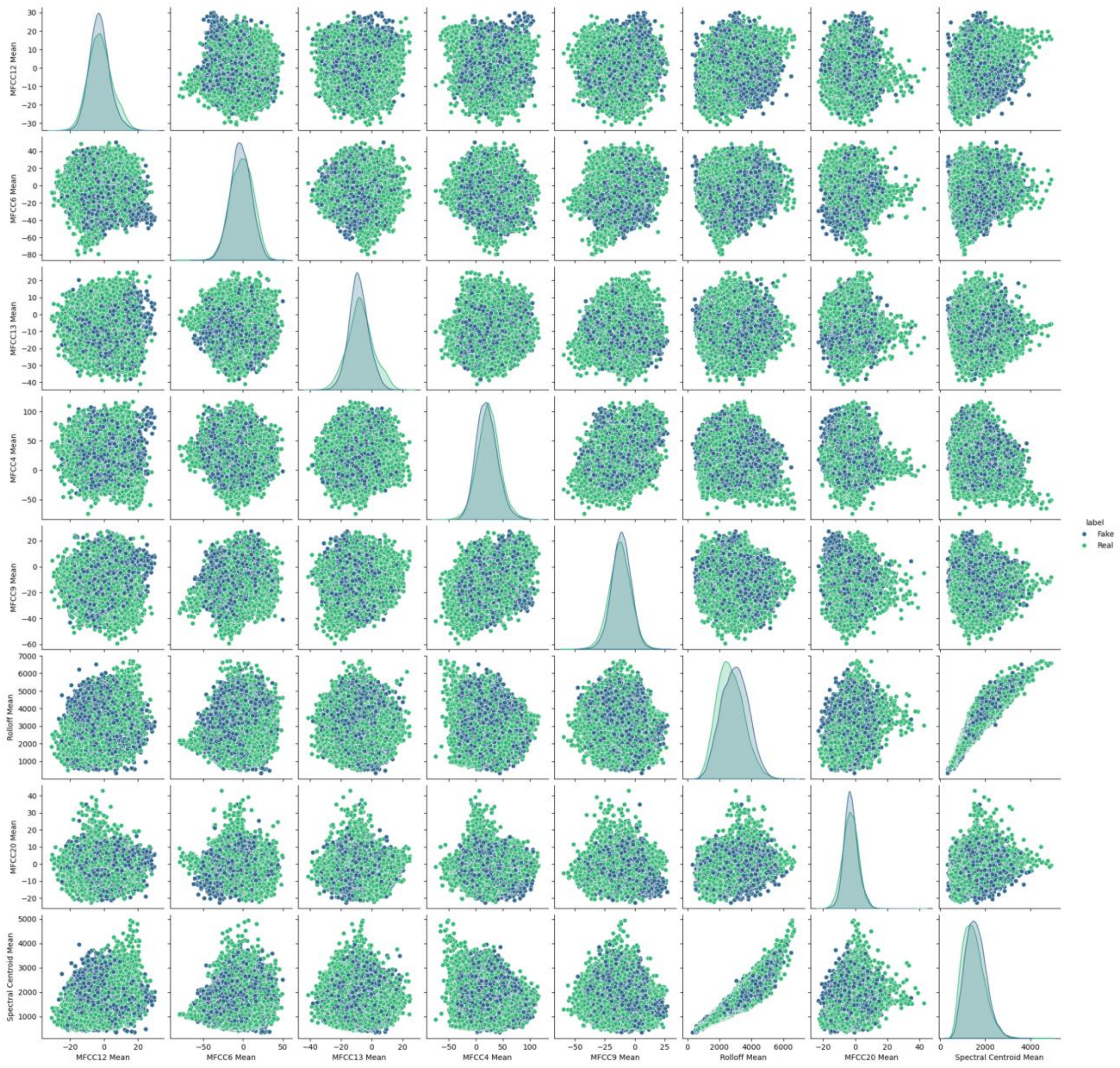
To further understand the strong and weak relations, we used a pairplot from Seaborn to visualize the correlation.

SNS Pairplot for Best Features:



We can see a clear separation in data points of the fake and real audio. This is supported by the Kernel Density Estimations (KDE) on the diagonals.

