Title: Human Voice Classification and Clustering

By,

Harish K M

Abstract:

This project presents a machine learning-based system to classify and cluster human voice samples using extracted audio features. The aim is to differentiate voices based on gender and discover natural groupings using clustering algorithms. The workflow includes data preprocessing, exploratory analysis, model training (classification and clustering), and evaluation. The final solution is deployed as a Streamlit application, allowing users to input features and receive predictions. This system supports use cases such as speaker identification, gender recognition, and speech analytics.

Introduction:

Voice recognition and classification are pivotal in domains such as security, customer service, accessibility, and smart devices. Human voices carry rich acoustic features that can be used to identify characteristics like gender, speaker identity, or mood. In this project, we develop a machine learning-based system that classifies voices by gender and clusters them into groups based on audio features.

The project leverages a structured dataset containing spectral, pitch, energy, and MFCC features from voice recordings. We apply supervised and unsupervised learning techniques to build robust models and evaluate their performance using standard metrics. The ultimate goal is to deliver a working Streamlit application for real-time voice feature prediction.

Dataset Description:

The dataset used in this project contains extracted acoustic features from human voice recordings, labeled for gender classification. It is structured in a tabular format, where each row represents a unique voice sample and each column represents a specific acoustic feature.

Feature Categories:

The dataset contains feature-engineered numerical values representing characteristics of the voice in the spectral, pitch, energy, and MFCC domains.

Feature Type	Description
Spectral Features	**Describe frequency distribution and tonal quality**
`mean_spectral_centroid`, `std_spectral_centroid`	Indicates brightness and its variability

`mean_spectral_bandwidth, std_spectral_bandwidth`	Frequency spread and variability
`mean_spectral_contrast`	Tonal contrast
`mean_spectral_flatness`	Noisiness of the signal
`mean_spectral_rolloff`	Sharpness or spectral energy drop-off

Feature Type	**Description**
Energy Features	**Describe loudness and signal strength**
`zero_crossing_rate`	Percussiveness or noisiness
`rms_energy`	Root Mean Square energy (loudness)
`log_energy`	Log-compressed energy
`energy_entropy`	Randomness in energy

Feature Type	**Description**
Pitch Features	**Voice pitch characteristics**
`mean_pitch, min_pitch, max_pitch, std_pitch`	Pitch levels and variation

Feature Type	**Description**
Spectral Shape Features	**Distribution properties**
`spectral_skew`	Asymmetry of the spectrum
`spectral_kurtosis`	Peakiness of the spectrum

Feature Type	**Description**
MFCC Features	**Describe timbre (voice quality)**
`mfcc_1_mean to mfcc_13_mean`	Mean values of 13 MFCCs
`mfcc_1_std to mfcc_13_std`	Standard deviations of MFCCs

Target Feature:

Label: The gender class

- 0 → Female
- 1 → Male

Data Preprocessing:

Before training models or performing clustering, the dataset was carefully preprocessed to ensure it was clean, consistent, and ready for machine learning workflows.

1. **Data Inspection:**
    - Checked the dataset for null values, duplicates, or inconsistent entries.
    - Result: No missing values or duplicates found, so no imputation was necessary.

    Code:

    ```
    df.info() # Check for basic info

    df.describe() # Summary statistics

    df.isnull().sum().sort_values(ascending=False) # Check for missing values
    ```

2. **Data Cleaning:**
    - Verified data types of all columns all features were already in numeric format which is suitable for ML algorithms.
    - The label column was confirmed to be binary (0 for female, 1 for male), requiring no further encoding.
3. **Feature Scaling (Normalization):**

    To ensure uniform scale and improve model performance, all numeric features were standardized using:

    - **Scaler Used**: `StandardScaler` from `sklearn.preprocessing`
    - **Purpose**: This transformation ensures each feature has a mean of 0 and standard deviation of 1.

    Why StandardScaler?

    - Essential for models like SVM, K-Means, and Neural Networks that are sensitive to feature magnitudes.
    - Maintains the shape of the original distribution but adjusts scale.

    Code:

    ```
    from sklearn.preprocessing import StandardScaler

    scaler = StandardScaler()

    X_scaled = scaler.fit_transform(X)

    X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)
    ```

4. **Feature and Label Separation:**

    - Features (`X`) were separated from the target variable (`y`) for both clustering and classification.
    - For unsupervised clustering, only `X_scaled` was used.

Code:

```
X = df.drop('label', axis=1)

y = df['label']
```

5. **Train-Test Split for Classification:**

    - Used `train_test_split` with stratification to maintain class balance (equal gender distribution) in training and testing datasets.

    Code:

```
X_train, X_test, y_train, y_test =
train_test_split(X_scaled_df, y, test_size=0.2,
random_state=42, stratify=y)
```

Feature Engineering & Selection:

To enhance model performance and reduce computational complexity, feature engineering and selection techniques were applied. This step focused on identifying and retaining the most informative features while reducing noise and redundancy in the dataset.

1. **Feature Engineering:**

    The dataset already consisted of extracted audio features, including spectral, pitch, energy, and MFCC-based characteristics. These features captured rich information about each voice sample. Although no new features were derived manually, the following processing was applied:

    - **Standardization**: All features were scaled using `StandardScaler` to ensure uniform contribution across models.
    - **Label Handling**: The label column was used only for supervised classification tasks and excluded during unsupervised clustering.

