Acoustical Command Recognition for Directional Control

Syed Murtuza Quadri
Sanjit Chavan

Summer Semester, 2025

Semester Project in Sensor Signal Processing
Supervisor: Prof. Dr.-Ing. Andreas König

Overview

1. **Introduction**
   - Sound and Waveforms
   - Audio Signals
2. **Data Origin & Acquisition**
   - Data Acquisition Challenges
   - Standardized Data Acquisition
   - Raw Signal
3. **Signal Preprocessing**
   - DC Offset
   - Event Detection
   - Padding
4. **FFT approach**
   - Fast Fourier Transform
   - Feature Extraction
   - Dimensionality Reduction
   - Classification (KNN, SVM, Decision -Tree), with live classification

Overview

5. **Spectrograms**
   - Feature Extraction
   - Dimensionality Reduction
   - Classification (KNN, SVM RBF)
6. **Reduced Spectrograms**
7. **Live Classification**

Project Tasks

To design, implement, and evaluate a system for Acoustical Command Recognition that translates spoken directional commands into real-time control of an entity on a graphical interface.

1. **Data Collection:** Record 'up', 'down', 'left', 'right', & 'stop' commands (>30 samples/class) from multiple speakers.
2. **Preprocessing:** Convert audio to .npy, extract the spoken "event," and split data into train/test sets.
3. **Live Demo:** Create a Python GUI to show real-time entity control based on voice commands.

a) **FFT Approach:**
- Extract features using Fast Fourier Transform (FFT).
- Apply Dimensionality Reduction.
- Classify using 3+ models

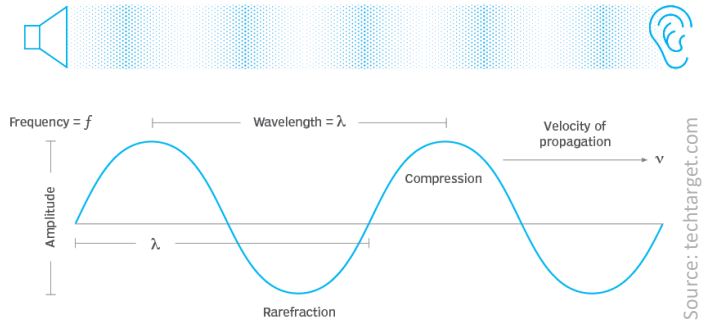
b) **Spectrogram Approach:**
- Compute spectrograms as features.
- Classify using 3+ models

c) **Compressed Spectrogram Approach:**
- Compute and compress spectrograms.
- Classify using 1+ model from 1.b for performance comparison.

Potential Use Cases

- **IoT / Embedded Systems:** Smart lights, fans, or toys that need local, offline control without internet dependency.
- **Robotics:** Controlling small robots, drones, or vehicles using voice. The "up, down, left, right, stop" set is exactly a fit.
- **Accessibility:** Helping people with motor impairments issue a few reliable commands without needing full speech-to-text.
- **Industrial Environments:** Workers can use simple voice commands (stop/go) with machinery where pressing buttons isn’t safe or convenient.
- **Games & VR/AR:** Voice commands for controlling avatars or navigating menus.

Sound and Waveforms



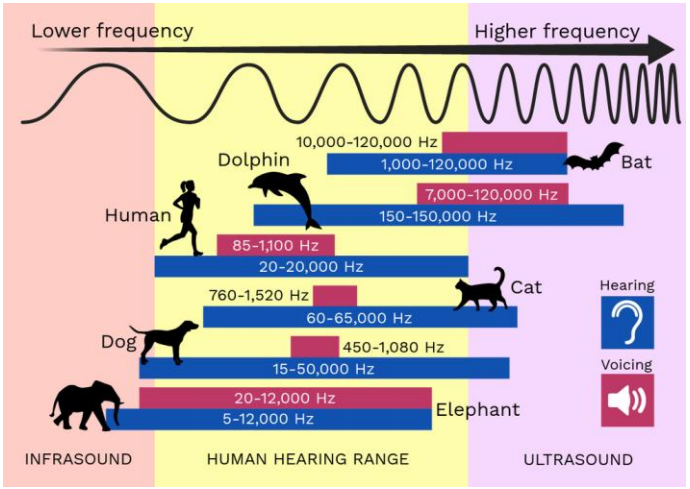
Amplitude: Represents the maximum pressure variation from the equilibrium. It directly relates to perceived loudness, often quantified in decibels (dB) for a logarithmic scale.

Frequency (f): The rate of oscillation of the pressure wave, measured in Hertz (Hz). Determines the perceived pitch of a sound, critical parameter for identifying phonemes and acoustic events in speech.

Wavelength (λ): The spatial period of the wave, representing the distance over which the wave's shape repeats.

Sound propagates through a medium as longitudinal waves, particle displacement is parallel to wave propagation.

Fundamental wave equation: **velocity =f·λ**



Diverse acoustic frequency ranges across species, with the human hearing and voicing spectrum

Sound and Waveforms

Perception: Equal-Loudness Contours

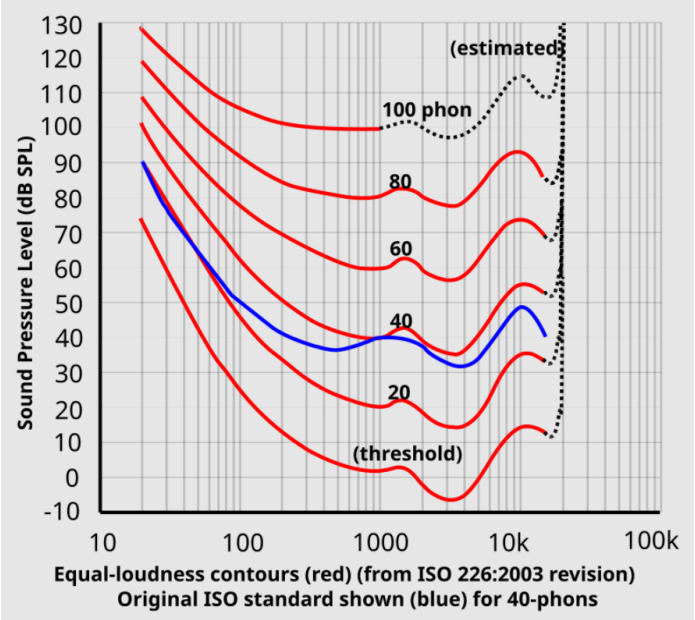
Each contour represents a specific phon level, where the phon is a unit of perceived loudness.

The horizontal axis represents frequency on a logarithmic scale, spanning from infrasound to ultrasound.

The vertical axis indicates Sound Pressure Level (SPL) in decibels (dB), a physical measure of sound intensity.

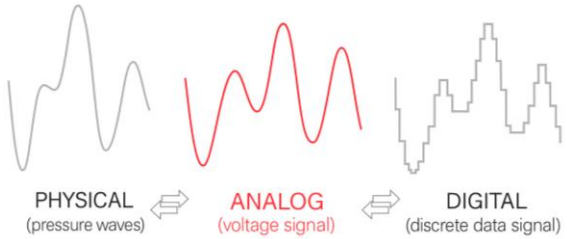
Each curve connects points of equal subjective loudness.

For instance, the lowest contour (0 phon) depicts the absolute threshold of human hearing, where minimal SPLs become just perceptible.



Source: n.wikipedia.org/wiki/Equal-loudness_contour

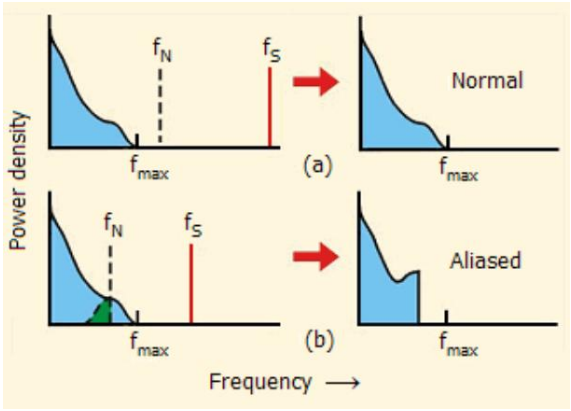
Audio Signals



Analog Signal (Voltage Signal):
Transduced physical signal (e.g., by a microphone) into a continuous electrical voltage waveform. This is an analogous representation of the physical sound.

Digital Signal (Discrete Data Signal):
Analog signal converted into a sequence of discrete numerical values (samples), suitable for digital storage and processing.

Nyquist-Shannon Sampling Theorem:
Any band-limited continuous-time analog signal can be perfectly reconstructed from its discrete samples, provided the sampling rate (f_s) is at least twice the highest frequency component (f_{max}) present in the analog signal.



Source: http://webapps.chem.uoa.gr

Representation: $f_s \geq 2 \times f_{max}$
Nyquist Frequency (f_N): Defined as half the sampling rate ($f_N = f_s/2$). It represents the maximum frequency that can be accurately captured without aliasing.

Audio Signals

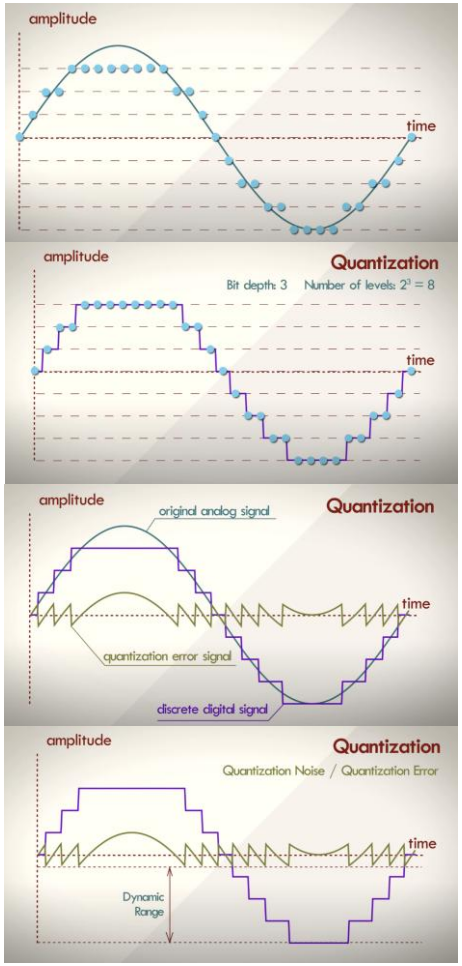
Quantization: Conversion of continuous analog amplitudes into discrete digital values.

Bit Depth (Q): Determines the number of discrete amplitude levels (2^Q). Higher bit depth = finer resolution = more accurate digital representation.

Quantization Noise: Inherent error from rounding analog amplitudes to the nearest discrete level.

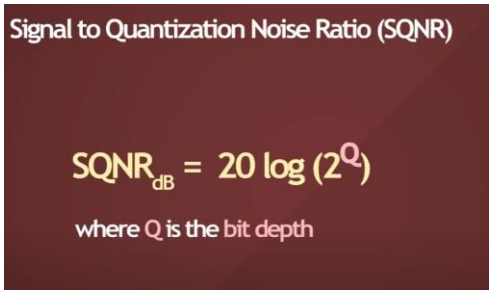
Quantization Error: The difference between original and quantized amplitude, manifesting as noise. Limits signal fidelity; higher bit depth reduces this noise.

Dynamic Range: The ratio between the loudest undistorted signal and the quietest discernible signal/noise floor.



SQNR: Measures signal power relative to quantization noise power.

Higher SQNR = better signal quality, less audible noise.



https://www.youtube.com/@akashmurthy

2. Data Origin & Acquisition

Data Acquisition Challenges

Uncontrolled Environment:
Initial recordings were acquired from various acoustic environments, introducing significant background noise (e.g., fan noise) and acoustic variability that complicated subsequent processing stages like Event Detection

Heterogeneous Sensor Configuration:
Data was captured using a mix of microphones (e.g., Bluetooth headsets, internal laptop mics), leading to inconsistent frequency responses and Signal-to-Noise Ratios (SNRs) across the dataset.

Inefficient Sampling: The recordings were made at a high sampling rate of 44.1 kHz. While suitable for high-fidelity audio, this is excessive for human speech, where the most important acoustic information (phonetic content) is concentrated below 8 kHz. This resulted in unnecessary data volume and computational load.

Limited Speaker Diversity: The initial speaker pool was geographically and linguistically homogeneous, posing a risk of creating a model that would not generalize well to different accents or vocal characteristics.

Standardized Data Acquisition

Standardized Sensor and Environment:
To mitigate the challenges, a standardized recording setup was established. The laptop's built-in microphone was chosen as the consistent acoustic sensor for all recordings.

Speaker Demographics: To enhance the model's generalizability, the final dataset includes a more diverse set of speakers.

