Pattern Recognition Project Report

Submitted by:

1. Ahmed Tarek Abdelal 1200088
2. Hanzada Fayez Yehia 1200075
3. Jomana Hossam Youssef 1200023

Team: 8

Table Of Contents:

Project Report: Speech Classification System

(a) Project Pipeline

This project presents a comprehensive and modular speech classification system, focusing on transforming raw audio signals into class predictions through a series of structured stages. Each component of the pipeline has been developed for reusability, scalability, and robustness in handling real-world audio data.
The major stages in the pipeline include:

1. **Preprocessing Module**: Cleans and augments raw audio data.
2. **Feature Extraction/Selection Module**: Extracts handcrafted acoustic and prosodic features.
3. **Model Training and Evaluation Module**: Trains models, tunes hyperparameters, and evaluates performance.
4. **Performance Analysis Module**: Computes classification metrics.
5. **Enhancements and Future Work**: Explores areas for improvement and expansion.

(b) Preprocessing Module

The preprocessing module is essential for converting real-world, noisy audio recordings into clean, standardized inputs for feature extraction. It consists of several steps, each designed to address specific challenges in audio processing.

1. Audio Loading

- Audio files are loaded using the `librosa` library at a fixed sampling rate of 16 kHz.
- Error handling ensures the system skips unreadable or corrupt files without halting execution.

2. Silence Removal

- Uses `librosa.effects.split` to segment non-silent regions.
- Reduces computational overhead and emphasizes informative speech segments.

3. Volume Normalization

- Standardizes audio loudness across all files to eliminate recording condition biases.

4. Noise Reduction

- Applies the `noisereduce` library to suppress stationary background noise.
- Particularly effective for real-world audio with environmental disturbances.

5. Data Augmentation

- Improves model robustness and mitigates class imbalance.
- Techniques include:
  - **Gaussian Noise**: Simulates environmental noise.
  - **Time Stretching**: Slightly speeds up or slows down the audio.
  - **Pitch Shifting**: Alters pitch without changing duration.
  - **Shifting**: Time shifts the waveform.
- Augmentation is conditional based on class distribution, applying more augmentation to underrepresented classes.

6. Padding (Optional)

- Audio clips can be padded or trimmed to 5 seconds for consistent input length.
- Not used in the current configuration but supported for future expansion.

7. Caching Mechanism

- Saves preprocessed data as `.npy` files.
- Speeds up experimentation by avoiding redundant computations.

8. Dataset Splitting

- Uses stratified sampling to maintain label distribution.

- Default split: 75% training, 10% validation, 15% testing.

9. SMOTE for Class Balancing

- Optional technique to oversample minority classes using `SMOTETomek`.
- Combats class imbalance while reducing noisy overlaps between classes.

(c) Feature Extraction/Selection Module

This module transforms time-domain waveforms into fixed-length, meaningful numerical representations that can be used by classifiers.

1. Feature Categories

The extracted features fall into three main categories:

- **Spectral Features**:
  - Capture timbral and frequency-based characteristics.
- **Prosodic Features**:
  - Capture rhythm and intonation patterns.
- **Phonetic Features**:
  - Represent physical speech characteristics.

2. Feature Descriptions

- **MFCC (Mel-Frequency Cepstral Coefficients)**:
  - 13 coefficients that describe the short-term spectral envelope.
- **Chroma Features**:
  - Energy distribution across 12 pitch classes.
- **Spectral Contrast**:
  - Differences between spectral peaks and valleys.
- **Spectral Centroid**:
  - Perceived brightness of the audio.
- **RMS Energy**:
  - Signal strength.

- **Zero Crossing Rate (ZCR)**:
    - Number of sign changes in the signal.

- **Pitch (F0)**:
    - Extracted using YIN algorithm (mean, std, max).

- **Loudness**:
    - Derived from dB-scaled magnitude spectrogram.

- **Duration Statistics**:
    - Count, mean, and std of voiced segments.

- **Formants (F1, F2)**:
    - Vocal tract resonances extracted via `parselmouth`.

3. Feature Vector Construction

- All features are statistically summarized (mean, std, max where applicable).

- Combined into a single fixed-length vector per audio sample.

- Supports consistent input shape for all models.

4. Feature Caching

- Saves extracted features and labels as `.npy` files.

- Version-controlled to allow experimentation with different feature sets.

(d) Model Selection/Training Module

This module provides a flexible and extensible structure for training and evaluating classification models.

1. BaseModel Interface

- Defines essential methods: `train`, `predict`, `score`.

- Integrates with MLflow for logging metrics, models, and parameters.

- Provides `load_model_from_run` to reload past models.

2. ModelPipeline Wrapper

- Manages the end-to-end training workflow:
    - Preprocessing
    - Model fitting
    - Prediction
    - Evaluation
    - Logging
- Accepts any class extending `BaseModel`.
- Supports optional Optuna-based hyperparameter tuning.

3. XGBoostModel

- Custom implementation of the XGBoost classifier.
- Features:
    - GPU acceleration via CuPy (if available).
    - Class weighting using `compute_sample_weight`.
    - Optuna integration for optimizing parameters such as:
        - `learning_rate`
        - `max_depth`
        - `scale_pos_weight`

4. Training Workflow

- Training involves initializing the `ModelPipeline` with the model and dataset.
- Optionally configures MLflow tags and Optuna trials.
- Logs artifacts for reproducibility.

(e) Performance Analysis Module

This module handles the evaluation of trained models using standard classification metrics. It offers structured outputs for programmatic use and human-readable summaries.

Features:

- Returns both:
    - Dictionary of metrics (for logging or MLflow use).
    - String-based report (for terminal or UI output).
- Metrics include:
    - Accuracy
    - Precision
    - Recall
    - F1-score (macro, micro, weighted)
- Handles zero-division cases gracefully.

(f) Optional Modules

Although not essential for core functionality, additional tools were implemented to support experimentation and usability.

MLflow Integration

- All training experiments are tracked using MLflow.
- Automatically logs:
    - Parameters
    - Metrics
    - Artifacts (models, features, plots)
    - Tags for filtering runs

Optuna Integration

- Enables hyperparameter search over a configurable number of trials.

- Automatically selects best-performing configuration based on validation F1-score.

Utility Scripts

- Scripts were developed for batch preprocessing, experiment orchestration, and performance benchmarking.

(g) Enhancements and Future Work

Several areas for improvement and future development have been identified:

1. Deep Learning Integration

- Replace XGBoost with end-to-end CNN or transformer-based models.
- Use raw waveform inputs or spectrograms.

2. Real-Time Inference

- Convert system to support real-time predictions via audio stream segmentation.

3. Deployment

- Build an inference API using FastAPI or Flask.
- Package model using ONNX or TorchScript.

4. Data Expansion

- Incorporate additional corpora to increase generalization.
- Include multilingual support.

5. Feature Learning

- Replace handcrafted features with learned embeddings (e.g., wav2vec, OpenL3).

6. GUI Dashboard

- Develop an interactive UI for analyzing uploaded speech files.
- Include visualization for waveform, spectrogram, and feature importance.