2. **Feature Selection:**

    To reduce dimensionality and retain only the most relevant features for classification, a univariate statistical feature selection technique was applied:

    - **Method**: `SelectKBest` from `sklearn.feature_selection.`
    - **Scoring Function**: `mutual_info_classif` (measures mutual dependence between features and target)
    - **Number of Features Selected**: Top 15

    Code:

```
from sklearn.feature_selection import SelectKBest,
mutual_info_classif

selector =
SelectKBest(score_func=mutual_info_classif, k=15)

X_selected = selector.fit_transform(X_train,
y_train)

selected_features =
X.columns[selector.get_support()]
```

Benefits of Feature Selection:

- **Improved model performance**: By removing irrelevant/noisy features
- **Faster training times**: Fewer dimensions reduce complexity
- **Better generalization**: Avoids overfitting on high-dimensional data

Exploratory Feature Questions & Observations:

1. Are there significant differences in spectral centroid between the two genders?

   Code:

   ```
   sns.boxplot(data=df, x='Gender',
   y='mean_spectral_centroid')
   ```

   Interpretation:

   - Female voices show higher spectral centroid values than male voices.
   - This indicates brighter or sharper vocal characteristics in females due to higher frequency components.
   - There is clear separation in distribution, suggesting it is a useful feature for classification.

2. Is there a strong relationship between spectral bandwidth and pitch variability?

   Code:

   ```
   sns.scatterplot(data=df, x='mean_spectral_bandwidth',
   y='std_pitch', hue='Gender', alpha=0.6)
   ```

   Interpretation:

   - Weak to moderate positive correlation observed.
   - No strong linear trend, but some gender-specific grouping appears.
   - These features may contribute to clustering even if not highly correlated.

3. Do male and female voices differ in mean pitch?

   Code:

   ```
   sns.histplot(data=df, x='mean_pitch', hue='Gender',
   kde=True, element="step")
   ```

   Interpretation:

   - Clear bimodal distribution.
   - Female voices concentrate at higher pitch frequencies, while males skew lower.
   - Highly discriminative feature for gender classification.

4. Is spectral contrast higher in male or female voices?

   Code:

   ```
   sns.violinplot(data=df, x='Gender',
   y='mean_spectral_contrast')
   ```

   Interpretation:

   - Slightly higher average spectral contrast in female voices.
   - However, there is considerable overlap, so this may be supplementary rather than a standalone discriminator.

5. How does zero-crossing rate vary across genders?

   Code:

   ```
   sns.boxplot(data=df, x='Gender', y='zero_crossing_rate')
   ```

   Interpretation:

   - Female voices have higher zero-crossing rates, aligning with the presence of more high-frequency components.
   - Useful as a supporting feature in classification.

6. Is pitch standard deviation (pitch variability) higher for one gender?

   Code:

   ```
   sns.histplot(data=df, x='std_pitch', hue='Gender',
   kde=True, element="step")
   ```

   Interpretation:

   - Female voices show higher pitch variability.

- Indicates more expressive or modulated pitch patterns.
- Adds depth to models that benefit from voice modulation characteristics.

7. What is the correlation between mean_pitch and spectral_rolloff?

   Code:

   ```
   sns.scatterplot(data=df, x='mean_pitch',
   y='mean_spectral_rolloff', hue='Gender', alpha=0.6)
   ```

   Interpretation:

   - Moderate correlation observed (slightly positive).
   - Differences between genders are visible, but not linearly separable.
   - Jointly these features may enhance model decisions.

8. Does spectral skew or kurtosis reveal voice timbre differences?

   Code:

   ```
   sns.boxplot(data=df, x='Gender', y='spectral_kurtosis')
   ```

   Interpretation:

   - Slight differences in mean values, with female voices tending to have higher skew and kurtosis.
   - Captures asymmetry and peaking in frequency distributions.
   - More useful in combination than isolation.

9. Are there clusters of voices based on spectral_centroid and rms_energy?

   Code:

   ```
   sns.boxplot(data=df, x='Gender', y='spectral_skew')
   ```

   Interpretation:

   - Some natural grouping visible across gender lines.
   - Male and female voices cluster in different regions — a good sign for unsupervised clustering methods like KMeans or DBSCAN.

10. Is there redundancy among spectral features (e.g., centroid vs rolloff)?

    Code:

    ```
    sns.heatmap(corr, cmap='coolwarm', annot=False, center=0)
    ```

    Interpretation:

    - High correlations between spectral features like centroid, bandwidth, rolloff.

- Indicates redundancy, justifying feature selection or dimensionality reduction (e.g., via PCA or SelectKBest).

PCA:

Why Use PCA:

i. **Dimensionality Reduction:**

PCA reduces the number of features while retaining as much variance (information) as possible.

- Your dataset may have 30+ features, which is hard to visualize or cluster intuitively.
- PCA projects this high-dimensional data into 2D or 3D (for visualization), capturing most of the data's structure.

ii. **Improves Clustering & Visualization:**

- In unsupervised learning (like K-Means, DBSCAN), many features may introduce noise or redundancy.
- PCA simplifies the data structure and exposes clearer patterns or natural groupings.
- Helps plot data in 2D with better separation of clusters, even if clusters aren't perfectly circular.

iii. **Handles Multicollinearity:**

- Many features (e.g., spectral centroid, rolloff, bandwidth) are correlated.
- PCA transforms correlated features into a set of uncorrelated principal components, removing redundancy.
- This often stabilizes models and avoids overfitting in high-dimensional space.