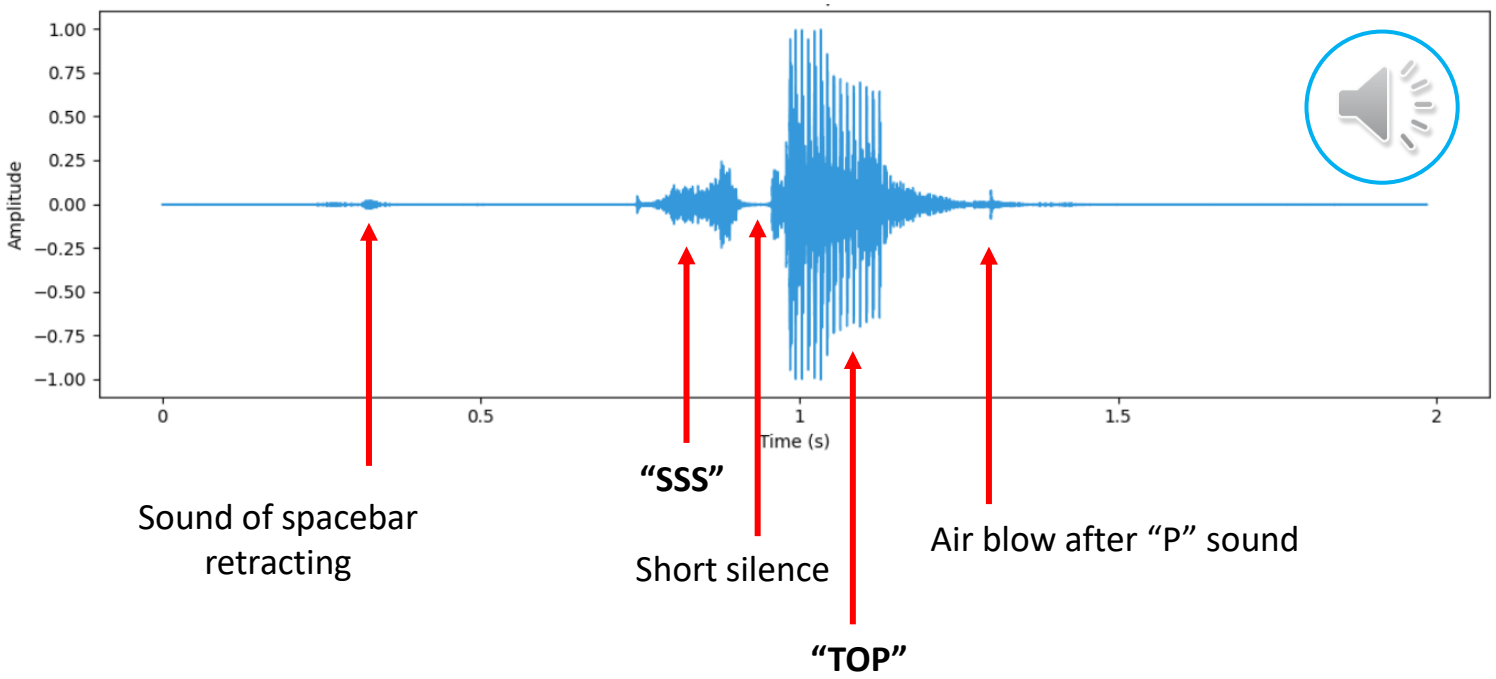
- Speaker 1: Male, India
- Speaker 2: Male, India
- Speaker 3: Male, India
- Speaker 4: Female, Egypt
- Speaker 5: Female, India
- Speaker 6: Male, India
- Speaker 7: Male, Myanmar

Optimized Sampling Rate: 16 kHz Sampling Rate was chosen as an industry standard for speech processing, efficiently capturing all relevant phonetic information while significantly reducing data volume and computational load. (Further down-sampled later.)

The acquisition script captures 2-second monophonic (CHANNELS=1) audio at an optimized 16 kHz (RATE=16000) sampling rate and 16-bit depth (FORMAT=pyaudio.paInt16). It processes the audio stream with a buffer size of 1024 frames (CHUNK=1024)

Raw Signal

Person 1: “STOP”

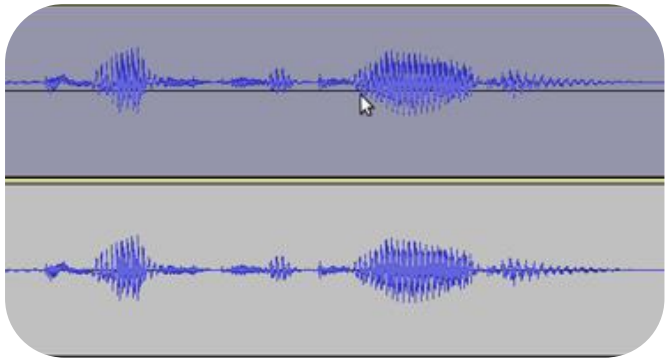


3. Signal Preprocessing

DC Offset

DC offset:
DC offset is an unwanted phenomenon where the average amplitude of a signal is not centered at zero. In a digital audio signal, this means the **mean value** of the samples is not equal to zero.

This is often caused by a hardware fault in the recording device or an improper connection to the microphone, which adds a constant voltage to the signal.



Impact on Processing: Computational Inefficiency: A non-zero mean value can complicate signal processing algorithms, particularly those that assume a zero-mean signal (e.g., FFT).

DC offset - how much?

[2]:
```
import numpy as np

audio_data = np.load("npy_data/audio_data.npy")

def calculate_dc_offset(index):
    audio_sample = audio_data[index]
    dc_offset = np.mean(audio_sample)
    return dc_offset

index = 150
dc_offset = calculate_dc_offset(index)
print(f"DC offset for sample {index}: {dc_offset:.6f}")
```

```
DC offset for sample 150: 0.000034
```

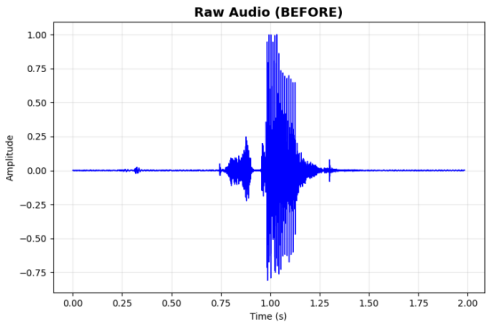
Through initial analysis of the raw signals, it was confirmed that no discernible DC offset was present in the dataset, allowing this preprocessing step to be skipped.

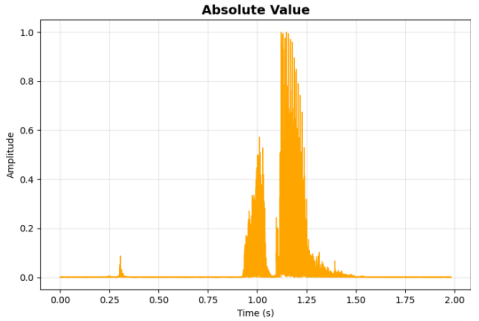
Event Detection

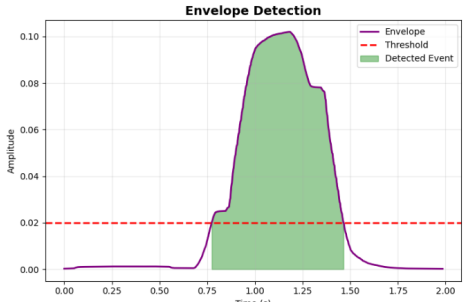
Voice Activity Detection – VAD: The objective of this stage is to automatically segment the raw audio and isolate the core voice command, discarding the surrounding silence and noise. This is a critical preprocessing step that improves the robustness and efficiency of subsequent feature extraction and classification.

Absolute Value: The raw audio signal is first converted to its absolute value. This transforms the bipolar waveform into a unipolar signal that represents the instantaneous energy of the audio, regardless of its positive or negative amplitude.

Envelope Detection: The absolute signal is then passed through a low-pass filter to compute its envelope. The provided script uses a simple moving average filter with `win_length` of 8000 samples (0.5s). This envelope effectively smooths out the rapid fluctuations of the signal, providing a clean representation of its short-term energy profile.





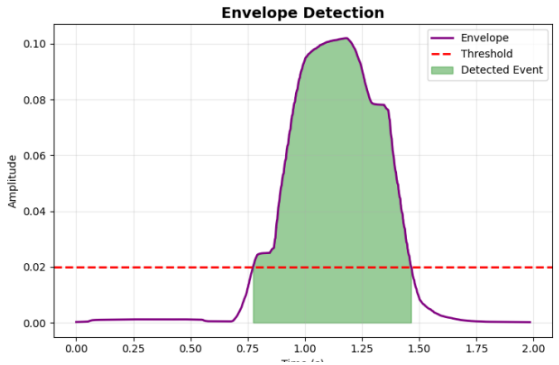


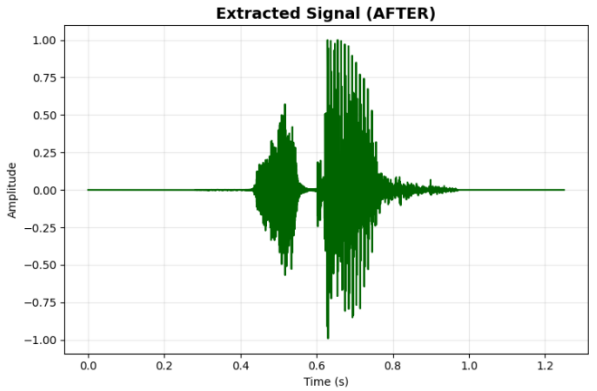
Event Detection & Padding

Thresholding: A fixed amplitude threshold (`THRESHOLD = 0.02`) is applied to the calculated envelope. Any segments of the signal where the envelope's amplitude exceeds this threshold are identified as an "active event" or speech.

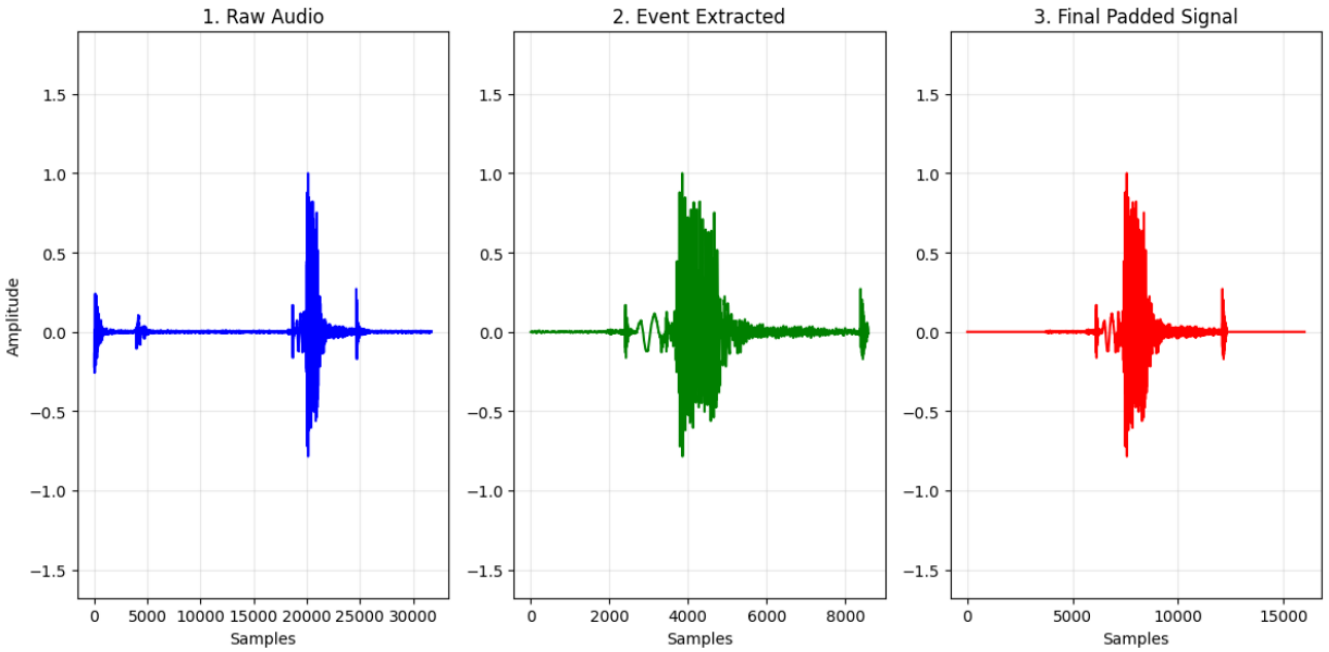
Signal Extraction and Zero Padding: The start and end points of the detected event are used to extract the relevant audio segment from the original raw signal. This extracted signal is then either padded with zeros or truncated to a standardized length, ensuring consistent input dimensions for the feature extraction pipeline.

Parameter Tuning: The `win_length` for envelope detection and the `THRESHOLD` were empirically tuned. The threshold value was chosen by trial and error, visually inspecting the plots and confirming that the extracted audio contained the full command without significant silence or noise. This iterative process ensures the VAD is robust to the specific characteristics of the dataset.





