International Technological University

Gender Recognition Using Voice

By

Barrenkala Maheswar

ID: 38433315

Professor: John Kim

Subject: **Deep Learning Programming**

Table of Contents

Introduction

Purpose

In recent years, deep learning techniques have demonstrated significant success in pattern recognition and signal processing. This project aims to harness the power of deep learning to develop an advanced Gender Recognition by Voice (GRV) system.

Scope

The primary objective is to create a robust and accurate model capable of discerning gender based on acoustic features extracted from speech signals. The project will span key stages, including data preprocessing, feature extraction, model training, and performance evaluation.

Audience

This project is designed for researchers, developers, and practitioners interested in the application of deep learning to gender recognition from voice data. The documentation provides insights into the methodology, techniques, and considerations involved in building an effective GRV system.

Abstract

Title: Gender Recognition by Voice Using Deep Learning

In recent years, the application of deep learning techniques has demonstrated remarkable success in various fields, particularly in the realm of pattern recognition and signal processing. This project aims to leverage deep learning methodologies for the development of an advanced Gender Recognition by Voice (GRV) system. The primary objective is to create a robust and accurate model capable of discerning gender based on the acoustic features extracted from speech signals.

The proposed system will employ state-of-the-art deep neural network architectures, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), to learn intricate patterns within voice data. The input dataset will comprise diverse samples of speech recordings obtained from a wide range of individuals, encompassing different age groups, linguistic backgrounds, and accents.

The project will involve several key stages, including data preprocessing, feature extraction, model training, and performance evaluation. Feature extraction will focus on capturing relevant acoustic characteristics, such as pitch, formants, and spectral features, to ensure the model's ability to generalize across diverse voices. The deep learning model will be trained on a labeled dataset with gender annotations, employing optimization techniques to enhance its learning capabilities.

The effectiveness of the proposed GRV system will be assessed through rigorous performance evaluation metrics, including accuracy, precision, recall, and F1 score. Additionally, the model will undergo testing on an independent dataset to validate its generalization capabilities and robustness across different voice samples.

The successful implementation of this project holds potential applications in various domains, including human-computer interaction, voice-based user authentication, and gender-specific market research. Moreover, the project contributes to the broader discourse on the ethical considerations surrounding gender recognition technologies, emphasizing the importance of fairness and avoiding biases in model predictions.

Overview

Background:

Gender recognition using voice has gained significant importance in various applications, including human-computer interaction, security systems, and accessibility features. Traditional methods often rely on superficial stereotypes, but advancements in machine learning and signal processing have opened avenues for more accurate and unbiased gender identification based on vocal patterns.

This system leverages state-of-the-art technologies to analyze and interpret voice characteristics, providing a more nuanced and reliable approach to gender recognition. By extracting meaningful features from voice data, the system aims to move beyond simplistic gender stereotypes and contribute to a more inclusive and sophisticated understanding of speaker identity.

Objectives:

The primary objectives of the gender recognition system are:

Accuracy: Develop a model that achieves high accuracy in determining the gender of a speaker, minimizing false positives and false negatives.

Robustness: Ensure the model's performance is consistent across diverse datasets and is not significantly affected by variations in accent, language, or recording conditions.

Ethical Considerations: Address ethical concerns by prioritizing fairness and avoiding biases related to gender stereotypes or cultural influences.

Scalability: Design the system to handle a variety of applications, from small-scale integrations to large-scale deployments in real-world scenarios.

Interpretability: Provide transparency in the decision-making process of the model to enhance trust and understanding, especially in applications where accountability is crucial.

Key Features:

Comprehensive Data Collection

The system utilizes a diverse and representative dataset, encompassing a wide range of voices across different demographics, ensuring inclusivity in gender recognition.

Advanced Preprocessing Techniques

To enhance model performance, the preprocessing pipeline includes sophisticated methods for noise reduction, audio conversion, and normalization, addressing challenges associated with real-world audio data.

Multi-modal Feature Extraction

The system employs a multi-modal approach, extracting features such as Mel-Frequency Cepstral Coefficients (MFCC), pitch, and formants to capture the rich spectral and temporal characteristics of voice signals.

Hybrid Model Architecture

Combining the strengths of both traditional machine learning models and deep neural networks, the system employs a hybrid architecture to leverage the benefits of different approaches.

Confidence Level Estimation

The system provides a confidence level for each gender prediction, allowing users to gauge the reliability of the model's output and facilitating decision-making in applications that require high precision.

Continuous Improvement

The system is designed with adaptability in mind, allowing for continuous improvement through regular updates, model retraining, and feedback mechanisms to enhance accuracy and keep pace with evolving speech patterns.

System Architecture

Data Collection:

Dataset Selection

The data collection process begins with the careful selection of a diverse and representative dataset. This dataset includes voices from different genders, age groups, and linguistic backgrounds to ensure the model's generalizability.

Annotation

Each audio sample in the dataset is annotated with the corresponding gender label, providing ground truth information for supervised model training.

Data Augmentation

To enhance the model's robustness, data augmentation techniques such as pitch shifting, time stretching, and background noise injection are applied. This helps mitigate biases related to specific recording conditions.

Preprocessing:

Audio Conversion

Raw audio files are converted to a standardized format (e.g., WAV) to facilitate consistent processing. This step ensures uniformity in the input data across various recording sources.

Noise Reduction

Advanced noise reduction algorithms, such as spectral subtraction or deep learning-based denoising, are employed to minimize the impact of background noise, improving the signal-to-noise ratio.

Normalization

Normalization techniques are applied to standardize the amplitude of audio signals, preventing variations in volume from influencing the model's training and performance.

Feature Extraction:

Mel-Frequency Cepstral Coefficients (MFCC)

MFCCs are extracted from preprocessed audio signals to capture spectral characteristics. The system typically considers the first few coefficients that best represent the speaker's voice.

Pitch Analysis

Pitch information is extracted to capture variations in vocal pitch, which can be a significant factor in distinguishing between male and female speakers.

Formant Analysis

Formant frequencies are analyzed to capture the resonance properties of the vocal tract, providing additional discriminative features for gender recognition.

Model Training:

Model Selection

The system supports various model architectures, including Support Vector Machines (SVMs), Convolutional Neural Networks (CNNs), and recurrent models like Long Short-Term Memory networks (LSTMs). The choice of the model depends on the dataset size, complexity, and computational resources.

Hyperparameter Tuning

Hyperparameters such as learning rates, regularization terms, and batch sizes are tuned to optimize the model's performance during the training phase.

Cross-Validation

To ensure the model's generalization capability, cross-validation is employed, assessing performance on multiple folds of the dataset.

Gender Recognition:

Inference

The trained model is used to make gender predictions on new, unseen voice samples. This involves feeding the extracted features into the model and obtaining probability scores for each gender class.

Decision Threshold

A decision threshold is applied to convert probability scores into binary gender predictions. This threshold can be adjusted based on the desired balance between precision and recall.

Confidence Level:

Confidence Metrics

The system calculates confidence metrics, such as probability scores or entropy, to quantify the model's certainty in its predictions.

Confidence Threshold

A confidence threshold is set to filter out predictions with low confidence, ensuring that only reliable gender classifications are presented to the end-user or application.