SNS Pairplot for Worst Features:



These are the weakest performing features in our dataset. These may be overwhelming but all we need to look at are the KDE's on the diagonals. There is a lot of overlapping between real and fake audio data therefore it is not helpful to our model. Due to this, we decided to drop the 8 weakest features.

Ensemble Model

We chose Bagging as our primary ensemble model to address the high variance in our predictions, given the wide nature of our dataset. This approach helps to stabilize and improve the model's performance by reducing variance across individual learners.

The decision to use Logistic Regression as the base learner was influenced by the simplicity and interpretability of the model, which pairs well with ensemble techniques. Additionally, since Bagging reduces variance, the inherent bias of Logistic Regression becomes less of a concern, allowing us to benefit from its efficiency and scalability.

By using an ensemble of 100 Logistic Regression models, we leverage the diversity introduced by random sampling to capture subtle patterns in the dataset while smoothing out noise and inconsistencies.

To confirm our hypothesis about Bagging being better for our dataset, we did also try a Boosting model which gave us worse accuracy.

Training

We used sklearn's train_test_split function to divide the data into training (80%) and testing (20%) sets. The training data was further processed through the BaggingClassifier, ensuring the creation of diverse training subsets. This diversity is key to the success of Bagging, as it ensures that each base learner trains on a slightly different perspective of the data, enhancing generalization.

To optimize the performance of the BaggingClassifier, we experimented with the number of estimators (n_estimators) and set it to 100 based on initial validation results. Additionally, Logistic Regression models were configured with a maximum iteration of 1000 to ensure convergence on the training subsets.

Testing

To evaluate the performance and generalizability of our ensemble model, we utilized a 10-fold cross-validation strategy. The following procedure was implemented for testing.

Data Partitioning: The training data, which is 80% of the original dataset after using train test split, is divided into 10 equal-sized folds. This partition was performed using a custom cross_validate function. The data is also shuffled randomly before partitioning to ensure a representative distribution of samples across all folds.

Cross-Validation Loop: The evaluation was conducted within a loop iterating 10 times, where in each iteration:

- An instance of the BaggingClassifier with 100 estimators was used. This ensured each fold's evaluation was independent, preventing any bias.
- The trained bagging classifier makes predictions on the held-out test fold.
- The accuracy of the predictions on the test fold is calculated and stored

Final Evaluation: The trained bagging classifier is used to make predictions on the held-out 20% test set. This final evaluation provides the measure of our model's performance on unseen data, indicating how well it would perform in real-world scenarios. The cross-validation of these predictions on the test set was calculated to be ~88.21%. The final accuracy came out to ~88.06%.

References

Ali, G., Rashid, J. and Usman, M. (2024) *Arxiv*, *Beyond the Illusion: Ensemble Learning for Effective Voice Deepfake Detection*. Available at: https://ieeexplore.ieee.org/document/10676991/authors#authors (Accessed: 07 February 2025).

Audio Feature Extraction (2021) *Devopedia*. Available at: https://devopedia.org/audio-feature-extraction#Chauhan-2020 (Accessed: 06 February 2025).

Chauhan, N.S. (2020) *Audio data analysis using Deep Learning with python (part 1)*, *KDnuggets*. Available at: https://www.kdnuggets.com/2020/02/audio-data-analysis-deep-learning-python-part-1.html (Accessed: 05 February 2025).

Deepfake fraud doubles down: Regula survey (no date) *Regula*. Available at: https://regulaforensics.com/news/deepfake-fraud-doubles-down/ (Accessed: 05 February 2025).

Lyu, S. (no date) *Deepfake detection: Current challenges and next steps | IEEE conference publication | IEEE Xplore*. Available at: https://ieeexplore.ieee.org/abstract/document/9105991 (Accessed: 06 February 2025).

Preeti Kale (2023) *(PDF) deepfake audio detection and justification with explainable artificial intelligence (XAI)*. Available at: https://www.researchgate.net/publication/374798687_Deepfake_audio_detection_and_justification_with_Explainable_Artificial_Intelligence_XAI (Accessed: 05 February 2025).

Sharafudeen, M. *et al.* (2024) *A blended framework for audio spoof detection with sequential models and bags of auditory bites*, *Scientific reports*. Available at: https://pmc.ncbi.nlm.nih.gov/articles/PMC11364551/ (Accessed: 05 February 2025).