Cairo University
Faculty of Engineering

Department of Computer Engineering

# Neural Network

## Project
## Speaker Gender Age Recognition

**Submitted by:**
Kareem Mohamed (ID: 9220600)
Moamen Hefny (ID: 9220886)
Marwan Mohamed (ID: 9220808)
Youssef Tarek (ID: 9220990)

**Spring 2025**

# Contents

# 1 Preprocessing

The preprocessing stage was responsible for preparing raw audio files before feature extraction and training. To handle this, we developed a pipeline built around the `Essentia` library, which provided a solid foundation for reading and processing audio data.

## 1.1 Audio Loading and Validation

Each audio file was first checked for existence and size to avoid corrupt or empty files. Using Essentia's `MonoLoader`, the audio was converted to mono format and loaded into memory. Along with the raw waveform, we also extracted the sample rate and computed the duration of the clip. Files that failed any check were automatically skipped to maintain dataset quality.

## 1.2 Preprocessing Options

Our preprocessing pipeline included several configurable operations, which could be toggled on or off via command-line arguments. These steps were applied in the following order:

- **Trimming:** Clips were trimmed or padded to a fixed target duration (default 4 seconds). This ensured that all input samples were of uniform length, which is essential for consistent feature extraction.
- **Volume Normalization:** Each clip was normalized to a target RMS level (default 0.2), allowing for consistent loudness across the dataset.
- **Noise Reduction:** Although implemented, this option was disabled by default to avoid distorting clean audio. It could be enabled to suppress background noise below a given threshold.
- **Silence Removal:** Segments of silence were detected and removed based on a minimum silence duration (default 500ms) and a threshold level (default 0.01). This helped in focusing the model on actual speech content.

Each of these steps could be individually disabled using flags such as `--no-trim`, `--no-normalize`, `--no-noise-reduction`, and `--no-silence-removal`.

## 1.3 Batch Processing and Dataset Handling

To handle large datasets efficiently, we implemented support for batch processing using a TSV file. This file contained paths to audio samples and optional metadata. We could specify a range of lines to process (e.g., start and end line), a maximum number of files, and the output directory.

# 2 Cleaning Dataset

Before training, the dataset underwent a cleaning phase to ensure that all entries were valid and usable. This step focused on filtering out problematic files, reducing noise in metadata, and preparing a balanced subset for training.

## 2.1 Metadata Filtering and Validation

We built a metadata cleaning tool to process the original TSV file associated with the dataset. This tool dropped irrelevant columns, removed rows that pointed to missing or unreadable audio files, and retained only entries that could be successfully loaded and analyzed.

For each entry, the audio file was checked for existence and duration. Only those with a valid path and a positive duration were kept. The cleaned metadata was saved in a new TSV file with just the filename, age group, gender, and duration.

## 2.2 Configurable Cleaning Script

The cleaning process was configurable through command-line arguments. Key parameters included:

- `--dataset-path=<path>`: Specifies the directory containing the audio files.
- `--metadata-file=<file>`: Points to the input metadata TSV file.
- `--clean-tsv`: Enables the metadata validation and cleaning step.

## 2.3 Categorization by Gender and Age Group

After filtering, the entries were categorized based on gender and age group. The target categories were set to `male` and `female` for gender, and `twenties` and `fifties` for age groups. Each audio entry was mapped into its corresponding category based on these two labels.

## 2.4 Exporting a Balanced Dataset

To ensure balanced training data, the user was prompted to enter the number of samples per category to include in the final dataset. For each category, the tool randomly selected the desired number of samples (shuffling the data beforehand) and saved the resulting subset to a new TSV file.

The exported file contained one entry per line in the format: filename, age, gender, and duration. All categories were kept balanced with the same number of entries.

# 3 Feature Extraction

To convert audio data into a structured form suitable for machine learning, we implemented a robust feature extraction pipeline. This system supports multiple types of acoustic features commonly used in speech processing. After extensive experimentation with different feature sets and their combinations, we found that MFCCs (Mel-Frequency Cepstral Coefficients) with 26 coefficients yielded the highest classification accuracy.

## Supported Feature Types

Our feature extraction module is capable of computing the following acoustic features:

- **MFCC (Mel-Frequency Cepstral Coefficients)**: Captures the timbral texture of audio; both mean and standard deviation across frames can be computed.
- **Chroma Features**: Represents the energy distribution across 12 pitch classes (e.g., C, C#, D, etc.) and is useful for capturing harmonic content.
- **Spectral Contrast**: Measures the difference in energy between spectral peaks and valleys across frequency bands.
- **Tonnetz Features**: Based on harmonic relationships and tonal centroid features.
- **Mel Spectrogram**: Represents power across Mel-scaled frequency bands and captures temporal-spectral information.