When to Use PCA:

- Clustering Visualization: To reduce high-dimensional data to 2D or 3D for plotting cluster structures.
- Too Many Features: If the feature count is large (e.g., MFCCs + spectral + pitch), PCA simplifies the data.
- Correlated Features: To remove multicollinearity and create orthogonal (independent) inputs.
- Speed Up Modeling: PCA reduces input dimensions, which can speed up algorithms like K-Means, DBSCAN.
- Noise Reduction: To remove less informative components, retaining only signal-dominant directions.
- Feature Engineering: Sometimes the first few components themselves serve as new features.

Code:

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2) # reduce the data to 2 principal components.

X_pca = pca.fit_transform(X_scaled)
```

Clustering:

Clustering is an unsupervised machine learning technique that groups similar data points into clusters based on patterns or similarity.

Points in the same cluster are more similar to each other than to points in other clusters. Ex: Grouping voices with similar spectral features.

Purpose:

- Discover hidden patterns or structure in unlabeled data.
- Group similar items together without knowing the output labels.
- Simplify and summarize large datasets.
- Detect anomalies (e.g., outliers that don't belong to any cluster).
- Preprocess or reduce dimensions for other ML models.

When to use:

- You have no target/output labels (i.e., it's unsupervised).
- You want to explore structure in your dataset.
- You can do: Customer segmentation

Types of Clustering Algorithms:

1. **Partition-Based Clustering:**
    - **Algorithm**: K-Means
    - Divides data into k clusters.
    - Each point belongs to the cluster with the nearest centroid.
2. **Density-Based Clustering:**
    - **Algorithm**: DBSCAN (Density-Based Spatial Clustering of Applications with Noise)
    - Groups points that are densely packed and identifies outliers.
3. **Hierarchical Clustering:**
    - **Algorithm**: Agglomerative (bottom-up) or Divisive (top-down)

- Builds a tree (dendrogram) of clusters.

4. **Model-Based Clustering:**
    - **Algorithm**: Gaussian Mixture Models (GMM)
    - Assumes data is generated from a mixture of several Gaussian distributions.

K-Means Clustering:

K-Means is a partition-based clustering algorithm that divides a dataset into K distinct clusters based on similarity. It assigns each data point to the nearest cluster centroid, then updates centroids until the groupings no longer change (converge). "K" is the number of clusters you define beforehand.

Purpose:

The goal of K-Means is to:

- Group similar data points together.
- Minimize intra-cluster variance (i.e., make data points in each cluster as close as possible).
- Find natural structure in unlabeled data.

In your project, it's used to group voice samples based on audio features, potentially separating genders.

When to Use:

Use K-Means when:

- You want to segment or group data without labels.
- You know (or can estimate) the number of clusters (K) in advance.
- Your clusters are roughly spherical or evenly sized.
- You're working with numerical features (like MFCCs, pitch, spectral data).

How K-Means Works (Step-by-step):

- Initialize K cluster centers (randomly or using K-Means++).
- Assign each data point to the nearest centroid.
- Update each centroid to be the mean of the assigned points.
- Repeat steps 2–3 until convergence (i.e., no change in assignments or centroids).

Why Use K-Means:

- Fast and scalable (good for large datasets).
- Easy to implement and interpret.
- Works well when clusters are compact and well-separated.

Limitation:

- You must specify K beforehand.
- Doesn't handle outliers or noise well.
- Assumes spherical and equally sized clusters.
- Sensitive to initial centroid positions (can affect results).

Code:

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2, random_state=42) # n_clusters – No of clusters
```

DBSCAN Clustering:

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm. It groups together data points that are closely packed (dense regions) and marks points in low-density regions as outliers (noise). It does not require you to predefine the number of clusters.

Purpose:

The purpose of DBSCAN is to:

- Discover clusters of arbitrary shapes based on data density.
- Identify noise/outliers in data.
- Group data without knowing K (number of clusters) in advance.
- In project, explore if voice features form natural dense groupings (e.g., gender-based).

When to Use:

- Your data has non-spherical or irregularly shaped clusters.
- You don't know the number of clusters (K).
- You want to detect noise or outliers.
- You need a clustering method that works well with varying cluster densities.

How DBSCAN Works (Step-by-step):

1. **Pick two parameters:**
    - eps: Radius to search for neighboring points.
    - min_samples: Minimum points to form a dense region.
2. **For each point:**
    - If it has at least min_samples points within eps, it's a core point.
    - Points within eps of a core point are density-reachable.
    - Points not reachable from any core point are marked as outliers/noise.
3. Group all density-connected points into a cluster.

Why Use DBSCAN:

- Automatically detects number of clusters.
- Can find arbitrarily shaped clusters.
- Handles noise/outliers well.
- No need to specify K.

Limitation:

- Performance depends heavily on choosing the right eps and min_samples.
- Struggles with varying density across clusters.
- Slower for high-dimensional data (like many audio features).