Data Collection

Dataset:

Dataset Source

The dataset used for gender recognition is sourced from diverse and representative collections, such as the VoxCeleb dataset, which contains audio samples from a wide range of individuals including celebrities, ensuring inclusivity in terms of age, ethnicity, and accent.

Labeling

Each audio sample in the dataset is meticulously labeled with the corresponding gender information. The labeling process ensures the availability of ground truth data, essential for supervised learning during model training.

Dataset Size

The dataset comprises a substantial number of samples, balancing the need for model complexity with computational feasibility. The size is carefully chosen to avoid overfitting while capturing the diversity necessary for robust gender recognition.

Recording Setup:

Controlled Environment

Voice samples are recorded in a controlled environment to minimize external factors influencing the recording quality. This helps maintain consistency across the dataset and reduces the impact of extraneous variables.

High-Quality Microphones

Recording setups incorporate high-quality microphones to capture accurate representations of voice characteristics. This ensures the fidelity of the data, contributing to the system's ability to generalize well.

Multi-language Support

The dataset includes voice samples from speakers of various languages, ensuring the model's ability to recognize gender across linguistic diversity.

Data Augmentation:

Pitch Shifting

To simulate variations in vocal pitch, pitch-shifting techniques are applied to the audio samples. This augmentation enhances the model's ability to generalize across speakers with different pitch ranges.

Time Stretching

Time stretching is employed to introduce variations in speech rate, accommodating speakers who may naturally speak at different speeds.

Background Noise Injection

The dataset is augmented with background noise to expose the model to various acoustic environments, making it more resilient to real-world scenarios with non-ideal recording conditions.

Gender Balancing

Care is taken to ensure a balanced representation of both genders in the dataset, preventing biases that may arise from an unequal distribution.

Data Quality Control:

Anomaly Detection

Automated processes are implemented to identify and exclude anomalous samples, such as recordings with significant distortion or artifacts.

Speaker Verification

In addition to gender labels, a subset of the dataset is reserved for speaker verification tasks, ensuring that speakers exhibit consistent voice characteristics across multiple samples.

Preprocessing

Audio Conversion:

Format Standardization

Raw audio files are converted into a standardized format, commonly WAV, to ensure uniformity in the dataset. This simplifies the preprocessing pipeline and facilitates compatibility with various audio-processing libraries and tools.

Sample Rate Consistency

Audio samples are resampled to a consistent sample rate. This helps in preventing discrepancies that may arise from recordings with different sample rates, ensuring homogeneity across the dataset.

Noise Reduction:

Background Noise Identification

Advanced algorithms are employed to identify and characterize background noise in audio samples. This includes techniques such as spectral analysis and machine learning-based approaches.

Noise Profile Modeling

A model of the identified background noise is created, enabling adaptive noise reduction. This ensures that the model is capable of suppressing specific types of noise encountered in real-world scenarios.

Dynamic Noise Thresholding

Dynamic thresholding techniques are applied to adjust the level of noise reduction based on the characteristics of the input audio. This prevents over-filtering, which could otherwise distort the original voice signal.

Normalization:

Amplitude Normalization

Amplitude normalization is performed to standardize the volume levels across all audio samples. This is crucial for creating a consistent input for the subsequent feature extraction and model training stages.

Z-score Standardization

Z-score standardization is applied to ensure that the mean and standard deviation of the audio signal are centered around zero and one, respectively. This normalization technique helps mitigate the impact of outliers in the data.

Dynamic Range Compression

Dynamic range compression is employed to control the amplitude variations within an audio sample. This is particularly useful in maintaining audible details while preventing distortion in louder sections.

Resampling:

Uniform Sampling

Audio samples may be resampled to a uniform duration, ensuring consistency in the length of input sequences for the subsequent feature extraction process. This is particularly relevant for models that require fixed-length input.

Time-Frequency Resampling

For certain feature extraction methods, time-frequency resampling techniques may be applied to adapt the temporal and spectral resolution of the audio data, optimizing it for specific feature extraction algorithms like Mel-Frequency Cepstral Coefficients (MFCC).

The preprocessing stage plays a crucial role in enhancing the quality and uniformity of the input data, preparing it for effective feature extraction and subsequent model training. The combination of audio conversion, noise reduction, and normalization techniques contributes to the creation of a standardized and reliable dataset for gender recognition.

Feature Extraction

Mel-Frequency Cepstral Coefficients (MFCC):

Mel Filter bank

The audio signal is first passed through a filterbank of mel filters, spaced non-linearly to mimic the human auditory system's sensitivity to different frequencies. This captures the distribution of energy across various frequency bands.

Discrete Cosine Transform (DCT)

The log mel spectrogram is then subjected to a DCT to decorrelate the mel-frequency coefficients, resulting in a set of MFCCs. These coefficients represent the spectral characteristics of the audio signal, emphasizing the most discriminative features for gender recognition.

Frame-Level Representation

MFCCs are typically computed over short, overlapping frames of the audio signal, creating a frame-level representation. This accounts for the dynamic nature of speech signals, capturing changes in vocal characteristics over time.

Pitch:

Pitch Extraction

Pitch extraction involves determining the fundamental frequency of the voice, representing the perceived pitch. Techniques such as autocorrelation, cepstral analysis, or Fourier transform-based methods may be employed to estimate pitch values.

Pitch Histogram

A pitch histogram is created, summarizing the distribution of pitch values over the duration of the audio signal. This histogram provides insights into the overall pitch characteristics, which can be crucial for distinguishing between male and female speakers.

Statistical Descriptors

Mean, standard deviation, and other statistical descriptors are computed from the pitch histogram to create a concise pitch feature vector. These descriptors capture the central tendency and variability of pitch values in the voice signal.

Formants:

Formant Extraction

Formants are resonant frequencies in the vocal tract that contribute to the distinctiveness of vowels. Extraction involves identifying these formant frequencies through techniques like linear predictive coding (LPC) analysis.

Formant Trajectories

Formant trajectories, representing the changes in formant frequencies over time, are tracked throughout the duration of the audio signal. This captures the dynamic nature of speech and contributes to the gender-specific characteristics used for recognition.

Formant Ratio Analysis

Ratios of formant frequencies or formant bandwidths are computed, providing additional discriminative features. Gender-related differences in vocal tract length and size influence these ratios, contributing to the overall gender classification process.

Multi-modal Fusion:

Integration of Features

The extracted features, including MFCCs, pitch characteristics, and formant ratios, are integrated into a comprehensive feature vector. This multi-modal representation captures both spectral and temporal aspects of the voice signal, enhancing the model's ability to discern gender-related patterns.

Feature Scaling

Feature scaling is applied to ensure that each feature contributes proportionally to the model, preventing biases toward features with larger magnitudes.

Feature extraction, encompassing MFCCs, pitch, and formants, provides a rich representation of vocal characteristics. The combination of these features forms a comprehensive input for the subsequent stages of model training and gender recognition.

Model Training

Machine Learning Models:

Support Vector Machines (SVM)

Kernel Functions: Various kernel functions (linear, polynomial, radial basis function) are explored to capture complex relationships in the feature space.