Clean Signal – Simplified preprocessing pipeline

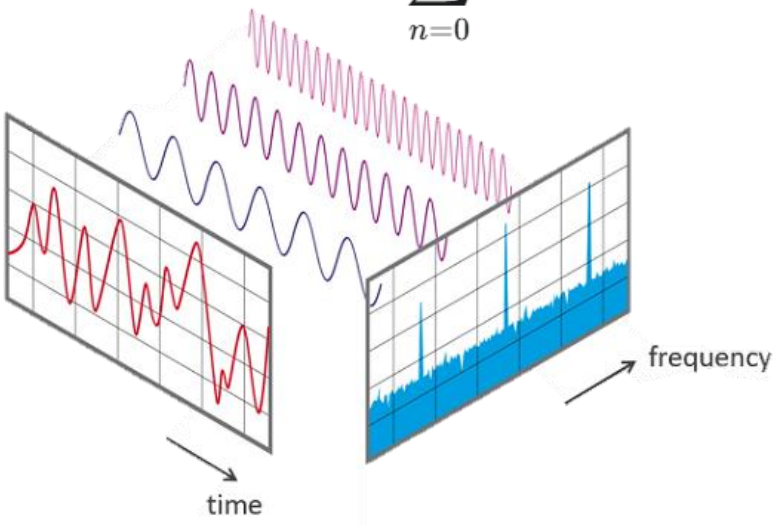


4. FFT approach for classification

Fast Fourier Transform

The **Fast Fourier Transform (FFT)** is an efficient algorithm for computing the Discrete Fourier Transform (DFT). It mathematically converts a signal from its original time domain to a frequency domain, revealing the frequency components present in the signal.

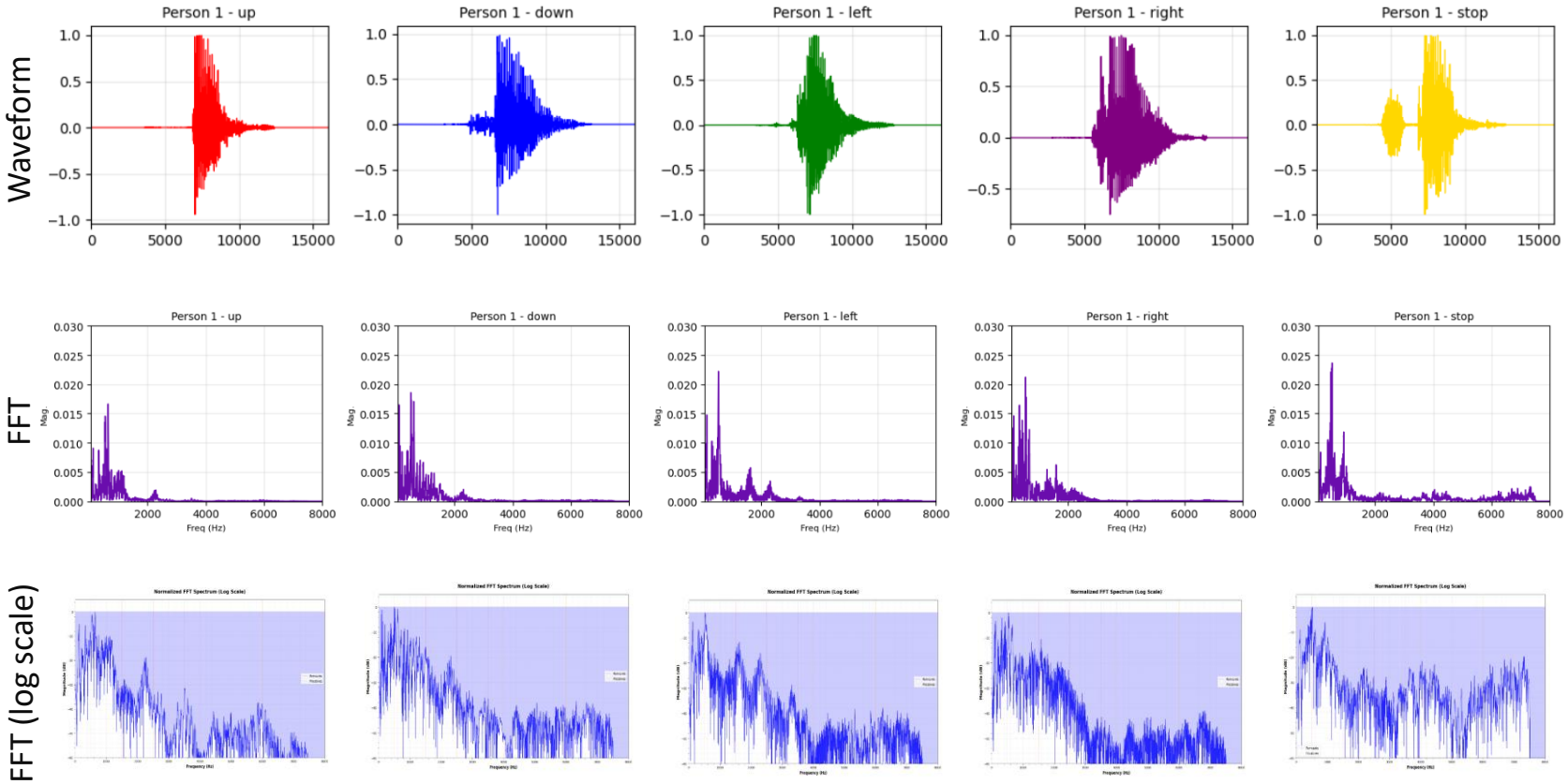
$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N}$$



- X_k is the output of the FFT at a specific frequency index, k. It's a complex number representing the magnitude and phase of that frequency.
- N is the total number of samples in the signal.
- x_n is the value of the signal at the time index, n.
- k is the current frequency bin index, ranging from 0 to $N-1$.
- n is the current time sample index, ranging from 0 to $N-1$.
- $e^{-i2\pi kn/N}$ is a complex exponential. This is the core of the transform, essentially comparing the signal to a series of sine and cosine waves.

Fast Fourier Transform

The FFT was implemented using the `fft` function from the `scipy.fft` module, a part of the SciPy library.



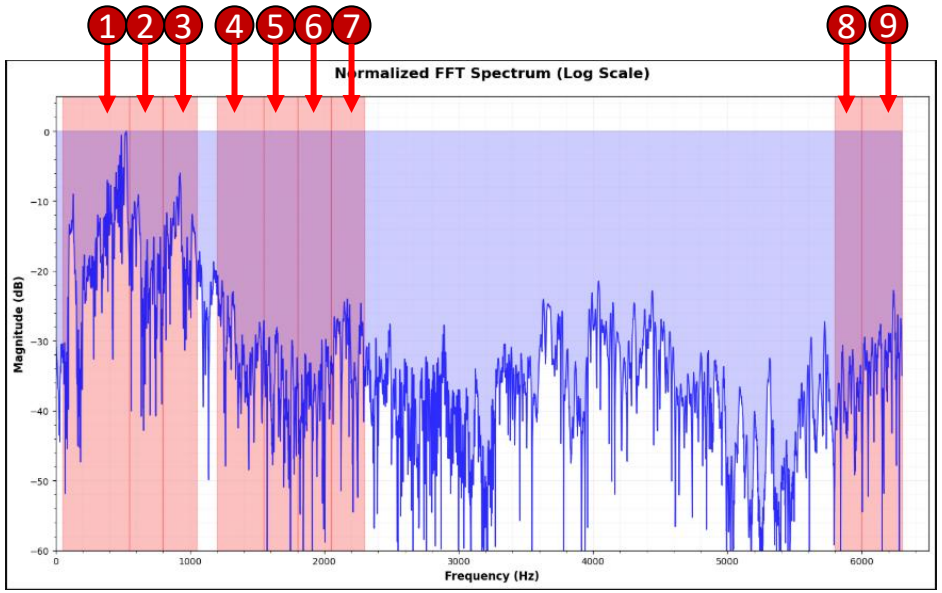
Feature Extraction

From FFT to a Feature Vector:

The goal of this phase is to transform the frequency spectrum of each audio command into a fixed-size, numerical representation that a machine learning classifier can understand. Instead of using the entire FFT spectrum, which is high-dimensional and noisy, we extract key statistical features.

Regions of Interest (ROI):

The code defines **nine** specific frequency bands or Regions of Interest (ROIs). For each ROI, the code isolates the corresponding FFT magnitude values.



- These regions were manually selected to capture the most significant frequency components in human speech, which are crucial for distinguishing between commands.
- Eliminating ROIs 8 and 9, which contain the highest frequencies in our model, resulted in a 6-7% drop in model accuracy.
- The sampling rate was down-sampled to **12.6 kHz** because our analysis showed no discriminative information above this range.

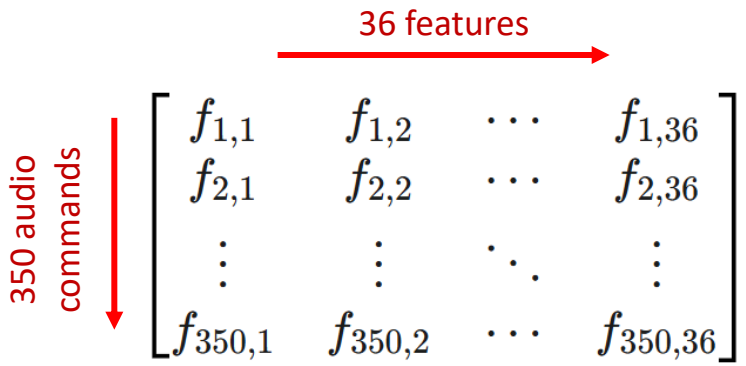
Feature Extraction

Statistical Feature Computation:
Within each ROI, four key statistical features are computed to represent the spectral characteristics of that frequency band.

- **Mean:** The average magnitude, representing the overall energy in the ROI.
- **Standard Deviation:** The spread of magnitudes, indicating the variability within the ROI.
- **Maximum:** The peak magnitude, highlighting the dominant frequency component.
- **Energy:** The sum of squared magnitudes, a robust measure of signal power in the ROI.

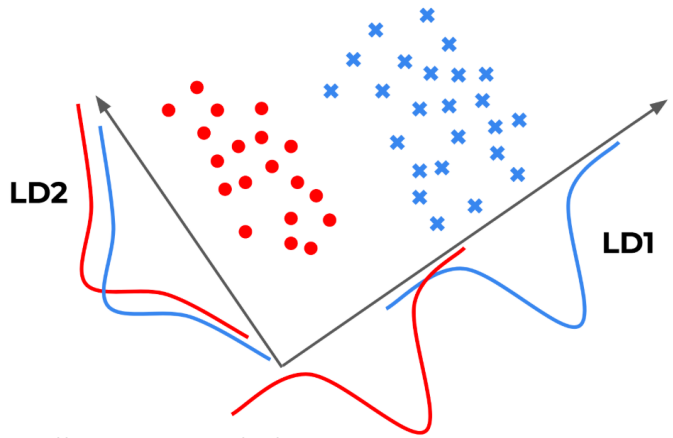
Logarithmic Scaling:
Each of these four features is transformed using a logarithmic scale. This is a standard practice in signal processing and machine learning to handle features that may span several orders of magnitude.

Result: This process generates a feature vector of a fixed size (9 ROIs × 4 features/ROI = 36 features) for each audio command. This vector is the raw input for the next stage.



Dimensionality Reduction: Linear Discriminant Analysis (LDA)

After feature extraction, the 36-dimensional feature vectors are a rich representation of the audio commands.
However, to simplify visualization and potentially improve classifier performance, we apply a dimensionality reduction technique called **Linear Discriminant Analysis (LDA).**



https://www.georgeho.org/lda/

Standardization:
Before applying LDA, the features are first standardized using a `StandardScaler`. This is a critical preprocessing step that ensures each feature has a zero mean and unit variance, preventing features with larger numerical ranges from disproportionately influencing the LDA algorithm.

LDA Objective:

Unlike PCA (Principal Component Analysis), which is an unsupervised method, LDA is a **supervised algorithm.**

Its primary goal is not just to reduce dimensions but to find a projection that **maximizes the separability between the different classes (commands).**

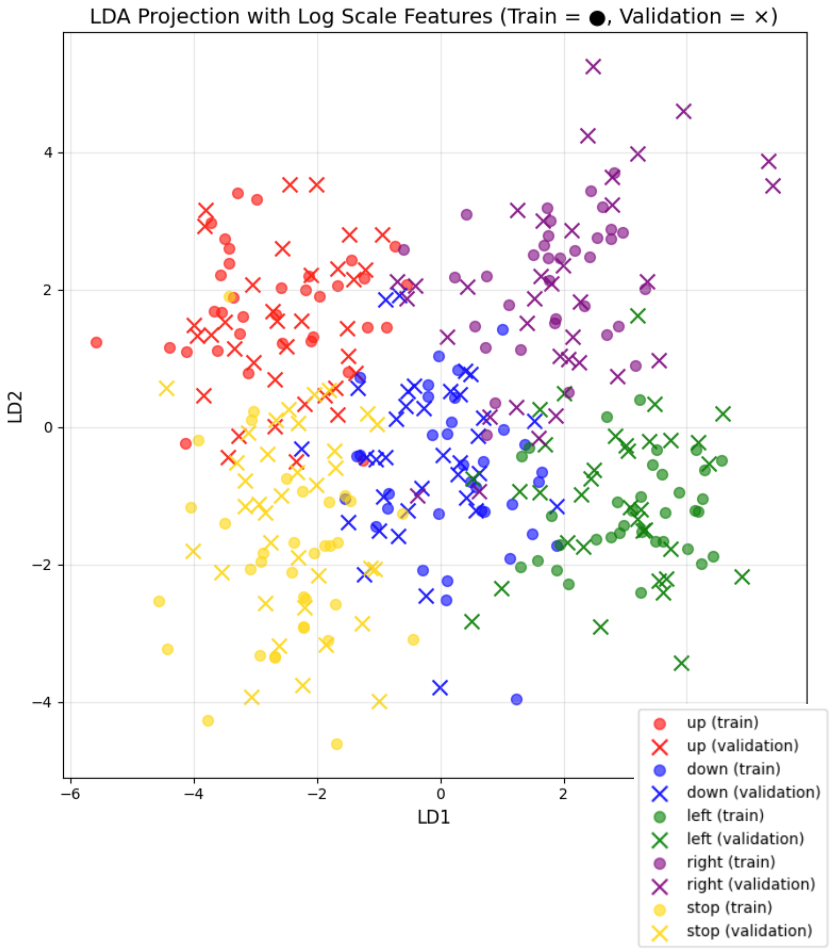
It projects the high-dimensional data onto a new, lower-dimensional subspace where the between-class variance is maximized, and the within-class variance is minimized.