Code:

```
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=2.0, min_samples=5)
```

Hierarchical (Agglomerative) Clustering:

Agglomerative Clustering is a type of hierarchical clustering algorithm that builds nested clusters by merging pairs of clusters from the bottom up. It starts with each data point as its own cluster and repeatedly merges the closest clusters until one big cluster or a defined number of clusters is formed. This process forms a tree-like structure called a dendrogram.

Purpose:

- Understand the hierarchical structure of data.
- Identify nested groupings at different levels of similarity.
- Provide both clustering and a visual dendrogram for insights.
- In project, It was used to cluster voices based on feature similarity.

When to Use Agglomerative Clustering:

Use Agglomerative Clustering when:

- You want to understand how data points relate hierarchically.

- You need flexibility to choose the number of clusters after clustering (by cutting the dendrogram).
- You expect the data to form nested or multi-scale clusters.
- You have small to medium datasets (as it's computationally expensive).

How It Works (Step-by-step):

- Start with each point as its own cluster.
- Compute a distance matrix between all clusters.
- Merge the two closest clusters.
- Update the distance matrix (based on linkage: single, complete, average, or ward).
- Repeat until all points are merged into one cluster tree.
- You can "cut" the dendrogram at any level to form the desired number of clusters.

Why Use Agglomerative Clustering:

- Does not require specifying the number of clusters upfront.
- Produces a dendrogram that offers insights into data relationships.
- Good for data with hierarchy or sub-clusters.
- Flexible linkage strategies (e.g., Ward, Complete, Average).
- In project, Agglomerative clustering produced clearly defined groups.

Limitations:

- Sensitive to noisy data or outliers.
- Choice of linkage method can significantly affect results.

Code:

```
from sklearn.cluster import AgglomerativeClustering

agglo = AgglomerativeClustering(n_clusters=2) # Assuming 2 clusters
```

Silhouette Score:

The Silhouette Score is a metric used to evaluate the quality of clusters formed by a clustering algorithm. It measures how similar a data point is to its own cluster compared to other clusters.

It ranges from -1 to +1:

- +1 → Perfectly assigned to its own cluster, well-separated from others.
- 0 → On or near the boundary between clusters.
- –1 → Likely assigned to the wrong cluster.

Purpose:

- Quantify how well-separated and cohesive the clusters are.
- Compare the performance of different clustering algorithms.
- Help decide the optimal number of clusters (k).
- In our project, Silhouette Score was used to compare K-Means, DBSCAN, and Agglomerative Clustering and select the best-performing method.

When to Use:

- You perform unsupervised learning (clustering) and need a way to evaluate clustering quality.
- You're comparing multiple algorithms or testing different parameter values (like k or `eps`).
- Ground truth labels (like gender) are not available or not used.

$$Silhouette\ Score = \frac{b - a}{\max(a, b)}$$

- **a** = average distance to all other points in the same cluster (intra-cluster distance).
- **b** = lowest average distance to points in the nearest neighboring cluster (inter-cluster distance).
- The overall score is the average of all sample scores.

Advantage:

- Doesn't require ground truth labels.
- Works with any clustering algorithm.
- Useful for evaluating cluster tightness and separation.
- Helps to compare models or tune parameters effectively.

Limitations:

- Not suitable for very large datasets (computationally intensive).
- Performs poorly on clusters of varying densities or shapes (e.g., DBSCAN clusters).
- Assumes clusters are convex and equally sized, which may bias against irregular clusters.

Code:

```
from sklearn.metrics import silhouette_score

kmeans_score = silhouette_score(X_scaled, kmeans_labels)

dbscan_score = silhouette_score(X_scaled, dbscan_labels) if
len(set(dbscan_labels)) > 1 else -1

agglo_score = silhouette_score(X_scaled, agglo_labels)
```

Calculated value:

Algorithm	Silhouette Score	Interpretation
K-Means	~ 0.176	Moderate cluster separation
DBSCAN	~ 0.014	Poor, clusters not well-formed
Agglomerative	~ 0.187	Best, most meaningful clusters

Therefore, Agglomerative Clustering was selected as the best clustering method.

Classification:

Classification is a type of supervised machine learning where the goal is to predict a categorical label (class) for a given input based on learned patterns from labeled training data.

For example, in your project, classification is used to predict the gender (male/female) of a speaker based on voice features.

Why Use Classification:

- Automatically assign categories to new data points.
- Detect patterns that distinguish different classes (like male vs female voices).
- Enable decision-making in real-world applications like: Email spam detection (spam or not spam)

When to Use Classification:

- Your target variable (label) is categorical.
- You have labeled training data (input features + correct output).
- You want to build a predictive model that assigns labels to new, unseen data.

Types of Classification:

1. Binary Classification:
    - Only two classes.
    - Example: Gender classification (male vs female)
2. Multiclass Classification:
    - More than two classes.
    - Example: Classifying voice types as soprano, alto, tenor, bass.
3. Multilabel Classification:
    - Each instance can belong to multiple labels at once.
    - Example: An audio clip might be both “speech” and “noisy”.

Advantages:

- Accurate prediction when trained well on labeled data.
- Can handle large, complex datasets with multiple features.
- Many powerful models available: Logistic Regression, Random Forest, SVM, Neural Networks, etc.
- Used in critical real-world systems (medical, finance, security).