Hyperparameter Tuning: Grid search or randomized search is employed to optimize parameters such as the regularization term (C) and kernel parameters.

Random Forests

Ensemble Learning: Decision trees are aggregated into a random forest, leveraging the diversity of multiple trees to enhance generalization.

Feature Importance: Analysis of feature importance guides the model interpretation and provides insights into the discriminative power of different features.

Gradient Boosting

Boosting Algorithm: Boosted models, such as XGBoost or LightGBM, are trained to sequentially correct errors made by preceding models, leading to improved accuracy.

Tree Depth and Learning Rate: Hyperparameters like tree depth and learning rate are fine-tuned to achieve optimal model performance.

Deep Learning Models;=;

Convolutional Neural Networks (CNN)

Spectral Image Representation: Audio spectrograms or other representations are treated as images, allowing the use of CNNs for learning hierarchical features.

Convolutional Layers: Convolutional layers with varying kernel sizes capture local and global patterns in the input spectrogram.

Recurrent Neural Networks (RNN);

Temporal Sequences: RNNs, especially Long Short-Term Memory (LSTM) networks, are employed to model temporal dependencies in sequential audio data.

Bidirectional Architectures: Bidirectional LSTMs capture information from both past and future time steps, enhancing the model's context awareness.

Hybrid Architectures;

Combining CNN and RNN: Hybrid architectures may integrate CNNs for spatial feature extraction and RNNs for sequential modeling, providing a holistic approach to voice-based gender recognition.

Attention Mechanisms: Attention mechanisms are incorporated to focus on specific regions of interest in the input data, enhancing the model's interpretability.

Transfer Learning;

Pre-trained Models

ImageNet Transfer: Leveraging pre-trained models from image classification tasks (e.g., ResNet, VGG16) and adapting them for audio-based gender recognition through transfer learning.

Fine-tuning Layers: Fine-tuning specific layers of pre-trained models allows the network to learn gender-specific features from the audio data.

Domain Adaptation:

Adapting to Audio Domains: Techniques like domain adaptation are explored to align the distribution of features between the source (ImageNet) and target (voice) domains.

Feature Alignment Networks: Networks that explicitly learn domain-invariant features are employed to enhance the transferability of the model.

Model Evaluation:

Performance Metrics

Accuracy and F1 Score: Standard metrics for evaluating classification performance on gender recognition tasks.

Confusion Matrix: A detailed breakdown of true positives, true negatives, false positives, and false negatives to assess model performance across different gender classes.

Cross-Validation

K-fold Cross-Validation: The dataset is partitioned into k folds, with each fold serving as a validation set at least once. This helps estimate the model's generalization performance.

Stratified Sampling: Ensures that each fold maintains the same distribution of gender labels as the entire dataset, preventing bias in the cross-validation process.

Model Optimization:

Hyperparameter Tuning

Bayesian Optimization: Advanced optimization techniques, such as Bayesian optimization, are applied to efficiently search the hyperparameter space for optimal configurations.

Regularization Techniques

Dropout and Batch Normalization: Regularization techniques, like dropout and batch normalization, are employed to prevent overfitting and improve model generalization.

Ensemble Learning

Model Ensembles: Multiple models with diverse architectures or trained on different subsets of the data are combined through ensemble methods to enhance overall prediction accuracy and robustness.

Model Deployment:

Exporting Trained Models

Model Serialization: Trained models are serialized for deployment using industry-standard formats such as TensorFlow SavedModel.

API Integration

RESTful API: A RESTful API is established to facilitate integration with various applications and services, enabling real-time gender recognition.

Cloud Deployment

Containerization: The model is containerized (e.g., Docker) for seamless deployment across cloud platforms, ensuring scalability and accessibility.

Continuous Monitoring

Model Health Checks: Continuous monitoring mechanisms are implemented to track the model's performance and trigger retraining or updates based on evolving data patterns.

Model training involves exploring a range of machine learning and deep learning approaches, optimizing hyperparameters, and ensuring robust performance through evaluation metrics. The trained models are then prepared for deployment in real-world applications.

Gender Recognition

Inference:

Feature Input

For gender recognition inference, preprocessed audio features, such as MFCCs, pitch, and formants, are fed into the trained model. The model processes this input to generate predictions for the gender of the speaker.

Real-time Inference

In real-time applications, the model is capable of making predictions on streaming audio data, allowing for instantaneous gender recognition during live interactions.

Decision Threshold:

Probability Scores

The model outputs probability scores for each gender class, representing the likelihood of the speaker being male or female. These scores are obtained through softmax activation in the output layer of the model.

Dynamic Thresholding

A decision threshold is applied to convert the probability scores into binary gender predictions. The threshold may be dynamically adjusted based on the desired trade-off between precision and recall, adapting to different application requirements.

Receiver Operating Characteristic (ROC) Analysis

ROC analysis may be conducted to determine the optimal decision threshold, considering the trade-offs between true positive rate and false positive rate.

Post-processing:

Majority Voting

In ensemble models or scenarios where multiple predictions are available, a majority voting mechanism may be employed. The final gender prediction is determined by the most frequently predicted class across different models or samples.

Confidence Filtering

Predictions with low confidence scores, below a predefined threshold, may be filtered out during post-processing. This helps improve the overall reliability of gender classifications.

Temporal Aggregation

For applications processing audio sequences, temporal aggregation techniques may be applied to refine predictions. This involves considering predictions over a temporal window and aggregating them to produce a more stable and consistent result.

Evaluation:

Confusion Matrix

The confusion matrix is employed to evaluate the model's performance, providing insights into true positives, true negatives, false positives, and false negatives.

Precision, Recall, and F1 Score

Precision, recall, and F1 score are computed to assess the model's ability to correctly identify each gender class, considering both false positives and false negatives.

Receiver Operating Characteristic (ROC) Curve

The ROC curve is utilized to visualize the model's performance across different decision thresholds, helping to identify the optimal operating point.

Real-world Considerations:

Bias Mitigation

Efforts are made to mitigate biases in gender recognition, ensuring that the model performs well across diverse demographics and avoids reinforcing stereotypes.

Sensitivity to Recording Conditions

The model's robustness to variations in recording conditions, such as background noise or microphone types, is evaluated to ensure reliable performance in real-world scenarios.

Gender recognition is not only about accurately predicting gender but also about addressing potential biases, ensuring ethical considerations, and adapting to dynamic real-world conditions. The inference, decision thresholding, and post-processing stages play a crucial role in refining predictions and optimizing the system for practical deployment.

Confidence Level

Confidence Metrics:

Softmax Probabilities

The model outputs softmax probabilities for each gender class. These probabilities represent the model's confidence in its predictions, with higher values indicating greater certainty.

Confidence Score

A confidence score is computed based on the probability scores, providing a single value that quantifies the overall confidence in the gender prediction. This score may be calculated using metrics such as entropy or the difference between the top two probability scores.

Prediction Variability

An analysis of the variability in prediction scores across multiple models or samples is conducted. A more stable and consistent set of probabilities contributes to higher confidence in the predictions.

Confidence Threshold:

Threshold Determination

A confidence threshold is set to determine whether a prediction is considered sufficiently confident. This threshold is application-dependent and can be adjusted based on the desired level of reliability.