Dimensionality Reduction: Linear Discriminant Analysis (LDA)

Implementation & Visualization:

- The code applies LDA to project the 36-dimensional feature vectors onto a 2-dimensional space `(n_components=2)`
- The resulting 2D data is then visualized on a scatter plot, showing the projection for both the training and validation set.
- This visualization was a crucial part of our iterative validation process. By observing the separation of the classes in the validation set, we could confirm the effectiveness of our chosen ROIs and refine our feature engineering to achieve better class separability.

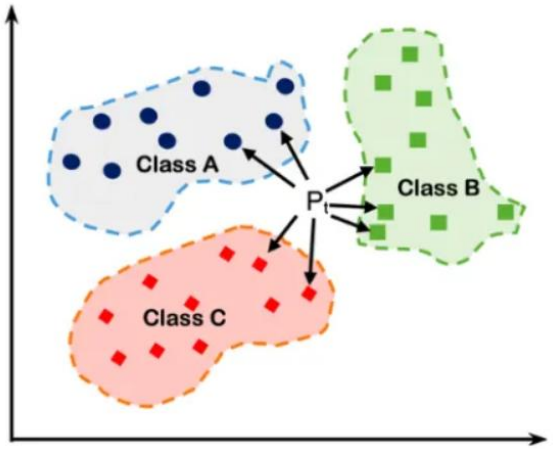
- **Implementation:** LDA was implemented using the `LinearDiscriminantAnalysis` class from the `sklearn.discriminant_analysis` module of the scikit-learn library. [1]



Classification: (i) k-Nearest Neighbors (k-NN) Classifier

k-NN:

- k-NN is a non-parametric, instance-based supervised learning algorithm used for classification.
- The core principle is simple: to classify a new data point, the algorithm finds the **'k' nearest data points** (neighbors) from the training set in the feature space and assigns the new point to the most common class among those neighbors.
- The "nearest" is typically determined using a distance metric, such as Euclidean distance. The number of neighbors, 'k', is the primary hyperparameter that must be tuned.

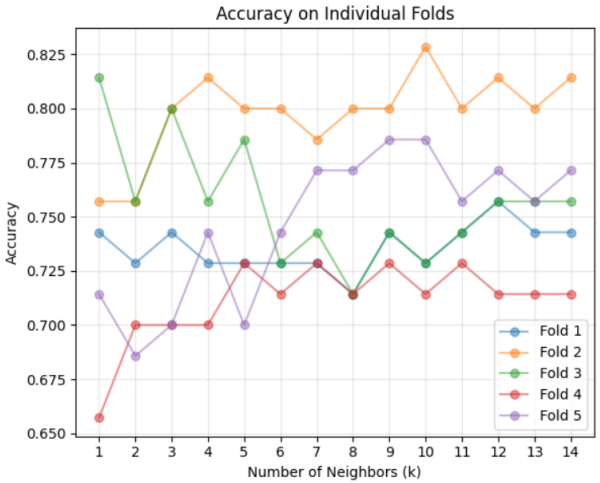


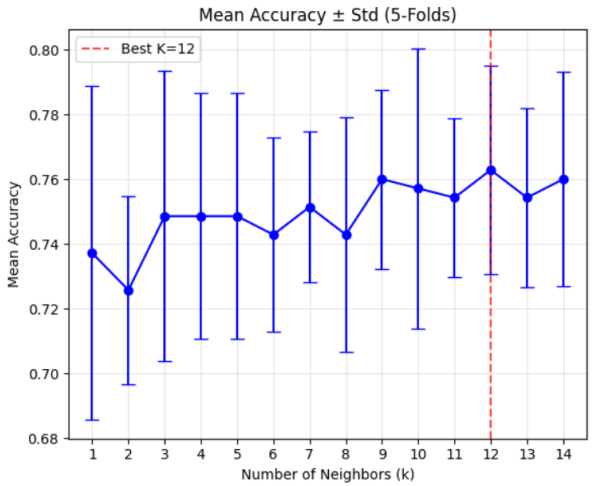
Hyperparameter Tuning: Stratified k-Fold Cross-Validation was used in development stages. In this technique, the dataset is partitioned into k (in this project, k=5) subsets, or "folds". The stratification ensures that each fold maintains the same proportional representation of classes as the original dataset.

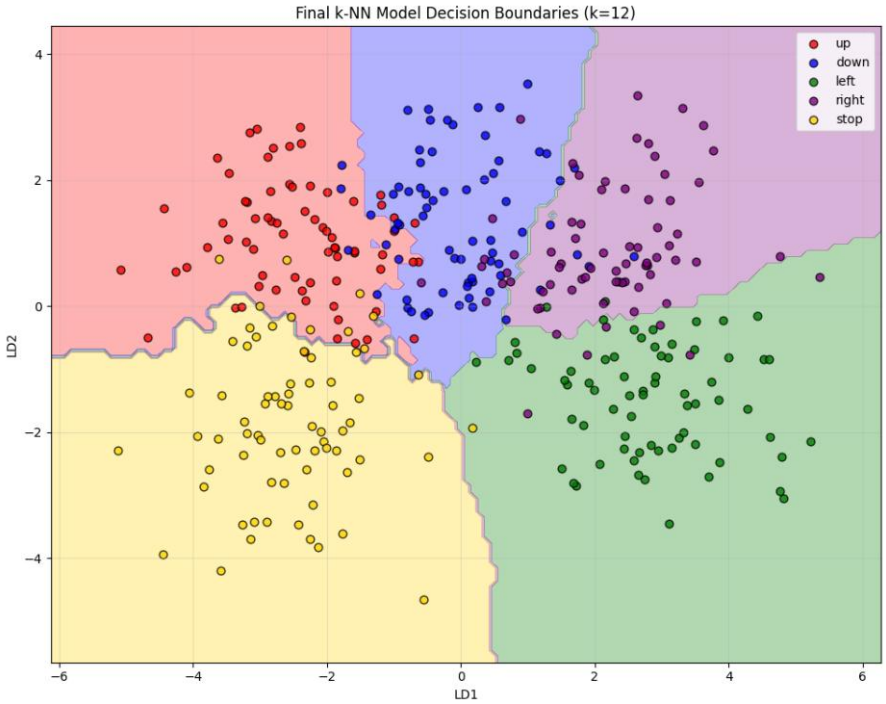
Implementation:

- Stratified k-fold cross-validation was implemented with the `StratifiedKFold` class from `sklearn.model_selection`.
- The k-NN classifier was implemented using the `KNeighborsClassifier` class from the `sklearn.neighbors` module.

(i) k-NN Classifier - Train Data

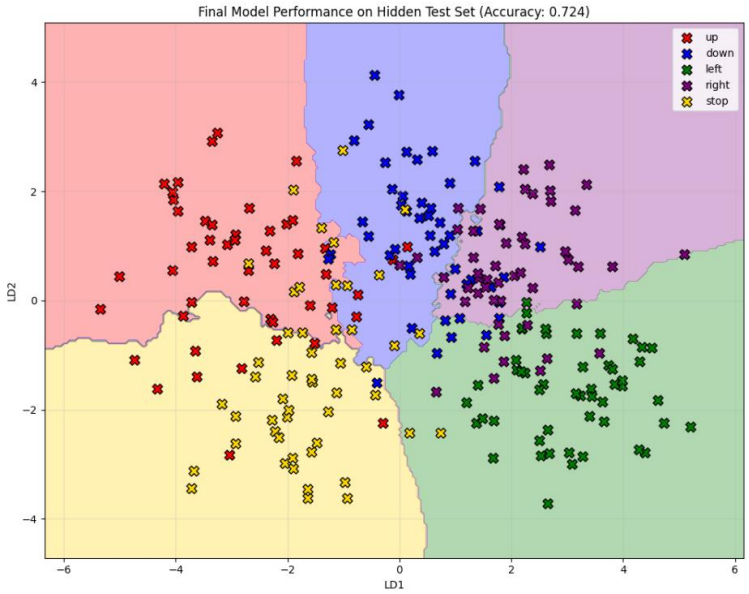


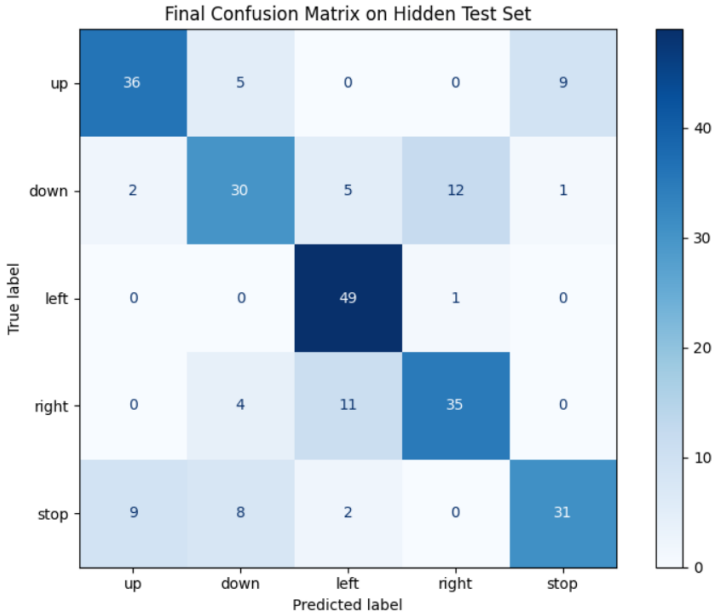




After Stratified 5-Fold Cross-Validation we take the optimal value (BEST_K = 12) found in the first step and train a single, definitive k-NN model on the entire development dataset.

(i) k-NN Classifier - Test Data





- This stage evaluates the final, trained k-NN model against a hidden test set.
- It loads the saved scaler, LDA, and k-NN models, processes 250 new audio files, and uses the same pipeline to predict the command for each file.
- The model's final performance shows an overall accuracy of **72.40%** on this unseen data.
- The results are detailed in a classification report and a confusion matrix, which visualizes the specific prediction errors.

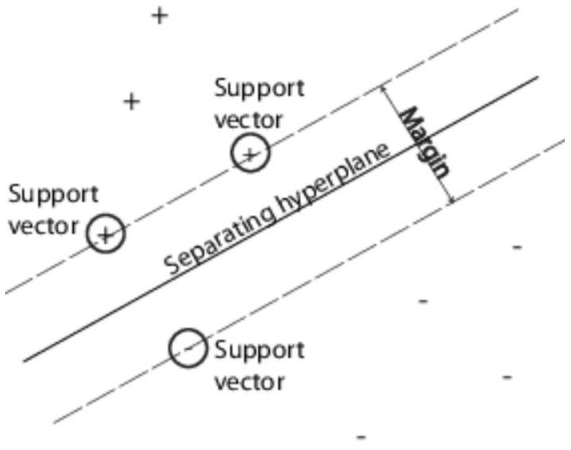
Classification: (ii) Support Vector Machine (SVM) Classifier

SVM:

- A Support Vector Machine is a powerful supervised learning algorithm used for classification that works by finding an optimal hyperplane that best separates data points of different classes in the feature space.
- The "optimal" hyperplane is the one that has the largest margin, which is the distance between the hyperplane and the nearest data points from any class.
- These nearest data points that define the margin are called "support vectors."
- The primary hyperparameter for a linear SVM is the regularization parameter, 'C', which controls the trade-off between achieving a low training error and maximizing the margin. [2]

Hyperparameter Tuning:

Stratified k-Fold Cross-Validation was used to tune the regularization parameter 'C', systematically testing a range of values to find the optimal performer, which was determined to be C=0.01 .