Limitations:

- Requires labeled data for training.
- Prone to overfitting if not regularized or validated properly.
- Poor generalization if features are not informative or unbalanced.
- Performance highly depends on feature quality and preprocessing.

Classification Model:

A classification model is a supervised machine learning algorithm that learns to assign labels to input data based on patterns discovered in the training dataset.

In your project, classification models learn from voice features (like pitch, spectral centroid, etc.) to predict the gender (male or female) of the speaker.

Why Use Classification Models:

Classification models are used to:

- Predict discrete/categorical outcomes, e.g., yes/no, male/female, class A/B/C.
- Automate decision-making tasks.
- Identify patterns in labeled datasets and apply them to unseen data.
- They are widely applied in: Voice and speech recognition, Email spam detection.

When to Use Classification Models:

Use classification models when:

- Your target variable is categorical (labels, not numbers).
- You have labeled training data (inputs with known output classes).
- You want to predict classes for future/unseen data.

Types of Classification Models:

1. **Logistic Regression:**

    - Simple, interpretable baseline model.
    - Best when features are linearly separable.

2. **Decision Trees:**

    - Tree-based model that splits data based on feature thresholds.
    - Easy to interpret but prone to overfitting.

3. **Random Forest:**

    - An ensemble of decision trees.
    - Improves accuracy and generalization, reduces overfitting.

4. **Support Vector Machine (SVM):**

    - Maximizes the margin between classes.
    - Works well for high-dimensional or non-linear data with kernel trick.

5. **Neural Networks:**

    - Mimics brain structure using layers of interconnected nodes.
    - Can learn complex patterns, good for large, complex datasets.

Advantages:

- **Automation**: Once trained, they can classify new data instantly.
- **Scalability**: Can handle millions of samples with good performance.
- **Customizability**: Many models offer hyperparameters for tuning.
- **Broad Applicability**: Used across domains from healthcare to finance to NLP.

Limitations:

- **Data dependency**: Requires good-quality, labeled data.
- **Bias**: Can be biased if data is imbalanced or skewed.
- **Overfitting risk**: Complex models like neural networks may memorize instead of generalizing.
- **Interpretability**: Some models (e.g., SVM, neural nets) are hard to explain.

Random Forest:

Random Forest is an ensemble machine learning algorithm that builds multiple decision trees and combines their outputs (through majority voting for classification or averaging for regression). It improves prediction accuracy and reduces overfitting by introducing randomness in both data selection and feature selection.

Why Use Random Forest:

You use Random Forest when:

- You want high accuracy.

- You want to avoid overfitting.
- You have a large number of features or complex relationships.
- In our project, It handles non-linear patterns in audio features and It provides feature importance scores to understand which voice features contribute most to gender classification.

When to Use Random Forest:

Use Random Forest when:

- You have a classification or regression task.
- The data is noisy or high-dimensional.
- You need a robust model that generalizes well.

Types / Components:

Random Forest is based on:

- **Bagging:** It's a technique where multiple models (usually the same type) are trained on different random subsets of the training data, sampled with replacement (i.e., bootstrapping). The final prediction is made by combining all models (e.g., majority vote for classification).
- **Random Subspace Method:** Instead of using all features to train each tree, this method selects a random subset of features at each split in the tree.
- **Decision Trees**: The base learners are individual CART decision trees.

There are no subtypes of Random Forest per se, but you can configure:

- Number of trees (`n_estimators`)
- Tree depth (`max_depth`)
- Feature subset size (`max_features`)
- Minimum samples per leaf or split

Advantages:

- **High Accuracy**: Often among the best performing models.
- **Robust to Noise & Overfitting**: Randomization reduces variance.
- **Handles Large Feature Sets**: Effective in high-dimensional data.
- **Feature Importance**: Can rank the most influential variables.
- **Works with Missing Values**: Can handle some missing data without preprocessing.

Limitations:

- **Less Interpretable**: Unlike a single decision tree, a forest is a "black box".
- **Slower to Train**: More trees = more computation, especially with many features.
- **Larger Memory Use**: Stores multiple trees.

Code:

```
rf_params = {'n_estimators': [100, 150], 'max_depth': [None,
10]}

rf_grid =
GridSearchCV(RandomForestClassifier(random_state=42),
rf_params, cv=3, scoring='accuracy')

rf_grid.fit(X_train, y_train)

rf_best = rf_grid.best_estimator_

y_pred_rf = rf_best.predict(X_test)

results['Random Forest'] = y_pred_rf

best_models['Random Forest'] = rf_best
```

Support Vector Machine:

SVM (Support Vector Machine) is a supervised machine learning algorithm used for classification and regression, though it's more commonly used for classification tasks.

It works by finding the optimal hyperplane that best separates data points of different classes in a high-dimensional space.

SVM tries to maximize the margin - the distance between the separating hyperplane and the nearest data points from each class (called support vectors).

When to Use SVM:

Use SVM when:

- The data is high-dimensional or non-linearly separable.
- You have limited data, but want high accuracy.
- You want a robust classifier that works well on clean data with distinct classes.

Why Use SVM:

- Effective in high-dimensional spaces, even when the number of features > number of samples.
- Can handle non-linear relationships using the kernel trick.
- Works well with small to medium-sized datasets.
- Robust to overfitting, especially in high-dimensional settings.