Sensitivity Analysis

Sensitivity analysis is performed to assess the impact of varying confidence thresholds on the system's overall performance. This helps strike a balance between the acceptance of predictions and the mitigation of false positives or false negatives.

Confidence Calibration:

Platt Scaling

Probabilistic outputs may undergo calibration techniques such as Platt scaling to align the predicted probabilities with the true likelihood of correctness.

Isotonic Regression

Isotonic regression is applied to adjust the model's probability estimates, especially when the model's confidence scores do not align well with the actual correctness likelihood.

Decision Support:

User Interface Integration

Confidence scores and associated metrics are integrated into user interfaces or application programming interfaces (APIs) to provide users with insights into the reliability of gender predictions.

Threshold Visualization

Visualization tools are developed to illustrate the impact of varying confidence thresholds on the system's performance, allowing users to make informed decisions based on their specific requirements.

Continuous Monitoring:

Confidence Drift Detection

Continuous monitoring mechanisms are established to detect confidence drift over time. Sudden changes in confidence levels may trigger retraining or recalibration processes to maintain the model's reliability.

Model Feedback Loop

Feedback loops are implemented, allowing users to provide feedback on model predictions. This feedback is used to iteratively improve the model and its confidence estimation over time.

Ethical Considerations:

Fairness Assessment

A fairness assessment is conducted to evaluate whether confidence scores exhibit biases across different demographic groups. Efforts are made to address and mitigate any observed disparities.

Explainability

Explainability tools are incorporated to provide insights into how the model arrives at specific confidence scores. This transparency fosters trust and allows users to understand the decision-making process.

The confidence level of the gender recognition system is crucial for applications where reliability is paramount. By carefully assessing and adjusting confidence metrics and thresholds, the system aims to strike a balance between precision and recall while considering ethical implications and real-world considerations.

Code & Result

!pip install numpy pandas tqdm scikit-learn tensorflow pyaudio librosa

```
Requirement already satisfied: numpy in c:\users\dell\anaconda3\lib\site-packages (1.23.5)
Requirement already satisfied: pandas in c:\users\dell\anaconda3\lib\site-packages (1.5.3)
Requirement already satisfied: tqdm in c:\users\dell\anaconda3\lib\site-packages (4.64.1)
Requirement already satisfied: scikit-learn in c:\users\dell\anaconda3\lib\site-packages (1.2.1)
Requirement already satisfied: tensorflow in c:\users\dell\anaconda3\lib\site-packages (2.15.0)
Requirement already satisfied: pyaudio in c:\users\dell\anaconda3\lib\site-packages (0.2.14)
Requirement already satisfied: librosa in c:\users\dell\anaconda3\lib\site-packages (0.10.1)
Requirement already satisfied: pytz>=2020.1 in c:\users\dell\anaconda3\lib\site-packages (from
pandas) (2022.7)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\dell\anaconda3\lib\site-
packages (from pandas) (2.8.2)
Requirement already satisfied: colorama in c:\users\dell\anaconda3\lib\site-packages (from
tqdm) (0.4.6)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\dell\anaconda3\lib\site-packages
(from scikit-learn) (2.2.0)
Requirement already satisfied: joblib>=1.1.1 in c:\users\dell\anaconda3\lib\site-packages (from
scikit-learn) (1.1.1)
Requirement already satisfied: scipy>=1.3.2 in c:\users\dell\anaconda3\lib\site-packages (from
scikit-learn) (1.10.0)
Requirement already satisfied: tensorflow-intel==2.15.0 in c:\users\dell\anaconda3\lib\site-
packages (from tensorflow) (2.15.0)
Requirement already satisfied: absl-py>=1.0.0 in c:\users\dell\anaconda3\lib\site-packages (from
tensorflow-intel==2.15.0->tensorflow) (2.0.0)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in
c:\users\dell\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (0.5.4)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in c:\users\dell\anaconda3\lib\site-packages
(from tensorflow-intel==2.15.0->tensorflow) (1.60.0)
Requirement already satisfied:
protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in
c:\users\dell\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (4.23.4)
Requirement already satisfied: packaging in c:\users\dell\anaconda3\lib\site-packages (from
tensorflow-intel==2.15.0->tensorflow) (22.0)
Requirement already satisfied: setuptools in c:\users\dell\anaconda3\lib\site-packages (from
tensorflow-intel==2.15.0->tensorflow) (65.6.3)
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0 in
c:\users\dell\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (2.15.0)
Requirement already satisfied: libclang>=13.0.0 in c:\users\dell\anaconda3\lib\site-packages
(from tensorflow-intel==2.15.0->tensorflow) (16.0.6)
Requirement already satisfied: tensorboard<2.16,>=2.15 in c:\users\dell\anaconda3\lib\site-
packages (from tensorflow-intel==2.15.0->tensorflow) (2.15.1)
Requirement already satisfied: typing-extensions>=3.6.6 in c:\users\dell\anaconda3\lib\site-
packages (from tensorflow-intel==2.15.0->tensorflow) (4.4.0)
```

Requirement already satisfied: flatbuffers>=23.5.26 in c:\users\dell\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (23.5.26)
Requirement already satisfied: keras<2.16,>=2.15.0 in c:\users\dell\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (2.15.0)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in c:\users\dell\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (0.31.0)
Requirement already satisfied: google-pasta>=0.1.1 in c:\users\dell\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (0.2.0)
Requirement already satisfied: termcolor>=1.1.0 in c:\users\dell\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (2.4.0)
Requirement already satisfied: opt-einsum>=2.3.2 in c:\users\dell\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (3.3.0)
Requirement already satisfied: six>=1.12.0 in c:\users\dell\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (1.16.0)
Requirement already satisfied: astunparse>=1.6.0 in c:\users\dell\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (1.6.3)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in c:\users\dell\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (1.14.1)
Requirement already satisfied: ml-dtypes~=0.2.0 in c:\users\dell\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in c:\users\dell\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (3.7.0)
Requirement already satisfied: soxr>=0.3.2 in c:\users\dell\anaconda3\lib\site-packages (from librosa) (0.3.7)
Requirement already satisfied: decorator>=4.3.0 in c:\users\dell\anaconda3\lib\site-packages (from librosa) (5.1.1)
Requirement already satisfied: pooch>=1.0 in c:\users\dell\anaconda3\lib\site-packages (from librosa) (1.4.0)
Requirement already satisfied: audioread>=2.1.9 in c:\users\dell\anaconda3\lib\site-packages (from librosa) (3.0.1)
Requirement already satisfied: msgpack>=1.0 in c:\users\dell\anaconda3\lib\site-packages (from librosa) (1.0.3)
Requirement already satisfied: soundfile>=0.12.1 in c:\users\dell\anaconda3\lib\site-packages (from librosa) (0.12.1)
Requirement already satisfied: lazy-loader>=0.1 in c:\users\dell\anaconda3\lib\site-packages (from librosa) (0.3)
Requirement already satisfied: numba>=0.51.0 in c:\users\dell\anaconda3\lib\site-packages (from librosa) (0.56.4)
Requirement already satisfied: llvmlite<0.40,>=0.39.0dev0 in c:\users\dell\anaconda3\lib\site-packages (from numba>=0.51.0->librosa) (0.39.1)
Requirement already satisfied: requests in c:\users\dell\anaconda3\lib\site-packages (from pooch>=1.0->librosa) (2.28.1)
Requirement already satisfied: appdirs in c:\users\dell\anaconda3\lib\site-packages (from pooch>=1.0->librosa) (1.4.4)
Requirement already satisfied: cffi>=1.0 in c:\users\dell\anaconda3\lib\site-packages (from soundfile>=0.12.1->librosa) (1.15.1)