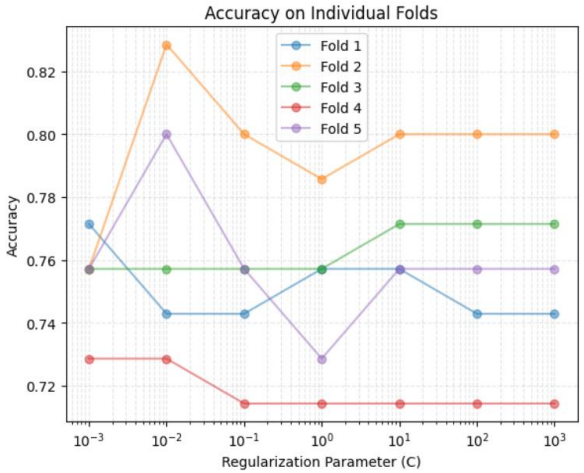


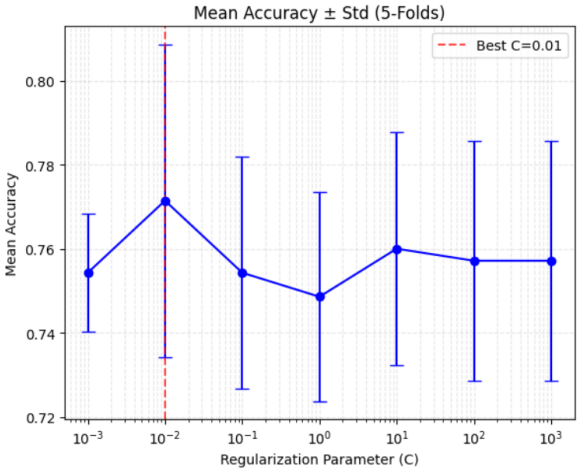
https://www.mathworks.com/help/stats/support-vector-machines-for-binary-classification.html

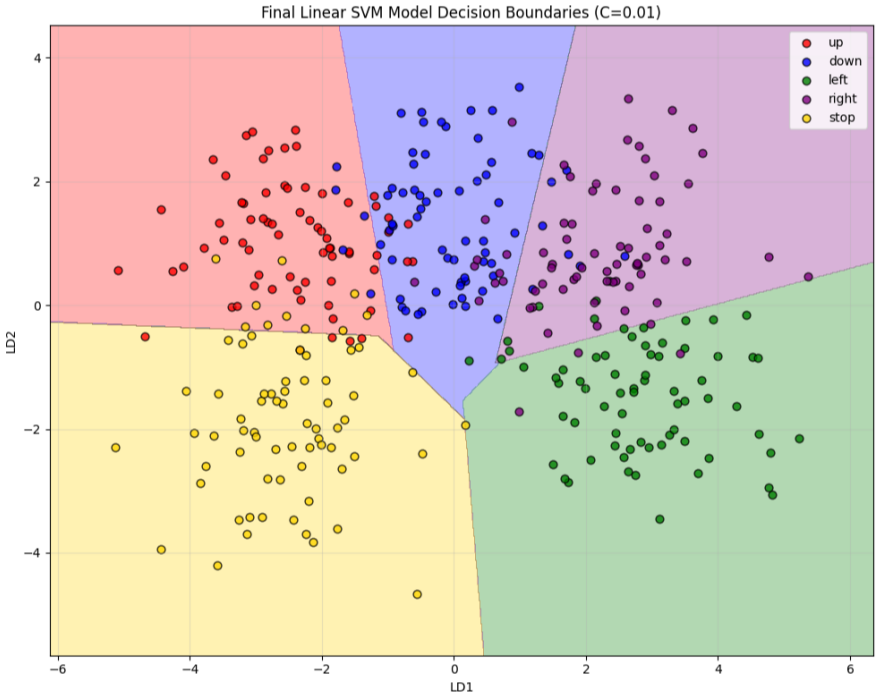
Implementation:

- Stratified k-fold cross-validation was implemented with the `StratifiedKFold` class from `sklearn.model_selection`.
- The SVM classifier was implemented using the `SVC` class from the `sklearn.svm` module, configured with a 'linear' kernel.

(ii) SVM Classifier - Train Data

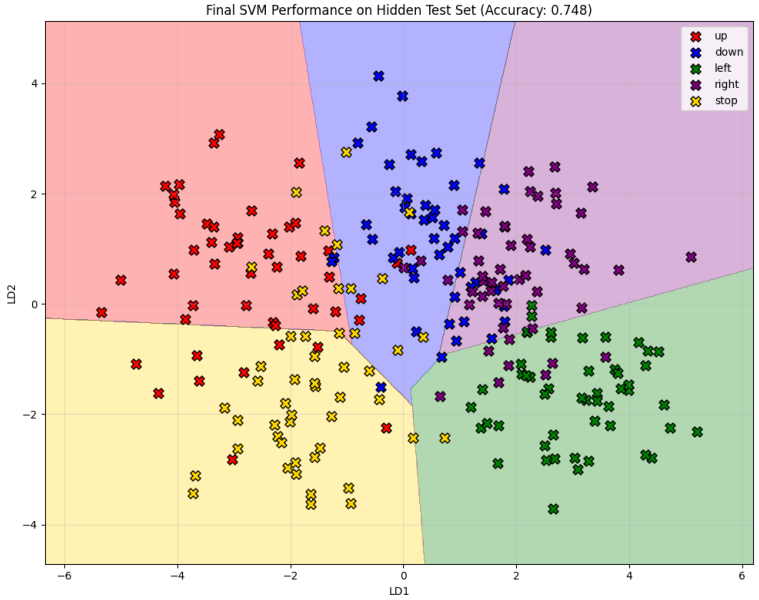


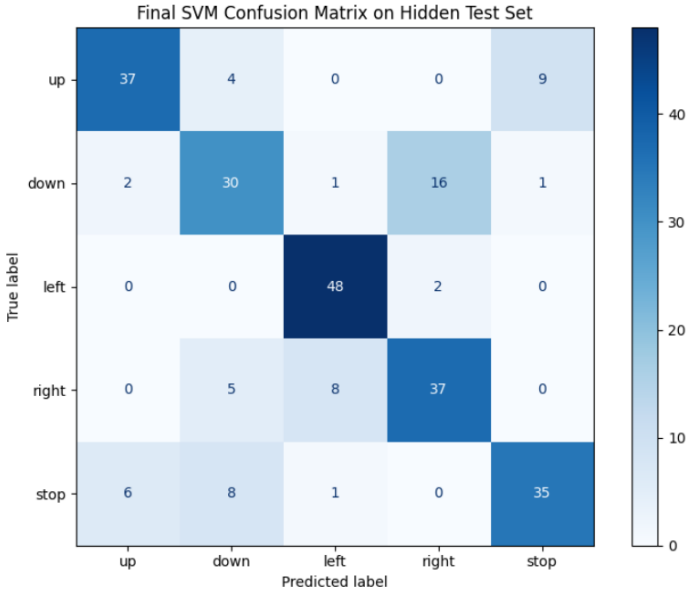




After Stratified 5-Fold Cross-Validation we take the optimal value (Best C: 0.01) found in the first step and train a single, definitive SVM model on the entire development dataset.

(ii) SVM Classifier - Test Data



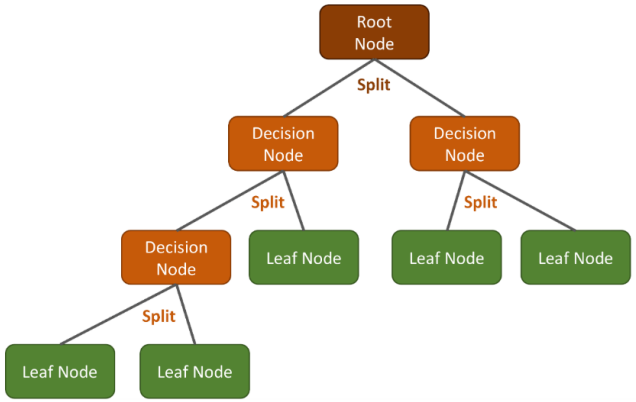


- This stage evaluates the final, trained SVM model against a hidden test set.
- It loads the saved scaler, LDA, and SVM models, processes 250 new audio files, and uses the same pipeline to predict the command for each file.
- The model's final performance shows an overall accuracy of **74.80%** on this unseen data.
- The results are detailed in the confusion matrix, which shows a very high accuracy for the 'left' command.

Classification: (iii) Decision Tree Classifier

Decision Tree::

- A Decision Tree is a supervised learning algorithm that uses a tree-like model of decisions.
- It works by learning simple, hierarchical decision rules inferred from the data features to predict the target class.
- The model splits the data into smaller subsets based on feature values, with each internal node representing a "test" on a feature and each leaf node representing a final class label.
- Key hyperparameters like `max_depth` (the longest path from the root to a leaf) and `min_samples_leaf` (the minimum number of samples required to be at a leaf node) are tuned to prevent the model from becoming overly complex and overfitting the training data.



FU Berlin, RESEDA, https://blogs.fu-berlin.de/reseda/random-forest

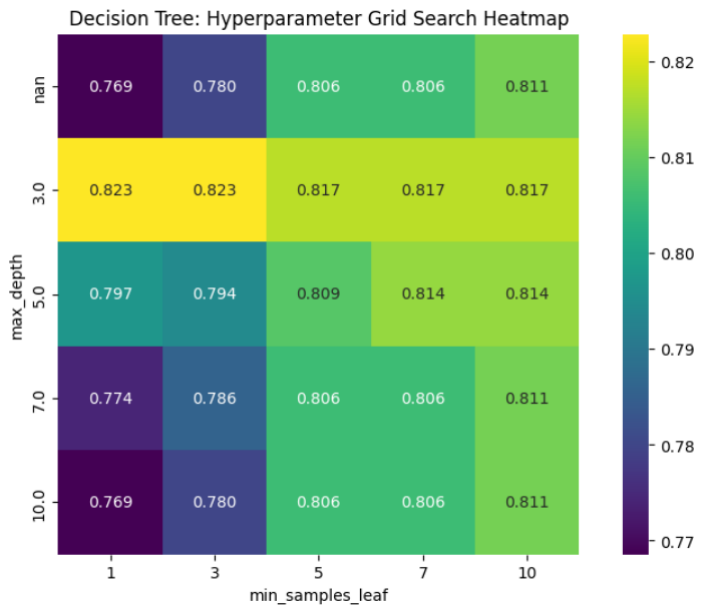
Hyperparameter Tuning:

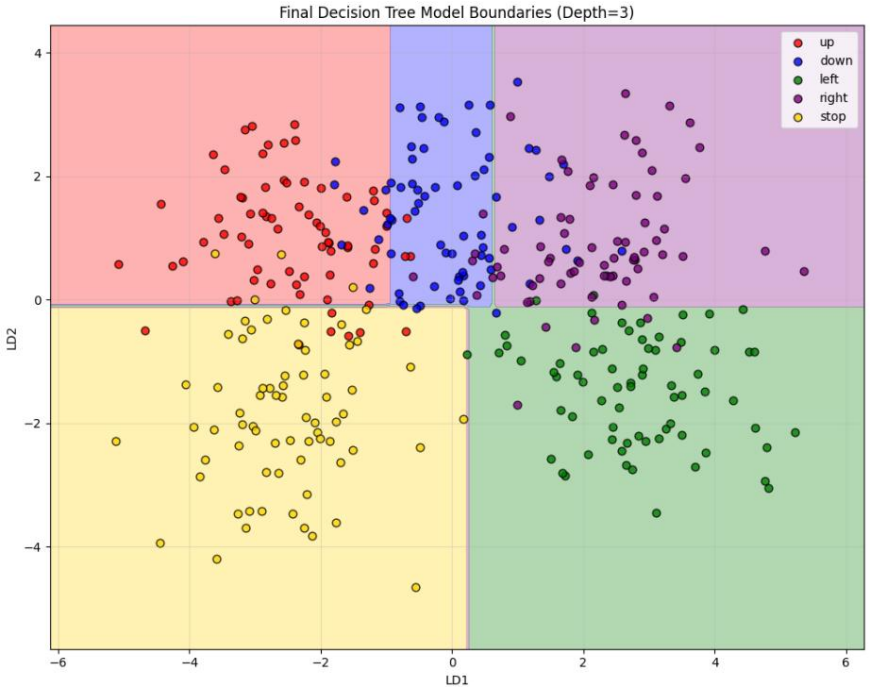
- `GridSearchCV` was used to simultaneously tune multiple hyperparameters and find the optimal combination.
- The best-performing combination was found to be a `max_depth` of 3 and a `min_samples_leaf` of 1.

Implementation:

- The hyperparameter search was performed using the `GridSearchCV` class from `sklearn.model_selection`.The Decision Tree classifier was implemented using the `DecisionTreeClassifier` class from the `sklearn.tree` module.