How SVM Works:

- **Linear SVM**: Finds a straight line (2D), plane (3D), or hyperplane (n-D) to separate classes.
- **Non-Linear SVM**: Uses a kernel function (like RBF, Polynomial) to transform data into higher dimensions where a linear separation is possible.
- **Support Vectors**: Critical data points that are closest to the decision boundary. These influence the position and orientation of the hyperplane.

Types of Kernels in SVM:

- **Linear**: When data is linearly separable.
- **Polynomial**: When interaction between features matters.
- **RBF (Gaussian)**: When the decision boundary is non-linear.
- **Sigmoid**: Used less often, similar to neural nets.

Advantages:

- Works well for clear margin of separation.
- Effective in high-dimensional spaces.
- Can use non-linear kernels for complex problems.
- Memory efficient (depends only on support vectors).

Limitations:

- Slower training on large datasets.
- Performance drops when classes are overlapping or not clearly separable.
- Requires careful tuning of parameters like `C`, `gamma`, and the choice of kernel.
- Not ideal for very large datasets due to high training time.

Best Model in project:

SVM is useful because it:

- Can capture complex non-linear boundaries between male and female voice features.
- Performs well in high-dimensional audio feature space.
- Is especially good with clear binary classification, like gender (male vs female).

Code:

```
svm_params = {'C': [1, 10], 'kernel': ['rbf', 'linear']}

svm_grid = GridSearchCV(SVC(probability=True), svm_params,
cv=3, scoring='accuracy')

svm_grid.fit(X_train, y_train)

svm_best = svm_grid.best_estimator_

y_pred_svm = svm_best.predict(X_test)
```

```
results['SVM'] = y_pred_svm

best_models['SVM'] = svm_best
```

Neural Network:

A Neural Network (NN) is a supervised machine learning model inspired by the structure of the human brain. It consists of layers of interconnected neurons that can learn complex patterns from data.

It is especially powerful for tasks like image recognition, speech recognition, and audio-based classification making it a strong candidate in your voice gender classification project.

Structure of a Neural Network:

- **Input Layer**: Takes in the features (e.g., pitch, spectral centroid, MFCC, etc.)
- **Hidden Layers**: One or more layers where computations are performed using activation functions.
- **Output Layer**: Produces the final prediction (e.g., Male or Female).

Why Use Neural Networks:

- Can model highly non-linear relationships.
- Works well with large feature sets.
- Learns directly from raw or engineered features without manual rule creation.
- Ideal for tasks involving pattern recognition, such as speech or audio classification.

When to Use Neural Networks:

- You have enough data and computational power.
- Your dataset has complex patterns or non-linear boundaries.
- You want to achieve high accuracy in classification tasks.
- You're dealing with multi-dimensional or time-series data (like audio signals).

How It Works:

- **Forward Pass:** Data is passed through layers. Each neuron computes a weighted sum and applies an activation function like ReLU, Sigmoid, or Tanh.
- **Loss Calculation:** The output is compared with the true label using a loss function (e.g., binary cross-entropy for classification).
- **Backward Pass (Backpropagation):** The network adjusts weights using gradient descent to reduce the loss.
- **Iterations (Epochs):** The model repeats this process over several passes through the data to learn.

Advantages:

- High flexibility — can approximate any function.
- Handles non-linearity better than traditional algorithms.
- Performs well on high-dimensional feature spaces.
- Learns hierarchical representations automatically.

Limitation:

- Requires more data and longer training time.
- Harder to interpret than tree-based models (like Random Forest).
- May overfit if the network is too large or not properly regularized.

Code:

```
mlp_params = {'hidden_layer_sizes': [(100,), (50, 50)],
'activation': ['relu', 'tanh']}

mlp_grid = GridSearchCV(MLPClassifier(max_iter=500),
mlp_params, cv=3, scoring='accuracy')

mlp_grid.fit(X_train, y_train)

mlp_best = mlp_grid.best_estimator_

y_pred_mlp = mlp_best.predict(X_test)

results['Neural Network'] = y_pred_mlp

best_models['Neural Network'] = mlp_best
```

Logistic Regression:

Logistic Regression is a supervised classification algorithm used to predict the probability of a binary outcome (e.g., Male or Female) based on one or more input features.

Unlike linear regression (which predicts a continuous value), logistic regression uses a sigmoid function to output probabilities between 0 and 1.

When to Use:

- When the target variable is binary.
- When relationships between features and the target are approximately linear in log-odds.
- When you need a simple, interpretable, and fast baseline model.
- When explainability matters (feature coefficients are easy to interpret).

Why Use It:

- It’s easy to implement and interpret.
- It gives probabilistic outputs (confidence scores).
- It performs well when classes are linearly separable.
- Works well with high-dimensional feature sets (especially after feature selection).

Advantages:

- Simple and fast to train.
- Outputs interpretable coefficients.
- Requires less data than complex models.
- Good baseline model to compare others against.

Limitation:

- Assumes linear decision boundary.
- Can underperform on complex or non-linear datasets.
- Sensitive to outliers.
- Doesn’t perform well when there’s multicollinearity between features.