Requirement already satisfied: wheel<1.0,>=0.23.0 in c:\users\dell\anaconda3\lib\site-packages (from astunparse>=1.6.0->tensorflow-intel==2.15.0->tensorflow) (0.38.4)
Requirement already satisfied: pycparser in c:\users\dell\anaconda3\lib\site-packages (from cffi>=1.0->soundfile>=0.12.1->librosa) (2.21)
Requirement already satisfied: markdown>=2.6.8 in c:\users\dell\anaconda3\lib\site-packages (from tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (3.4.1)
Requirement already satisfied: werkzeug>=1.0.1 in c:\users\dell\anaconda3\lib\site-packages (from tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (2.2.2)
Requirement already satisfied: google-auth<3,>=1.6.3 in c:\users\dell\anaconda3\lib\site-packages (from tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (2.25.2)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in c:\users\dell\anaconda3\lib\site-packages (from tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (0.7.2)
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in c:\users\dell\anaconda3\lib\site-packages (from tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (1.1.0)
Requirement already satisfied: idna<4,>=2.5 in c:\users\dell\anaconda3\lib\site-packages (from requests->pooch>=1.0->librosa) (3.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\dell\anaconda3\lib\site-packages (from requests->pooch>=1.0->librosa) (1.26.14)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\dell\anaconda3\lib\site-packages (from requests->pooch>=1.0->librosa) (2023.7.22)
Requirement already satisfied: charset-normalizer<3,>=2 in c:\users\dell\anaconda3\lib\site-packages (from requests->pooch>=1.0->librosa) (2.0.4)
Requirement already satisfied: rsa<5,>=3.1.4 in c:\users\dell\anaconda3\lib\site-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (4.9)
Requirement already satisfied: pyasn1-modules>=0.2.1 in c:\users\dell\anaconda3\lib\site-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (0.2.8)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in c:\users\dell\anaconda3\lib\site-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (5.3.2)
Requirement already satisfied: requests-oauthlib>=0.7.0 in c:\users\dell\anaconda3\lib\site-packages (from google-auth-oauthlib<2,>=0.5->tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (1.3.1)
Requirement already satisfied: MarkupSafe>=2.1.1 in c:\users\dell\anaconda3\lib\site-packages (from werkzeug>=1.0.1->tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (2.1.1)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in c:\users\dell\anaconda3\lib\site-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (0.4.8)
Requirement already satisfied: oauthlib>=3.0.0 in c:\users\dell\anaconda3\lib\site-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<2,>=0.5->tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (3.2.2)

In [3]:

```
import pandas as pd
import numpy as np
```

```
import os
import tqdm
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout
from sklearn.model_selection import train_test_split


label2int = {
    "male": 1,
    "female": 0
}


def load_data(vector_length=128):
    """A function to load gender recognition dataset from `data` folder
    After the second run, this will load from results/features.npy and results/labels.npy files
    as it is much faster!"""
    # make sure results folder exists
    if not os.path.isdir("results"):
        os.mkdir("results")
    # if features & labels already loaded individually and bundled, load them from there instead
    if os.path.isfile("results/features.npy") and os.path.isfile("results/labels.npy"):
        X = np.load("results/features.npy")
        y = np.load("results/labels.npy")
        return X, y
    # read dataframe
    df = pd.read_csv("balanced-all.csv")
    # get total samples
    n_samples = len(df)
    # get total male samples
    n_male_samples = len(df[df['gender'] == 'male'])
    # get total female samples
    n_female_samples = len(df[df['gender'] == 'female'])
    print("Total samples:", n_samples)
    print("Total male samples:", n_male_samples)
    print("Total female samples:", n_female_samples)
    # initialize an empty array for all audio features
    X = np.zeros((n_samples, vector_length))
    # initialize an empty array for all audio labels (1 for male and 0 for female)
    y = np.zeros((n_samples, 1))
    for i, (filename, gender) in tqdm.tqdm(enumerate(zip(df['filename'], df['gender'])), "Loading
data", total=n_samples):
        features = np.load(filename)
        X[i] = features
        y[i] = label2int[gender]
    # save the audio features and labels into files
```

```
    # so we won't load each one of them next run
    np.save("results/features", X)
    np.save("results/labels", y)
    return X, y


def split_data(X, y, test_size=0.1, valid_size=0.1):
    # split training set and testing set
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=7)
    # split training set and validation set
    X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=valid_size,
random_state=7)
    # return a dictionary of values
    return {
        "X_train": X_train,
        "X_valid": X_valid,
        "X_test": X_test,
        "y_train": y_train,
        "y_valid": y_valid,
        "y_test": y_test
    }


def create_model(vector_length=128):
    """5 hidden dense layers from 256 units to 64, not the best model, but not bad."""
    model = Sequential()
    model.add(Dense(256, input_shape=(vector_length,)))
    model.add(Dropout(0.3))
    model.add(Dense(256, activation="relu"))
    model.add(Dropout(0.3))
    model.add(Dense(128, activation="relu"))
    model.add(Dropout(0.3))
    model.add(Dense(128, activation="relu"))
    model.add(Dropout(0.3))
    model.add(Dense(64, activation="relu"))
    model.add(Dropout(0.3))
    # one output neuron with sigmoid activation function, 0 means female, 1 means male
    model.add(Dense(1, activation="sigmoid"))
    # using binary crossentropy as it's male/female classification (binary)
    model.compile(loss="binary_crossentropy", metrics=["accuracy"], optimizer="adam")
    # print summary of the model
    model.summary()
    return model
```

```
WARNING:tensorflow:From C:\Users\Dell\anaconda3\lib\site-
packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is
deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

In [4]:

```
import os
from tensorflow.keras.callbacks import ModelCheckpoint, TensorBoard, EarlyStopping

from utils import load_data, split_data, create_model

# load the dataset
X, y = load_data()
# split the data into training, validation and testing sets
data = split_data(X, y, test_size=0.1, valid_size=0.1)
# construct the model
model = create_model()

# use tensorboard to view metrics
tensorboard = TensorBoard(log_dir="logs")
# define early stopping to stop training after 5 epochs of not improving
early_stopping = EarlyStopping(mode="min", patience=5, restore_best_weights=True)

batch_size = 64
epochs = 100

# train the model using the training set and validating using validation set
model.fit(data["X_train"], data["y_train"], epochs=epochs, batch_size=batch_size,
validation_data=(data["X_valid"], data["y_valid"]),
        callbacks=[tensorboard, early_stopping])

# save the model to a file
model.save("results/model.h5")

# evaluating the model using the testing set
print(f"Evaluating the model using {len(data['X_test'])} samples...")
loss, accuracy = model.evaluate(data["X_test"], data["y_test"], verbose=0)
print(f"Loss: {loss:.4f}")
print(f"Accuracy: {accuracy*100:.2f}%")
```

```
WARNING:tensorflow:From C:\Users\Dell\anaconda3\lib\site-
packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated. Please use
tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Users\Dell\anaconda3\lib\site-
packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is deprecated. Please
use tf.compat.v1.train.Optimizer instead.