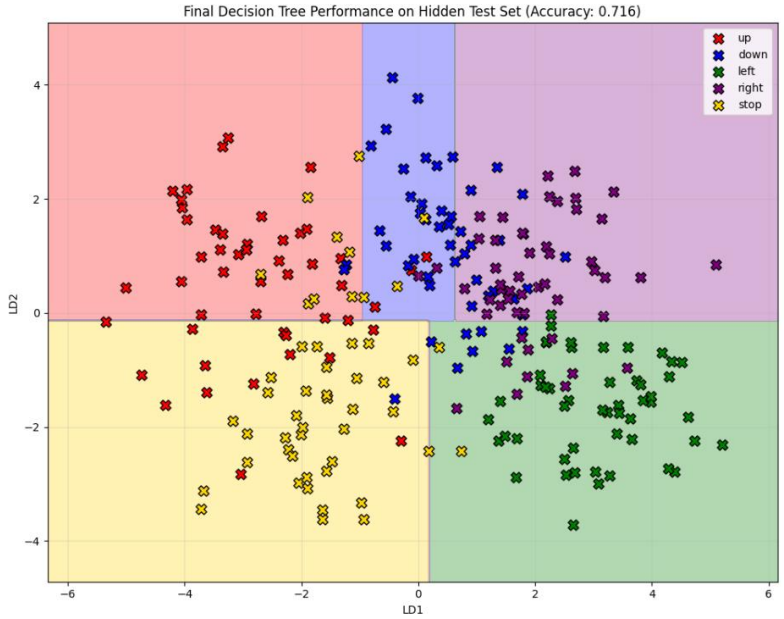
(iii) Decision Tree Classifier – Train Data

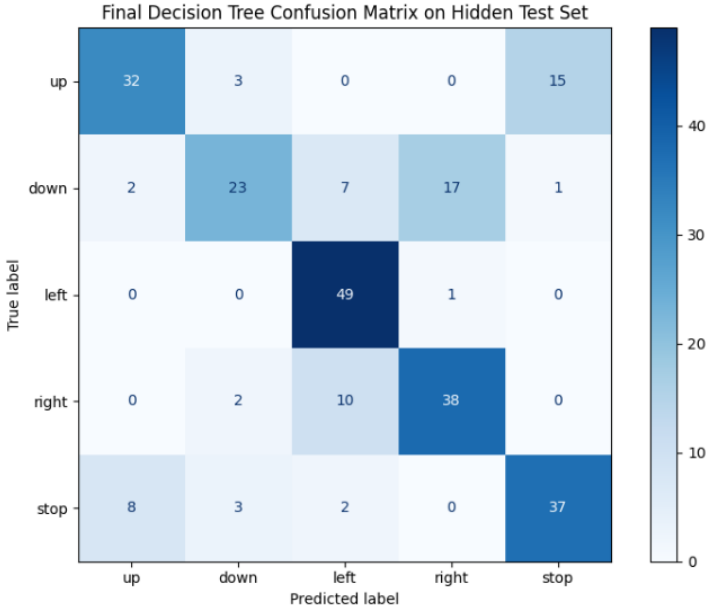




- The heatmap shows the result of a `GridSearchCV`, which used 5-fold cross-validation to find the best hyperparameters for the model .
- Using these optimal parameters, a single, definitive Decision Tree model was trained on the entire development dataset, resulting in the decision boundaries shown on the right .

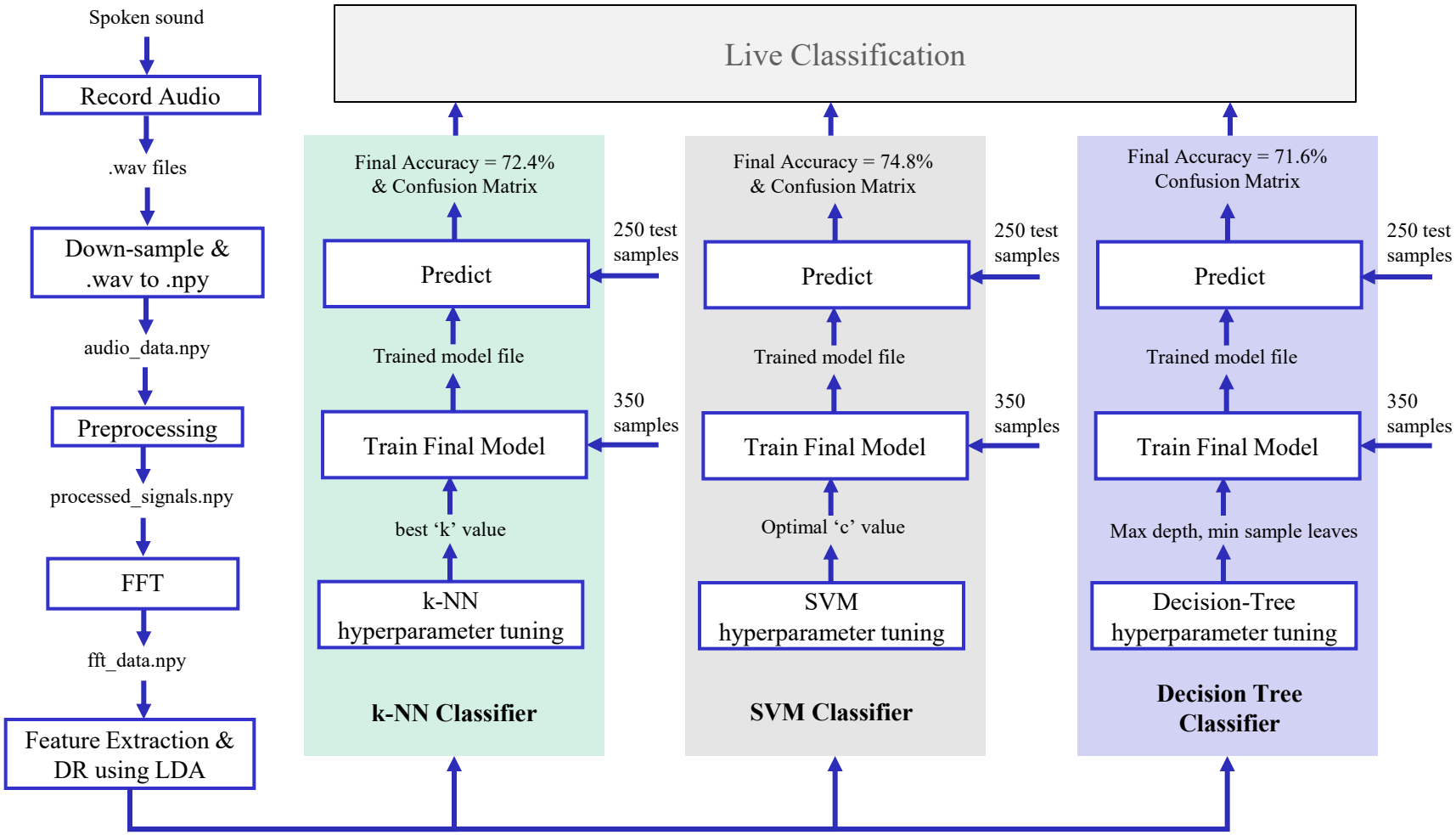
(iii) Decision Tree Classifier – Test Data



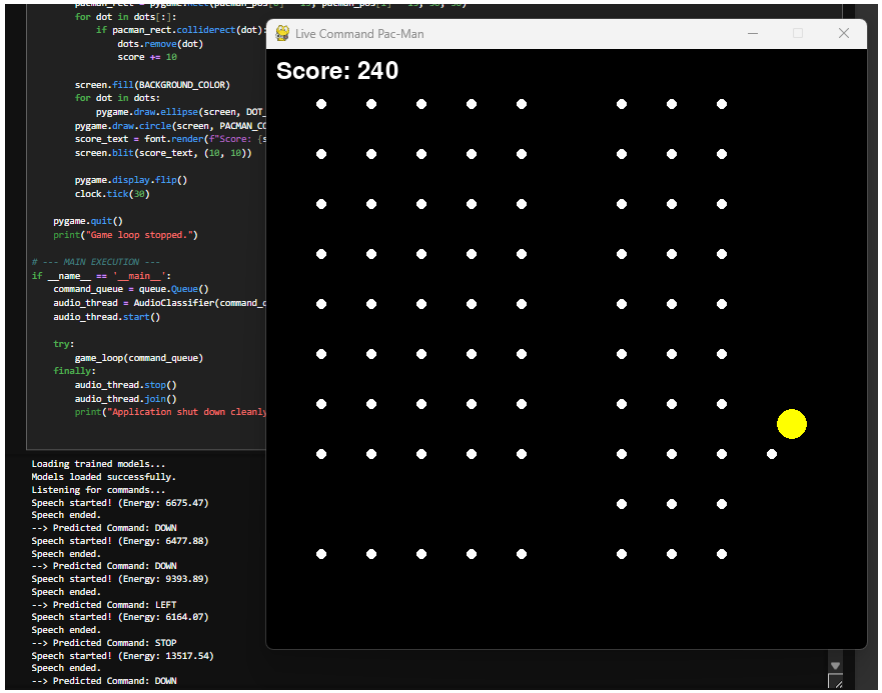


- This stage evaluates the final, trained Decision Tree model against a hidden test set.
- It loads the saved scaler, LDA, and Decision Tree models, processes 250 new audio files, and uses the same pipeline to predict the command for each file.
- The model's final performance shows an overall accuracy of **71.60%** on this unseen data.

End-to-End Classification Pipeline



Live Classification



- Automatic Capture: The system uses Voice Activity Detection (VAD) to listen for and automatically record spoken commands in real-time.
- Live Processing: Each captured command is instantly processed using the full feature extraction and prediction pipeline.
- Visual Feedback: The final predicted command immediately controls a character's movement in an interactive Pygame demo.

Conclusions

- Task: To build a complete system that recognizes spoken commands using FFT to control a live graphical application.
- Successfully built a system achieving 74.80% accuracy with an SVM, though some similar-sounding commands were occasionally confused.
- Our classical ML pipeline was effective, while state-of-the-art deep learning approaches could yield higher accuracy but require more data and complexity.
- Future Work: With more time, we would improve performance by exploring advanced feature engineering using Mel-Frequency Cepstral Coefficients (MFCCs), which are standard in speech recognition and by expanding the dataset.

5. Spectrogram approach for classification

Objective

Objective: Classify four voice commands — up, down, left, right — using spectrogram-based features.

Spectrograms preserve the natural order of sounds (sequential), enabling models to learn patterns of speech or commands. (Major Advantage over FFT)

Spectrogram Computation

- Convert audio to spectrograms.
- Extract (region of interest) ROI-based statistical features.
- Apply dimensionality reduction (LDA / PCA+LDA).
- Classify using KNN and SVM variants.

Sliding Window Concept in Spectrograms

- The audio signal is **split into small overlapping segments** (windows).
- For each window, the **frequency spectrum** is computed (via FFT).
- **Window length** defines how much audio is in each segment.
- **Overlap** means each new window starts before the previous one ends, ensuring smooth coverage of the signal and capturing rapid changes.
- This process **slides** along the entire signal, creating a sequence of frequency spectra → the **spectrogram**.
- **Analogy:** Like looking at a moving scene through a narrow frame — you see part of it at a time, but with overlap, the view changes smoothly.