Code:

```
logreg_params = {'C': [0.1, 1, 10]}

logreg_grid = GridSearchCV(LogisticRegression(max_iter=1000),
logreg_params, cv=3, scoring='accuracy')

logreg_grid.fit(X_train, y_train)

logreg_best = logreg_grid.best_estimator_

y_pred_logreg = logreg_best.predict(X_test)

results['Logistic Regression'] = y_pred_logreg

best_models['Logistic Regression'] = logreg_best
```

Decision Tree:

A Decision Tree is a supervised machine learning algorithm used for classification and regression tasks. It splits the data into subsets based on the values of input features using a tree-like structure.

Each internal node represents a decision (a test on a feature), each branch is an outcome of the test, and each leaf node represents a class label (e.g., Male or Female in your project).

How it Works:

- The tree chooses the best feature and threshold to split the dataset.
- The "best" split minimizes impurity using criteria such as:
    - Gini Impurity (common in classification)
    - Entropy / Information Gain.
- The splitting continues recursively until a stopping condition is met (e.g., max depth, no further gain, pure leaf).

When to Use:

- When your data is tabular and contains categorical or numerical features.
- When you want a model that is easy to interpret.
- When you want fast training and prediction, especially with small datasets.
- When relationships between features and the target are nonlinear or hierarchical.

Why Use It:

- It can capture nonlinear patterns.
- Doesn't require feature scaling (like SVM or Logistic Regression).
- Easy to visualize and understand.
- Can handle both categorical and numerical data.

Advantages:

- **Interpretable**: Easy to explain decisions via paths from root to leaf.
- **Fast**: Quick training and prediction.
- **Handles Mixed Data**: Works with both categorical and continuous features.
- **No Scaling Needed**: Doesn't require normalization or standardization.

Limitations:

- **Overfitting**: Can create complex trees that fit noise in the training data
- **Instability**: Small data changes can produce different trees.
- **Biased Splits**: Can favor features with more levels or variance.
- **Lower Accuracy**: May underperform compared to ensemble methods like Random Forest or XGBoost.

Code:

```
dt_params = {'max_depth': [5, 10, None]}

dt_grid = GridSearchCV(DecisionTreeClassifier(), dt_params,
cv=3, scoring='accuracy')

dt_grid.fit(X_train, y_train)

dt_best = dt_grid.best_estimator_

y_pred_dt = dt_best.predict(X_test)
```

```
results['Decision Tree'] = y_pred_dt

best_models['Decision Tree'] = dt_best
```

GridSearchCV: (Grid Search with Cross-Validation)

`GridSearchCV` is a model tuning technique that helps find the best combination of hyperparameters for a given algorithm by:

- Defining a grid of possible hyperparameter values.
- Trying every combination of these values.
- Evaluating model performance using cross-validation.
- Returning the best performing model.

Purpose:

- To optimize model performance by systematically tuning hyperparameters.
- Avoids manual trial-and-error tuning.
- Helps prevent underfitting or overfitting by finding the best settings.

When to Use:

When you're using models with tunable hyperparameters like:

- SVM (e.g., `C`, `kernel`, `gamma`)
- Random Forest (e.g., `n_estimators`, `max_depth`)
- Logistic Regression, KNN, etc.

When you want robust validation of hyperparameter choices using cross-validation.

How It Works (Step-by-Step):

- Define the model and the set of hyperparameters to tune.
- `GridSearchCV` automatically trains the model on every combination of those parameters.
- It uses k-fold cross-validation to evaluate each combination.
- It selects the best model (with highest score or lowest error) and provides the optimal parameters.

Advantage:

- Optimized Models: Helps find the best hyperparameter combo.
- Cross-Validation: More reliable performance estimates.
- Automation: Reduces manual tuning work.
- Performance: Often boosts accuracy or reduces error.

Limitation:

- **Slow**: Can be very time-consuming with large grids.
- **Computational Cost**: Requires training the model many times.
- **Rigid**: Tries all combinations even if some are unnecessary.

Classification Evaluation Metrics:

When training a classification model (like Random Forest, SVM, etc.), it's not enough to just look at accuracy — especially with imbalanced datasets (like if one gender dominates). We use multiple metrics to get a holistic view of model performance:

1) **Accuracy:**

Accuracy is the most basic and widely used metric to evaluate a classification model. It tells you how many predictions your model got correct out of all predictions it made.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} = \frac{TP + TN}{TP + TN + FP + FN}$$

Where:

- **TP** = True Positives (correctly predicted positive class)
- **TN** = True Negatives (correctly predicted negative class)
- **FP** = False Positives (incorrectly predicted as positive)
- **FN** = False Negatives (incorrectly predicted as negative)

It's simple to understand and calculate. Useful when your dataset has balanced classes. Accuracy can be deceptive if the classes are imbalanced.

Code:

```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy)
```

2) **Precision:**

Precision measures the accuracy of the positive predictions made by your model.

$$Precision = \frac{TP}{TP + FP}$$

Where:

- **TP (True Positives)** = Correctly predicted positives (e.g., correctly predicted female voices)
- **FP (False Positives)** = Incorrectly predicted positives (e.g., male voices wrongly predicted as female)

Precision is important when false positives are costly or undesirable. When the cost of false positives is high. In imbalanced datasets, where one class is more frequent than the other.