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

dense (Dense)	(None, 256)	33024
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 256)	65792
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32896
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 128)	16512
dropout_3 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 64)	8256
dropout_4 (Dropout)	(None, 64)	0
dense_5 (Dense)	(None, 1)	65

Total params: 156545 (611.50 KB)
Trainable params: 156545 (611.50 KB)
Non-trainable params: 0 (0.00 Byte)

Epoch 1/100
WARNING:tensorflow:From C:\Users\Dell\anaconda3\lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\Dell\anaconda3\lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

848/848 [==============================] - 10s 7ms/step - loss: 0.5754 - accuracy: 0.7635 - val_loss: 0.3938 - val_accuracy: 0.8437
Epoch 2/100
848/848 [==============================] - 8s 9ms/step - loss: 0.4217 - accuracy: 0.8318 - val_loss: 0.3592 - val_accuracy: 0.8622
Epoch 3/100
848/848 [==============================] - 6s 7ms/step - loss: 0.3800 - accuracy: 0.8503 - val_loss: 0.3142 - val_accuracy: 0.8757

```
Epoch 4/100
848/848 [==============================] - 9s 11ms/step - loss: 0.3616 - accuracy:
0.8606 - val_loss: 0.3017 - val_accuracy: 0.8800
Epoch 5/100
848/848 [==============================] - 9s 11ms/step - loss: 0.3488 - accuracy:
0.8664 - val_loss: 0.3141 - val_accuracy: 0.8787
Epoch 6/100
848/848 [==============================] - 8s 9ms/step - loss: 0.3408 - accuracy:
0.8691 - val_loss: 0.2885 - val_accuracy: 0.8817
Epoch 7/100
848/848 [==============================] - 6s 8ms/step - loss: 0.3287 - accuracy:
0.8747 - val_loss: 0.2777 - val_accuracy: 0.8951
Epoch 8/100
848/848 [==============================] - 5s 6ms/step - loss: 0.3204 - accuracy:
0.8797 - val_loss: 0.2829 - val_accuracy: 0.8923
Epoch 9/100
848/848 [==============================] - 5s 6ms/step - loss: 0.3218 - accuracy:
0.8769 - val_loss: 0.2653 - val_accuracy: 0.8959
Epoch 10/100
848/848 [==============================] - 6s 7ms/step - loss: 0.3157 - accuracy:
0.8791 - val_loss: 0.2605 - val_accuracy: 0.8976
Epoch 11/100
848/848 [==============================] - 5s 6ms/step - loss: 0.3112 - accuracy:
0.8837 - val_loss: 0.2613 - val_accuracy: 0.9019
Epoch 12/100
848/848 [==============================] - 7s 8ms/step - loss: 0.2967 - accuracy:
0.8886 - val_loss: 0.2517 - val_accuracy: 0.9036
Epoch 13/100
848/848 [==============================] - 6s 7ms/step - loss: 0.2955 - accuracy:
0.8876 - val_loss: 0.2522 - val_accuracy: 0.9029
Epoch 14/100
848/848 [==============================] - 7s 9ms/step - loss: 0.2920 - accuracy:
0.8896 - val_loss: 0.2663 - val_accuracy: 0.9026
Epoch 15/100
848/848 [==============================] - 8s 9ms/step - loss: 0.2871 - accuracy:
0.8918 - val_loss: 0.2597 - val_accuracy: 0.9006
Epoch 16/100
848/848 [==============================] - 6s 7ms/step - loss: 0.2894 - accuracy:
0.8907 - val_loss: 0.2514 - val_accuracy: 0.9077
Epoch 17/100
848/848 [==============================] - 10s 12ms/step - loss: 0.2822 - accuracy:
0.8941 - val_loss: 0.2457 - val_accuracy: 0.9082
Epoch 18/100
848/848 [==============================] - 19s 22ms/step - loss: 0.2801 - accuracy:
0.8946 - val_loss: 0.2475 - val_accuracy: 0.9071
Epoch 19/100
```

848/848 [==============================] - 14s 17ms/step - loss: 0.2774 - accuracy: 0.8968 - val_loss: 0.2387 - val_accuracy: 0.9066
Epoch 20/100
848/848 [==============================] - 9s 10ms/step - loss: 0.2767 - accuracy: 0.8971 - val_loss: 0.2467 - val_accuracy: 0.9097
Epoch 21/100
848/848 [==============================] - 6s 7ms/step - loss: 0.2721 - accuracy: 0.8983 - val_loss: 0.2418 - val_accuracy: 0.9102
Epoch 22/100
848/848 [==============================] - 7s 9ms/step - loss: 0.2771 - accuracy: 0.8976 - val_loss: 0.2498 - val_accuracy: 0.9054
Epoch 23/100
848/848 [==============================] - 6s 7ms/step - loss: 0.2780 - accuracy: 0.8963 - val_loss: 0.2384 - val_accuracy: 0.9119
Epoch 24/100
848/848 [==============================] - 6s 7ms/step - loss: 0.2731 - accuracy: 0.8977 - val_loss: 0.2394 - val_accuracy: 0.9057
Epoch 25/100
848/848 [==============================] - 8s 9ms/step - loss: 0.2705 - accuracy: 0.8990 - val_loss: 0.2325 - val_accuracy: 0.9163
Epoch 26/100
848/848 [==============================] - 6s 8ms/step - loss: 0.2717 - accuracy: 0.8987 - val_loss: 0.2366 - val_accuracy: 0.9124
Epoch 27/100
848/848 [==============================] - 7s 8ms/step - loss: 0.2665 - accuracy: 0.8992 - val_loss: 0.2271 - val_accuracy: 0.9097
Epoch 28/100
848/848 [==============================] - 7s 8ms/step - loss: 0.2658 - accuracy: 0.9011 - val_loss: 0.2313 - val_accuracy: 0.9132
Epoch 29/100
848/848 [==============================] - 7s 8ms/step - loss: 0.2650 - accuracy: 0.9021 - val_loss: 0.2293 - val_accuracy: 0.9147
Epoch 30/100
848/848 [==============================] - 7s 8ms/step - loss: 0.2658 - accuracy: 0.9005 - val_loss: 0.2321 - val_accuracy: 0.9144
Epoch 31/100
848/848 [==============================] - 7s 8ms/step - loss: 0.2608 - accuracy: 0.9017 - val_loss: 0.2260 - val_accuracy: 0.9182
Epoch 32/100
848/848 [==============================] - 8s 9ms/step - loss: 0.2645 - accuracy: 0.9021 - val_loss: 0.2311 - val_accuracy: 0.9132
Epoch 33/100
848/848 [==============================] - 7s 8ms/step - loss: 0.2610 - accuracy: 0.9028 - val_loss: 0.2348 - val_accuracy: 0.9115
Epoch 34/100

```
848/848 [==============================] - 10s 12ms/step - loss: 0.2608 - accuracy:
0.9038 - val_loss: 0.2371 - val_accuracy: 0.9130
Epoch 35/100
848/848 [==============================] - 8s 10ms/step - loss: 0.2577 - accuracy:
0.9038 - val_loss: 0.2435 - val_accuracy: 0.9105
Epoch 36/100
848/848 [==============================] - 8s 9ms/step - loss: 0.2618 - accuracy:
0.9012 - val_loss: 0.2287 - val_accuracy: 0.9170
Evaluating the model using 6694 samples...
C:\Users\Dell\anaconda3\lib\site-packages\keras\src\engine\training.py:3103: UserWarning: You
are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy.
We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
Loss: 0.2208
Accuracy: 91.75%
```