Spectrograms Normalization

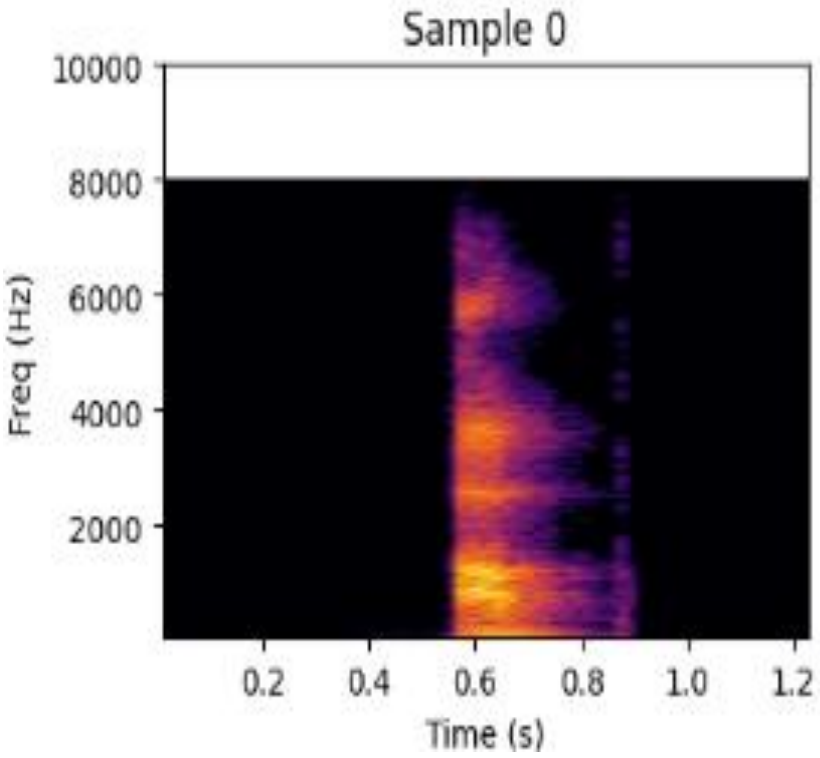


Figure 1 – Spectrogram Sample 0 Before normalization

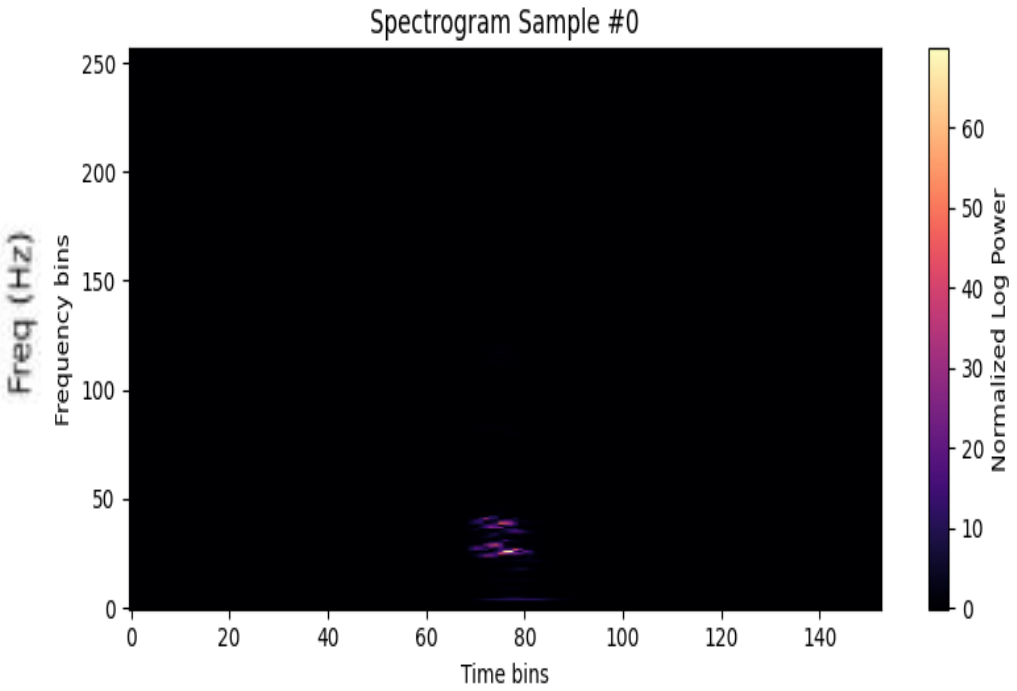


Figure 2 – Spectrogram Sample 0 After normalization

Spectrograms Normalization

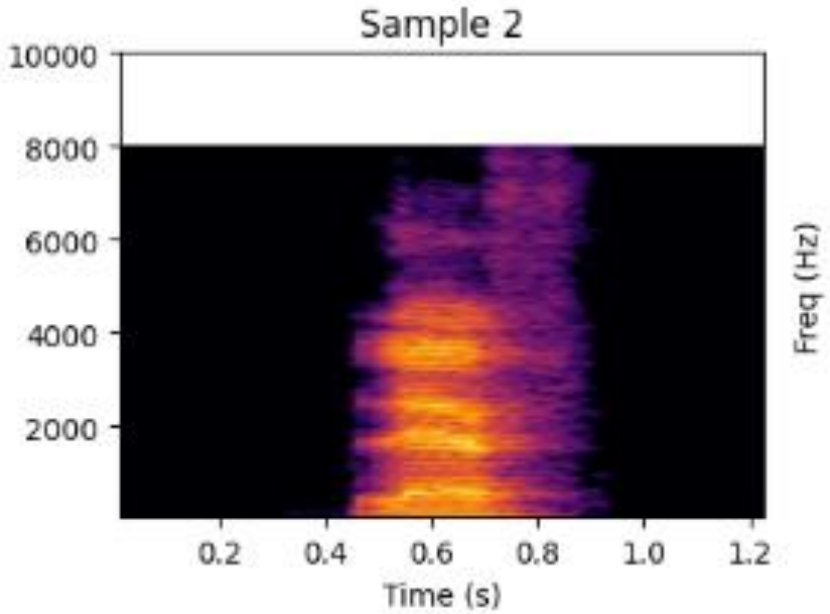


Figure 3 – Spectrogram Sample 2 Before Normalization

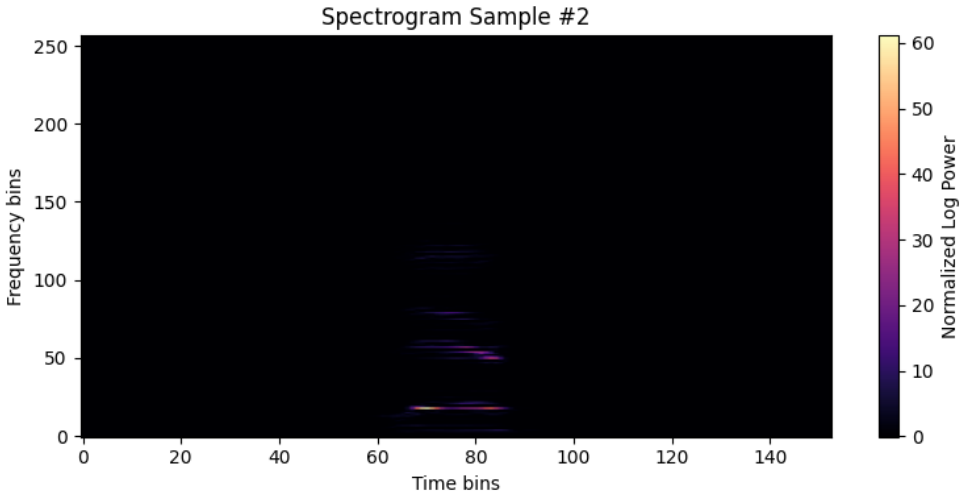


Figure 4 – Spectrogram Sample 2 After Normalization

Preprocessing

- Compute spectrogram (scipy.signal.spectrogram)
- Window length: 512 samples
- Overlap: 384 samples
- Apply **log compression** → reduces dynamic range.
- Apply **Z-score normalization** → enhances feature contrast.
- Output: Consistent **time–frequency representation** for classification.

Feature Extraction

Regions of Interest (ROIs) in Frequency (Hz) and Time(sec):

(50–550), (550–800), (800–1050), (1200–1550),(1550–1800), (1800–2050), (2050–2300),(5800–6800), (6800–7800)

Each ROI is analyzed across sequential time frames, preserving both frequency content and its evolution over time for better pattern recognition.

Feature calculation for each ROI:

- Mean amplitude
- Standard deviation
- Maximum amplitude
- Energy (sum of squares)

• **Final Feature Vector:**

4 statistics × 9 ROIs = 36 features per sample.

Outlier Removal and Feature Summary

Outlier Removal

In test data, for each class:

- Compute class mean & std in LDA space.
- Measure normalized L2 distance.
- Remove points with distance > 3.0 (likely noise or misclassification).

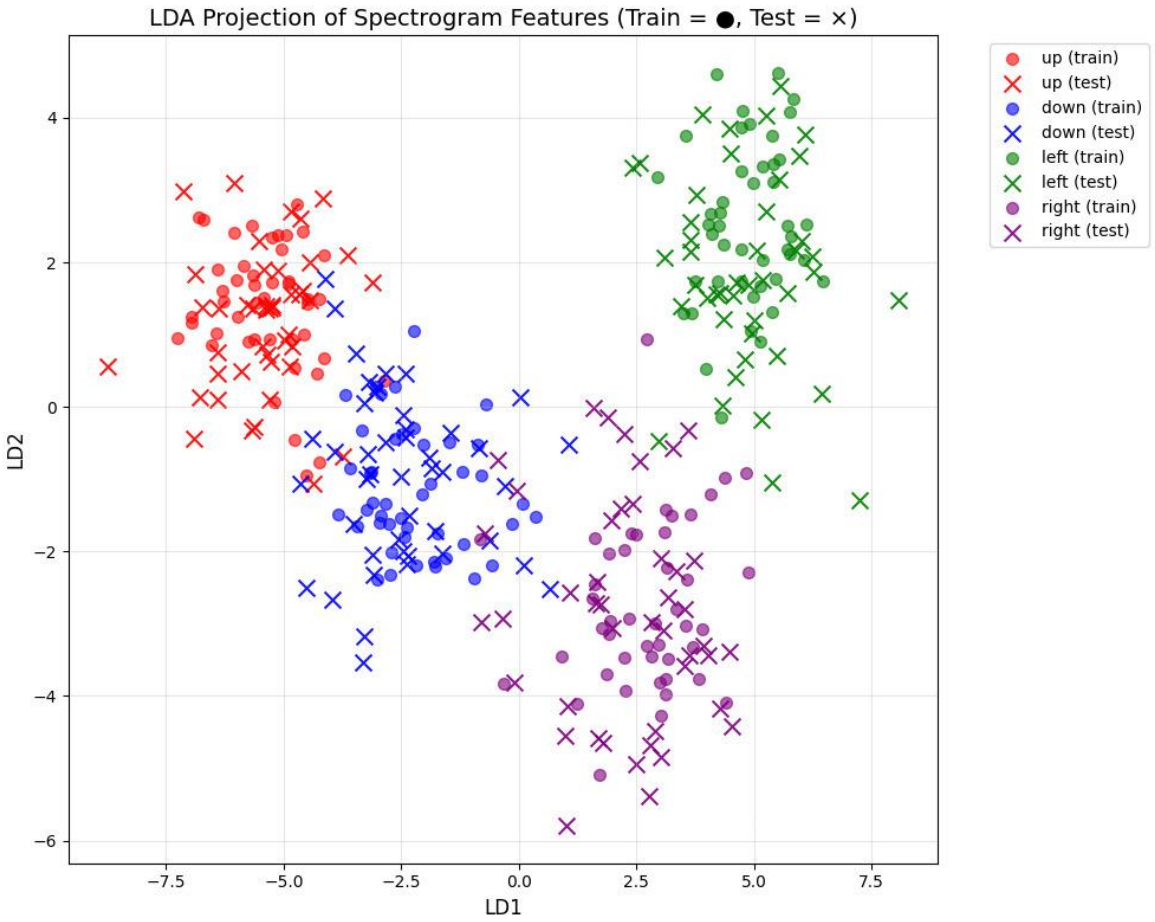
Summary of Parameters

- Features per sample: 36
- Features per ROI: 4 (log mean, log std, log max, log energy)
- Total ROIs: 9
- LDA Components: 2

Dataset & Setup

- **Audio Format:** 16 kHz sampling rate
- A sampling rate of 16 kHz was selected because it provides enough frequency detail for speech and command recognition while significantly reducing computational requirements.
- It is the lowest rate (found by trial and error) that preserves essential information for the model's purpose, making it ideal for real-time and live system performance.
- **Commands:** 4 classes (up=0, down=1, left=2, right=3)
- **Train/Test Split:** 50% / 50%
- **Train Dataset:** 180 Test Dataset: 180

LDA Projection of Spectrogram Features



This shows how well LDA separates commands

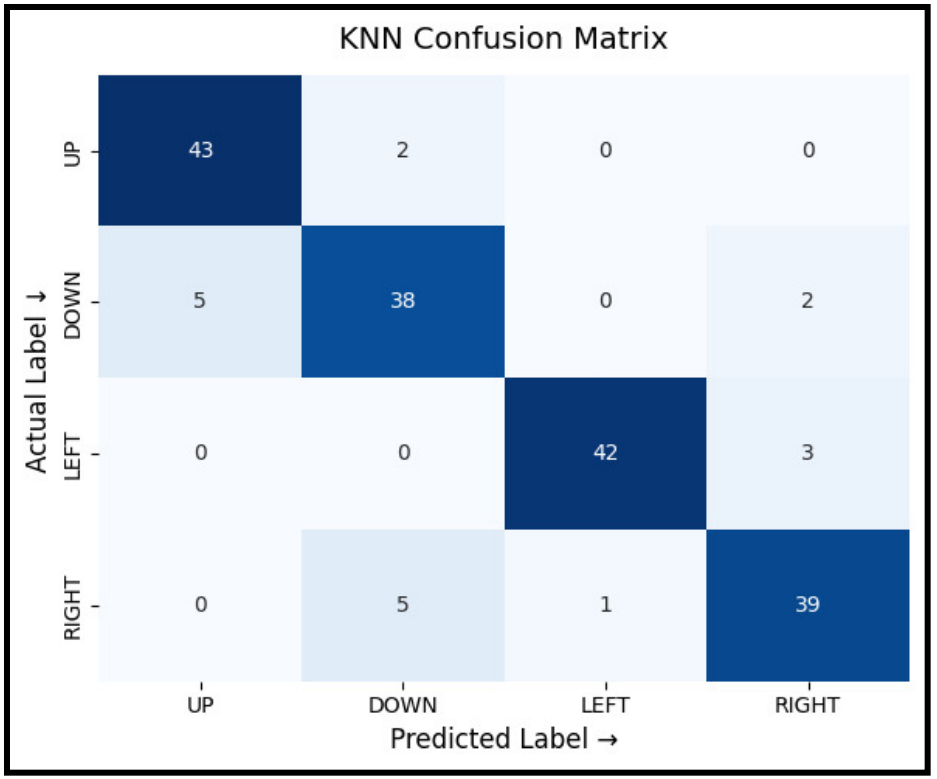
Classifiers & Training

- Input Features: LDA-transformed spectrogram statistics (2D representation).
- KNN: K-Nearest Neighbors
- SVM (Linear): Support Vector Machine
- SVM (RBF): Support Vector Machine Radial Basis Function (RBF) kernel SVM.

Performance Metrics: K-Nearest Neighbors

Input Features: LDA-transformed spectrogram statistics.