## Feature Selection and Performance

We systematically evaluated the effectiveness of each feature type, as well as several combinations. The table below summarizes the supported features and whether they were ultimately selected for model training:

| **Feature Type** | **Supported** | **Selected for Final Model** |
|---|---|---|
| MFCC (26 coefficients) | Yes | **Yes** |
| Chroma | Yes | No |
| Spectral Contrast | Yes | No |
| Tonnetz | Yes | No |
| Mel Spectrogram | Yes | No |

Table 1: Summary of supported features and final selection

## Pipeline Details

The pipeline reads metadata from a TSV file, processes the corresponding audio files, extracts the selected features, and saves them to new TSV files for training and testing. The system is designed to be configurable via command-line arguments, and supports options such as train/test split ratio, random seed initialization, and output directory specification.

Progress tracking and error reporting are included to enhance usability and transparency. In cases where an audio file is missing or unreadable, the pipeline logs the error without halting execution.

# 4 Model Training

## 4.1 Classifier Overview

The stacking classifier combines predictions from multiple base learners via a meta-learner to improve overall accuracy.

- **Base learners:** A set of individual classifiers trained on the original feature space.
- **Meta-learner:** A classifier trained on out-of-fold predictions of the base learners.
- **Cross-validation:** K-fold splitting to generate out-of-fold predictions without data leakage.

## 4.2 Algorithm Pseudocode

```
// Inputs: feature matrix X (N D), labels y (N),
//          base models B[1..L], meta_model M, folds K
Shuffle indices 0..N-1
Partition indices into K folds
Initialize Z[N][L] = 0

// Generate out_of_fold predictions
for l = 1 to L parallel:                // train each base in parallel
  for fold = 1 to K:
    train_idx = all indices not in fold
    test_idx  = indices in fold

    B[l].train( X[train_idx], y[train_idx] )
     y   = B[l].predict( X[test_idx] )
    for i in test_idx:
      Z[i][l] = y  [i]

// Train meta_learner
M.train( Z, y )

// Retrain each base on full data
for l = 1 to L:
  B[l].train( X, y )
```

## 4.3 Base Models Comparison

The following table summarizes all supported base learners and which combination was selected in the final system.

| **Model** | **Supported** | **Used** | **Notes** |
|---|---|---|---|
| Support Vector Machine (RBF) | Yes | **Yes** | High margin separation, tuned C and $\gamma$ |
| Random Forest | Yes | No | 700 trees, similar accuracy but slower |
| Extra Trees | Yes | No | 400 trees, fast but no gain over RF |
| k-Nearest Neighbors (KNN) | Yes | **Yes** | $k = 5$, Euclidean distance |
| Neural Network (2-layer) | Yes | No | 64+32 hidden units, prone to overfitting |

Table 2: Base learners supported and final selection

## 4.4 Configuration and Saving

- `--n-folds`: number of CV folds (default 5).
- `--seed`: random seed for reproducibility.
- `--train-path`, `--test-path`: feature TSV inputs.
- Models and configuration are saved to a directory with subfolders per base learner and a `config.txt` recording parameters.

# 5 Inference Pipeline

This stage runs the trained stacking classifier on a directory of test audio files, producing predicted labels and evaluation metrics.

## 5.1 Initialization and Configuration

The executable accepts the following command-line options:

- `--data-dir=<path>`: directory containing test audio files (`.wav` or `.mp3`).
- `--model-dir=<path>`: directory where the trained models and `summary.txt` reside.
- `--ground-truth=<file>`: TSV file with true labels for accuracy reporting.
- `--mode=<single|combined>`:
  - `single`: use one stacking model for both gender and age combined.
  - `combined`: load separate gender and age classifiers.
- `--gender-prefix`, `--age-prefix`: prefixes for subdirectories when `mode=combined`.

## 5.2 Model Loading

1. Parse `summary.txt` to retrieve hyperparameters (SVM C, gamma, KNN k, etc.).
2. Instantiate base learners (SVM, KNN) and the logistic regression meta-learner.
3. Call `loadModels()` on the appropriate directory or subdirectories.

## 5.3 Test File Discovery

All non-empty files with `.wav` or `.mp3` extensions in `data-dir` are collected. Empty or unreadable files are skipped.

## 5.4 Feature Extraction

- For each file, run the audio preprocessor (trimming and normalization disabled).
- Extract MFCC feature vectors using the same parameters as during training.
- Accumulate feature vectors into an Eigen matrix $\mathbf{X}$.

## 5.5 Prediction

1. If `mode=combined`, separately predict gender and age, then combine into a single class code (class $= 2 \times$ ageCode $+$ genderCode).
2. If `mode=single`, predict directly from the combined model.
3. Store the vector of predicted class indices.

## 5.6 Output and Evaluation

- Write predicted class indices to `results.txt` (one per line).
- Record total inference time in `time.txt`.
- If a ground-truth TSV is provided, compare predictions to true labels:
  - Generate `accuracy.txt` with one line per file showing filename, true label, predicted label, and "Correct"/"Wrong" flag.
  - Log overall accuracy percentage.