In [20]:

```
import pyaudio
import os
import wave
import librosa
import numpy as np
from sys import byteorder
from array import array
from struct import pack


THRESHOLD = 500
CHUNK_SIZE = 1024
FORMAT = pyaudio.paInt16
RATE = 16000

SILENCE = 30

def is_silent(snd_data):
    "Returns 'True' if below the 'silent' threshold"
    return max(snd_data) < THRESHOLD

def normalize(snd_data):
    "Average the volume out"
    MAXIMUM = 16384
    times = float(MAXIMUM)/max(abs(i) for i in snd_data)

    r = array('h')
    for i in snd_data:
        r.append(int(i*times))
    return r
```

```
def trim(snd_data):
    "Trim the blank spots at the start and end"
    def _trim(snd_data):
        snd_started = False
        r = array('h')

        for i in snd_data:
            if not snd_started and abs(i)>THRESHOLD:
                snd_started = True
                r.append(i)

            elif snd_started:
                r.append(i)
        return r

    # Trim to the left
    snd_data = _trim(snd_data)

    # Trim to the right
    snd_data.reverse()
    snd_data = _trim(snd_data)
    snd_data.reverse()
    return snd_data

def add_silence(snd_data, seconds):
    "Add silence to the start and end of 'snd_data' of length 'seconds' (float)"
    r = array('h', [0 for i in range(int(seconds*RATE))])
    r.extend(snd_data)
    r.extend([0 for i in range(int(seconds*RATE))])
    return r

def record():
    """
    Record a word or words from the microphone and
    return the data as an array of signed shorts.
    Normalizes the audio, trims silence from the
    start and end, and pads with 0.5 seconds of
    blank sound to make sure VLC et al can play
    it without getting chopped off.
    """
    p = pyaudio.PyAudio()
    stream = p.open(format=FORMAT, channels=1, rate=RATE,
        input=True, output=True,
        frames_per_buffer=CHUNK_SIZE)
```

```
    num_silent = 0
    snd_started = False

    r = array('h')

    while 1:
        # little endian, signed short
        snd_data = array('h', stream.read(CHUNK_SIZE))
        if byteorder == 'big':
            snd_data.byteswap()
        r.extend(snd_data)

        silent = is_silent(snd_data)

        if silent and snd_started:
            num_silent += 1
        elif not silent and not snd_started:
            snd_started = True

        if snd_started and num_silent > SILENCE:
            break

    sample_width = p.get_sample_size(FORMAT)
    stream.stop_stream()
    stream.close()
    p.terminate()

    r = normalize(r)
    r = trim(r)
    r = add_silence(r, 0.5)
    return sample_width, r

def record_to_file(path):
    "Records from the microphone and outputs the resulting data to 'path'"
    sample_width, data = record()
    data = pack('<' + ('h'*len(data)), *data)

    wf = wave.open(path, 'wb')
    wf.setnchannels(1)
    wf.setsampwidth(sample_width)
    wf.setframerate(RATE)
    wf.writeframes(data)
    wf.close()
```

```
def extract_feature(file_name, **kwargs):
    """
    Extract feature from audio file `file_name`
        Features supported:
            - MFCC (mfcc)
            - Chroma (chroma)
            - MEL Spectrogram Frequency (mel)
            - Contrast (contrast)
            - Tonnetz (tonnetz)
        e.g:
        `features = extract_feature(path, mel=True, mfcc=True)`
    """
    mfcc = kwargs.get("mfcc")
    chroma = kwargs.get("chroma")
    mel = kwargs.get("mel")
    contrast = kwargs.get("contrast")
    tonnetz = kwargs.get("tonnetz")
    file_path = "C:/Users/Dell/Downloads/gender-recognition-by-voice-master (1)/gender-
recognition-by-voice-master/26-496-0005.wav"
    X,  sample_rate = librosa.core.load(file_path)

    if chroma or contrast:
        stft = np.abs(librosa.stft(X))
    result = np.array([])
    if mfcc:
        mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40).T, axis=0)
        result = np.hstack((result, mfccs))
    if chroma:
        chroma = np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).T,axis=0)
        result = np.hstack((result, chroma))
    if mel:
        mel = np.mean(librosa.feature.melspectrogram(y=X, sr=sample_rate).T, axis=0)
        result = np.hstack((result, mel))
    if contrast:
        contrast = np.mean(librosa.feature.spectral_contrast(S=stft, sr=sample_rate).T,axis=0)
        result = np.hstack((result, contrast))
    if tonnetz:
        tonnetz = np.mean(librosa.feature.tonnetz(y=librosa.effects.harmonic(X),
sr=sample_rate).T,axis=0)
        result = np.hstack((result, tonnetz))
    return result


if __name__ == "__main__":
    # load the saved model (after training)
    # model = pickle.load(open("result/mlp_classifier.model", "rb"))
```

```
from utils import load_data, split_data, create_model
import argparse
parser = argparse.ArgumentParser(description="""Gender recognition script, this will load the model you trained,
                                and perform inference on a sample you provide (either using your voice or a file)""")
parser.add_argument("-f", "--file", help="The path to the file, preferred to be in WAV format")
args = parser.parse_args()
file = args.file
# construct the model
model = create_model()
# load the saved/trained weights
model.load_weights("results/model.h5")
if not file or not os.path.isfile(file):
    # if file not provided, or it doesn't exist, use your voice
    print("Please talk")
    # put the file name here
    file = r"C:\path\to\save\test.wav"
    # record the file (start talking)
    record_to_file(file)
# extract features and reshape it
features = extract_feature(file, mel=True).reshape(1, -1)
# predict the gender!
male_prob = model.predict(features)[0][0]
female_prob = 1 - male_prob
gender = "male" if male_prob > female_prob else "female"
# show the result!
print("Result:", gender)
print(f"Probabilities:     Male: {male_prob*100:.2f}%    Female: {female_prob*100:.2f}%")
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_60 (Dense)             (None, 256)               33024

dropout_50 (Dropout)         (None, 256)               0

dense_61 (Dense)             (None, 256)               65792

dropout_51 (Dropout)         (None, 256)               0

dense_62 (Dense)             (None, 128)               32896

dropout_52 (Dropout)         (None, 128)               0
```

```
 dense_63 (Dense)          (None, 128)            16512

 dropout_53 (Dropout)      (None, 128)            0

 dense_64 (Dense)          (None, 64)             8256

 dropout_54 (Dropout)      (None, 64)             0

 dense_65 (Dense)          (None, 1)              65

=================================================================
Total params: 156545 (611.50 KB)
Trainable params: 156545 (611.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
1/1 [==============================] - 0s 418ms/step
Result: male
Probabilities:     Male: 95.74%    Female: 4.26%
```