Code:

```
from sklearn.metrics import precision_score

precision = precision_score(y_test, y_pred,
pos_label='female')

print("Precision:", precision)
```

3) **Recall:**

Recall (also known as Sensitivity or True Positive Rate) measures the ability of the model to find all relevant cases — that is, how many actual positive cases it successfully identified.

$$Recall = \frac{TP}{TP + FN}$$

Where:

- **TP (True Positives)**: Correctly predicted positive samples.
- **FN (False Negatives)**: Actual positives wrongly predicted as negative.

Recall is crucial when missing a positive case is more costly than a false alarm. When false negatives are critical (e.g., medical tests, fraud detection, or missing female voices in your classification). In imbalanced datasets, where one class may be underrepresented.

Code:

```
from sklearn.metrics import recall_score

recall = recall_score(y_test, y_pred, pos_label='female')

print("Recall:", recall)
```

4) **F1-Score:**

The F1-Score is the harmonic mean of precision and recall — it balances both false positives and false negatives.

$$F1\ Score = 2\ X\ \frac{Precision\ x\ Recall}{Precision\ x\ Recall}$$

- It ranges from 0 to 1.
- A perfect model has an F1-score of 1.0.

F1-Score is especially useful when:

- You have imbalanced classes
- You want to balance the trade-off between false positives and false negatives
- Neither precision nor recall alone gives a full picture

When to Use F1-Score:

- When both precision and recall matter
- In imbalanced classification problems
- When you want a single metric to evaluate model performance holistically

Code:

```
from sklearn.metrics import f1_score

f1 = f1_score(y_test, y_pred, pos_label='female')

print("F1-Score:", f1)
```

5) Confusion Matrix:

A confusion matrix is a summary table used to evaluate the performance of a classification model.

It shows the number of:

- Correct predictions (both true positives and true negatives)
- Incorrect predictions (false positives and false negatives)

	Predicted: Female	**Predicted: Male**
Actual: Female	True Positive (TP)	False Negative (FN)
Actual: Male	False Positive (FP)	True Negative (TN)

It gives detailed insight into how your model is performing beyond just accuracy. Helps you understand where the model is making mistakes (e.g., misclassifying male as female). Always use after classification to visualize model performance. Crucial for imbalanced datasets where accuracy can be misleading.

Best Model:

- High F1-score → for balanced performance.
- High recall → if missing a class (e.g., misclassifying females) is costly
- High precision → if false alarms are undesirable

After evaluation the best model is SVM.

Streamlit app:

Objective:

- Classify human voices into male or female using machine learning models based on acoustic features.
- Cluster voices into distinct groups using unsupervised learning.
- Enable interactive EDA, prediction, and visualization of results.

App Pages Overview:

The app is menu-driven with 5 interactive pages:

- **Introduction**: Describes the motivation, dataset, and applications.
- **EDA (Exploratory Data Analysis)**: Visual analytics by feature/gender.
- **Prediction**: Live gender prediction using user input on top 15 features.
- **Best Classification**: Displays best model (SVM) results & metrics.
- **Best Clustering**: Visualizes clustering results using PCA projection.

Dataset:

- CSV File: `vocal_gender_features_new.csv`
- Columns: Pre-extracted numerical acoustic features from voice recordings.
- Target: `label` column (binary – 0 for female, 1 for male).

Preprocessing:

- Features (`X_full`) and Labels (`y`) are split.
- Data is standardized using `StandardScaler` for both training and full set.
- For classification, a 15-best-feature subset is selected using `SelectKBest`.

EDA Page Logic:

- Offers 10 feature analysis questions, each rendered using Seaborn plots.
- Visuals include:
  - Boxplots, histograms, scatterplots, violin plots, and heatmaps.
  - A special multi-subplot view for spectral skew and kurtosis.
  - Helps detect feature patterns across genders (e.g., pitch, centroid, energy).

Prediction:

Applies:

- Feature Selection: `SelectKBest(f_classif, k=15)`
- Scaling: `StandardScaler`
- Model: SVM with RBF kernel (C=10) trained on selected features.

Accepts user input via sliders for each feature.

Outputs:

- Predicted gender with confidence score
- Cluster position of the input using Agglomerative Clustering + PCA

Best Classification:

- Displays evaluation metrics for the best model (SVM):
  - Accuracy, Precision**,** Recall, F1-Score

Shows Confusion Matrix for full dataset predictions using:

- `ConfusionMatrixDisplay` with labels "Female" and "Male"

Best Clustering:

- Uses Agglomerative Clustering on full dataset.
- Computes and displays Silhouette Score to evaluate clustering quality.
- Projects clustering in 2D space using PCA.
- Visualizes clusters in a scatter plot with color-coded labels.

ML Models Used:

Classification Models:

- Support Vector Machine (SVM) - Best performing.
- Others (imported but not used in current code): RF, MLP, Logistic Regression, Decision Tree.

Clustering Models:

- Agglomerative Clustering - Best (based on silhouette score).
- Others (mentioned): KMeans, DBSCAN.

Conclusion:

This project successfully demonstrates the application of machine learning techniques for gender classification and unsupervised clustering using acoustic voice features. It demonstrates the potential of combining signal processing, statistical learning, and

visualization techniques to solve real-world problems in voice recognition. The pipeline developed here can serve as a foundation for advanced applications such as speaker identification, emotion detection, and personalized voice assistants.