In [21]:

```
import glob
import os
import pandas as pd
import numpy as np
import shutil
import librosa
from tqdm import tqdm


def extract_feature(file_name, **kwargs):
    """
    Extract feature from audio file `file_name`
        Features supported:
            - MFCC (mfcc)
            - Chroma (chroma)
            - MEL Spectrogram Frequency (mel)
            - Contrast (contrast)
            - Tonnetz (tonnetz)
        e.g:
        `features = extract_feature(path, mel=True, mfcc=True)`
    """
    mfcc = kwargs.get("mfcc")
    chroma = kwargs.get("chroma")
    mel = kwargs.get("mel")
    contrast = kwargs.get("contrast")
    tonnetz = kwargs.get("tonnetz")
    X, sample_rate = librosa.core.load(file_name)
```

```
    if chroma or contrast:
        stft = np.abs(librosa.stft(X))
    result = np.array([])
    if mfcc:
        mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40).T, axis=0)
        result = np.hstack((result, mfccs))
    if chroma:
        chroma = np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).T,axis=0)
        result = np.hstack((result, chroma))
    if mel:
        mel = np.mean(librosa.feature.melspectrogram(X, sr=sample_rate).T,axis=0)
        result = np.hstack((result, mel))
    if contrast:
        contrast = np.mean(librosa.feature.spectral_contrast(S=stft, sr=sample_rate).T,axis=0)
        result = np.hstack((result, contrast))
    if tonnetz:
        tonnetz = np.mean(librosa.feature.tonnetz(y=librosa.effects.harmonic(X),
sr=sample_rate).T,axis=0)
        result = np.hstack((result, tonnetz))
    return result

dirname = "data"

if not os.path.isdir(dirname):
    os.mkdir(dirname)


csv_files = glob.glob("*.csv")

for j, csv_file in enumerate(csv_files):
    print("[+] Preprocessing", csv_file)
    df = pd.read_csv(csv_file)
    # only take filename and gender columns
    new_df = df[["filename", "gender"]]
    print("Previously:", len(new_df), "rows")
    # take only male & female genders (i.e droping NaNs & 'other' gender)
    new_df = new_df[np.logical_or(new_df['gender'] == 'female', new_df['gender'] == 'male')]
    print("Now:", len(new_df), "rows")
    new_csv_file = os.path.join(dirname, csv_file)
    # save new preprocessed CSV
    new_df.to_csv(new_csv_file, index=False)
    # get the folder name
    folder_name, _ = csv_file.split(".")
    audio_files = glob.glob(f"{folder_name}/{folder_name}/*")
    all_audio_filenames = set(new_df["filename"])
```

```
    for i, audio_file in tqdm(list(enumerate(audio_files)), f"Extracting features of
{folder_name}"):
        splited = os.path.split(audio_file)
        # audio_filename = os.path.join(os.path.split(splited[0])[-1], splited[-1])
        audio_filename = f"{os.path.split(splited[0])[-1]}/{splited[-1]}"
        # print("audio_filename:", audio_filename)
        if audio_filename in all_audio_filenames:
            # print("Copyying", audio_filename, "...")
            src_path = f"{folder_name}/{audio_filename}"
            target_path = f"{dirname}/{audio_filename}"
            #create that folder if it doesn't exist
            if not os.path.isdir(os.path.dirname(target_path)):
                os.mkdir(os.path.dirname(target_path))
            features = extract_feature(src_path, mel=True)
            target_filename = target_path.split(".")[0]
            np.save(target_filename, features)
            # shutil.copyfile(src_path, target_path)
```

[+] Preprocessing balanced-all.csv
Previously: 66938 rows
Now: 66938 rows

In [40]:

```
!python test.py --file "C:/Users/Dell/Downloads/gender-recognition-by-voice-master (1)/gender-
recognition-by-voice-master/22-121148-0001.wav"
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 256)	33024
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 256)	65792
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32896
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 128)	16512
dropout_3 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 64)	8256
dropout_4 (Dropout)	(None, 64)	0

```
 dense_5 (Dense)          (None, 1)             65

=================================================================
Total params: 156545 (611.50 KB)
Trainable params: 156545 (611.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________

1/1 [==============================] - ETA: 0s
1/1 [==============================] - 0s 470ms/step
Result: female
Probabilities:     Male: 7.31%    Female: 92.69%
```

Future Improvements

Model Enhancement

Transfer Learning Updates

The model is subject to continuous enhancement through periodic updates, leveraging the latest advancements in transfer learning. This involves retraining specific layers or fine-tuning the entire model with newly available data, ensuring that the model remains adaptive to evolving voice patterns.

Online Learning

Online learning mechanisms are explored to enable the model to adapt to incremental changes in the data distribution. This allows the system to continuously improve its performance without the need for frequent retraining.

Hyperparameter Optimization

Ongoing efforts are made to refine model hyperparameters based on observed performance and emerging best practices. Techniques such as Bayesian optimization are employed to efficiently search for optimal configurations.

Dataset Expansion

Continuous Data Collection

The dataset undergoes continuous expansion through the inclusion of new voice samples. This involves establishing mechanisms for ongoing data collection, ensuring that the model is exposed to a diverse and up-to-date set of speakers.

Domain-specific Augmentation

To enhance the model's robustness in specific domains or applications, domain-specific data augmentation techniques are explored. This may involve collecting additional samples from targeted user groups or environments.

Ethical Considerations

Expansion efforts prioritize ethical considerations, including consent, privacy, and the avoidance of biases. Continuous engagement with diverse communities helps ensure that the dataset remains representative and unbiased.

Real-time Processing

Edge Computing Integration

Real-time processing capabilities are extended through the integration of edge computing solutions. This allows the gender recognition system to perform inference directly on edge devices, reducing latency and enabling applications with low-latency requirements.

Dynamic Resource Allocation

Efforts are made to implement dynamic resource allocation strategies, adapting processing requirements based on the real-time demands of the application. This ensures optimal performance in varying computational environments.

Streaming Architecture

The system architecture is optimized for streaming data, allowing seamless integration with applications that process voice input in real-time. This includes the design of efficient data pipelines and low-latency processing components.

Conclusion

Summary

In summary, the gender recognition system presented in this documentation is a comprehensive and adaptable solution designed to accurately identify the gender of a speaker based on voice characteristics. The system incorporates advanced techniques in data collection, preprocessing, feature extraction, model training, and post-processing. It embraces ethical considerations, addresses biases, and aims for transparency and interpretability in its decision-making process.

The model undergoes continuous enhancement through updates, hyperparameter optimization, and online learning mechanisms. Dataset expansion efforts prioritize diversity, ethics, and domain-specific considerations. Real-time processing capabilities are extended through edge computing integration, dynamic resource allocation, and streaming architecture.