Class	Precision	Recall	F1-Score
Up	0.90	0.96	0.92
Down	0.84	0.84	0.84
Left	0.98	0.93	0.95
Right	0.89	0.87	0.88



Hyperparameters

Parameter	Value
algorithm	auto
n_neighbors	3
p	1
weights	distance

Accuracy: 90%

KNN Decision Boundary

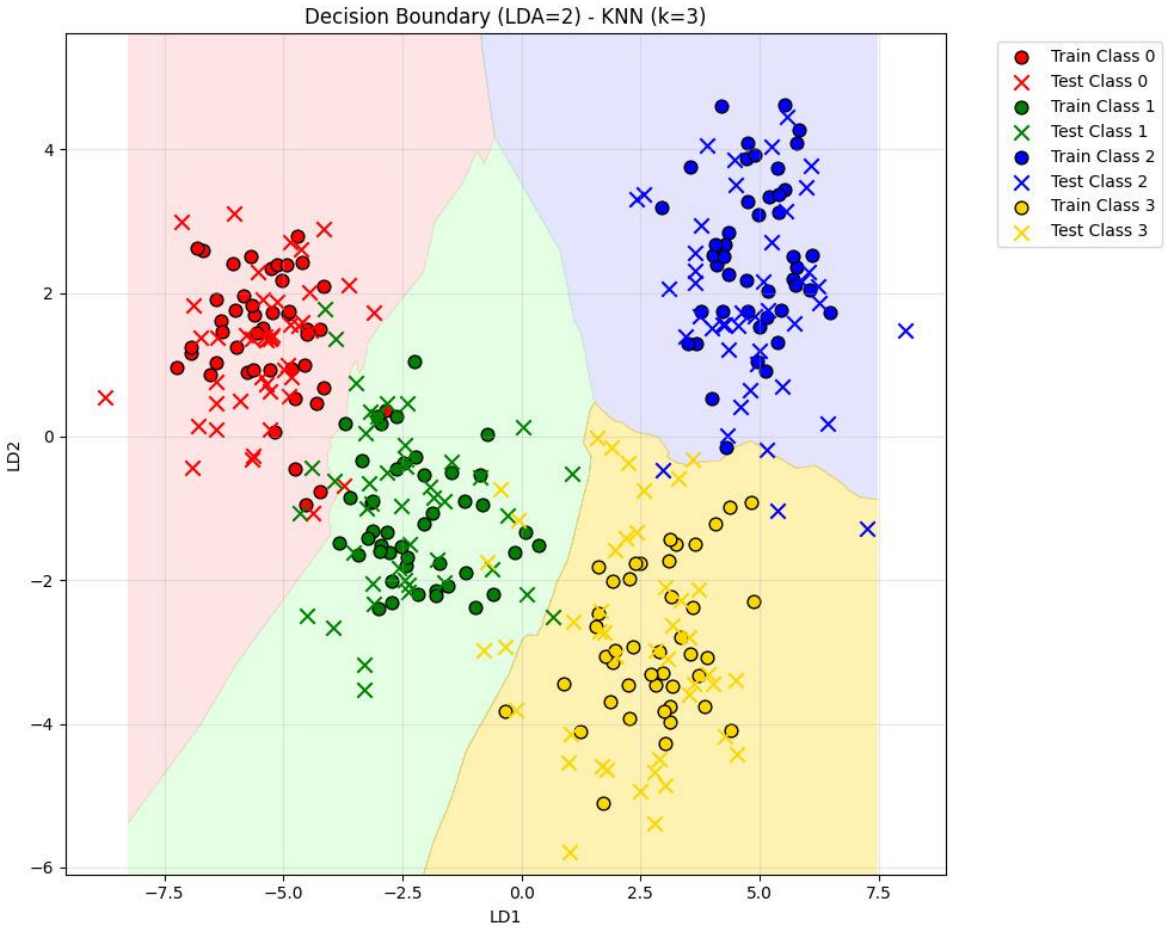


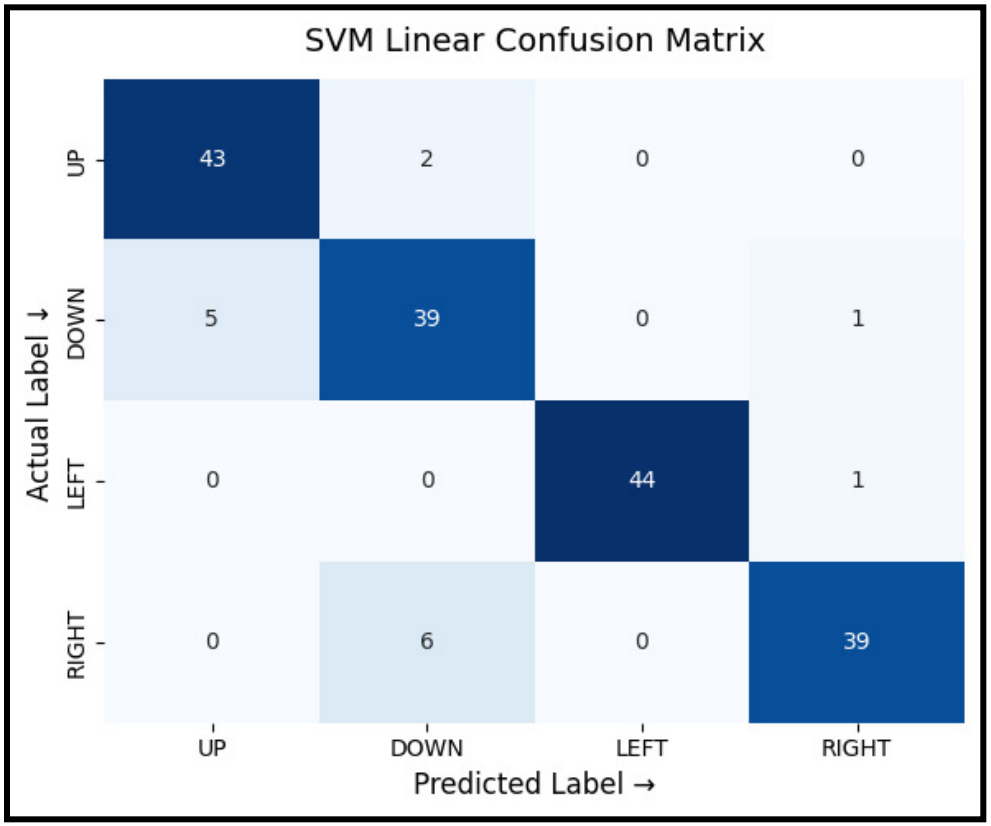
Figure 1 – KNN Decision Boundary (k = 3)

Performance Metrics: SVM Linear Kernel

Class	Precision	Recall	F1-Score
Up	0.90	0.96	0.92
Down	0.83	0.87	0.85
Left	1.00	0.98	0.99
Right	0.95	0.87	0.91

Hyperparameters

Parameter	Value
C	1000
Max_iter	5000
tol	0.001



Accuracy: 91.66%

SVM Linear Decision Boundary

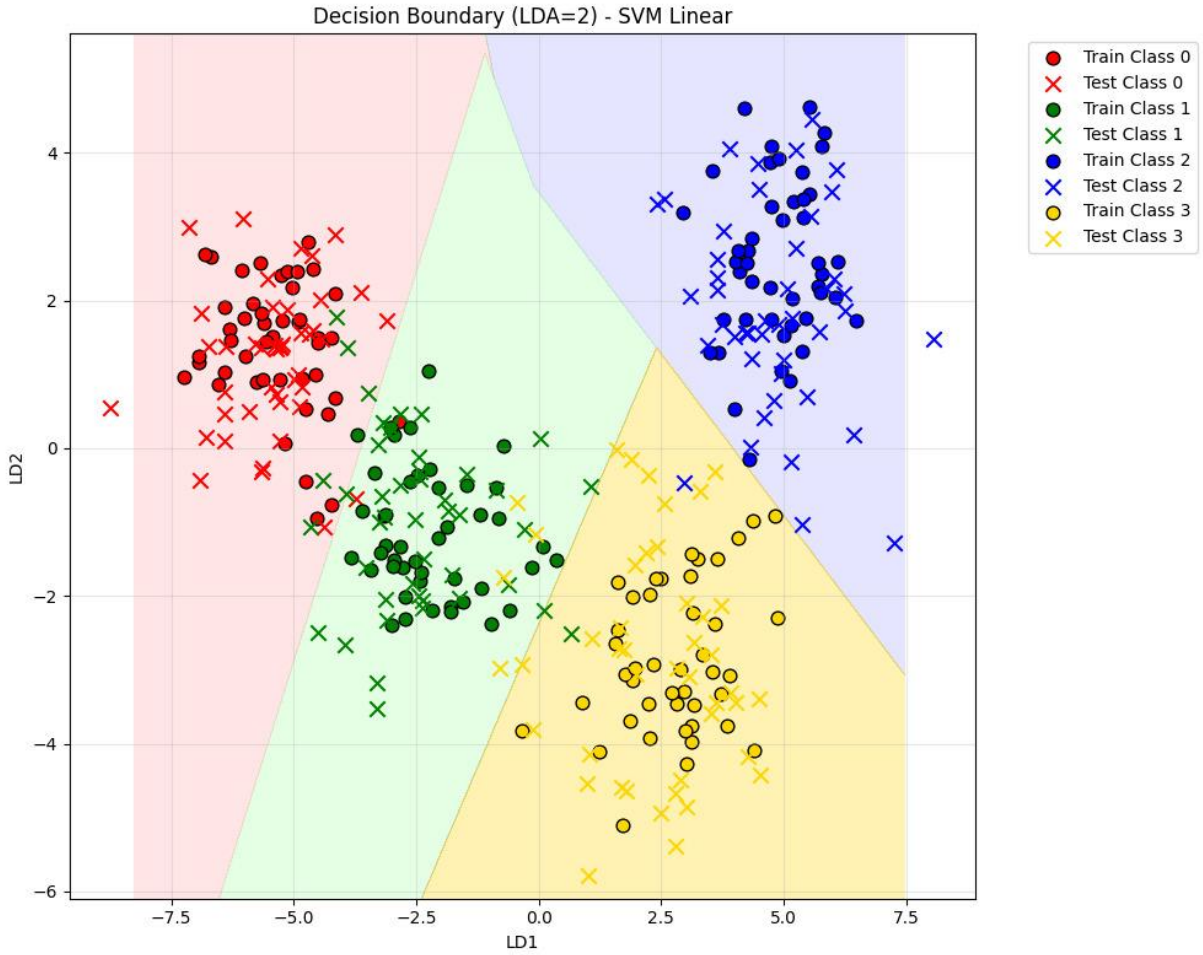


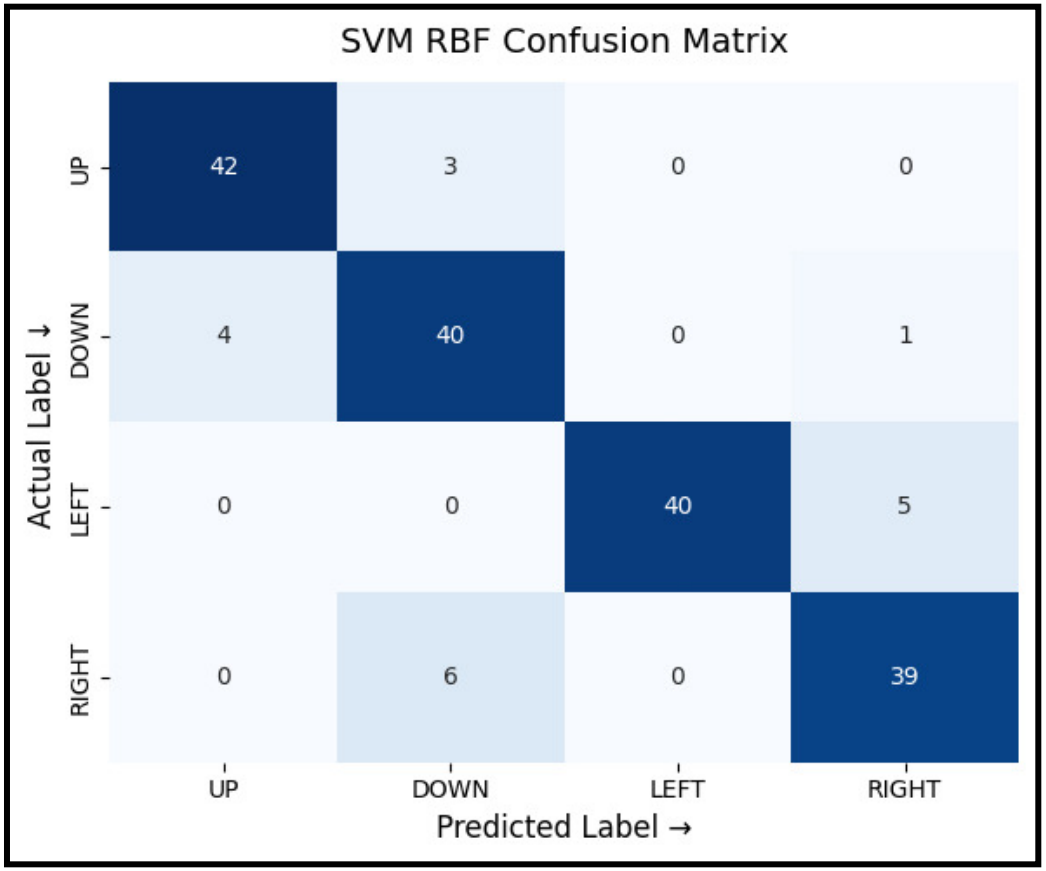
Figure 2 – SVM Linear Decision Boundary

Performance Metrics: SVM (RBF)

Class	Precision	Recall	F1-Score
Up	0.91	0.93	0.92
Down	0.82	0.89	0.85
Left	1.00	0.89	0.94
Right	0.87	0.87	0.87

Hyperparameters

Parameter	Value
C	1
gamma	auto
Max_itertol	1000
tol	0.001



Accuracy: 89.44%

SVM RBF Decision Boundary

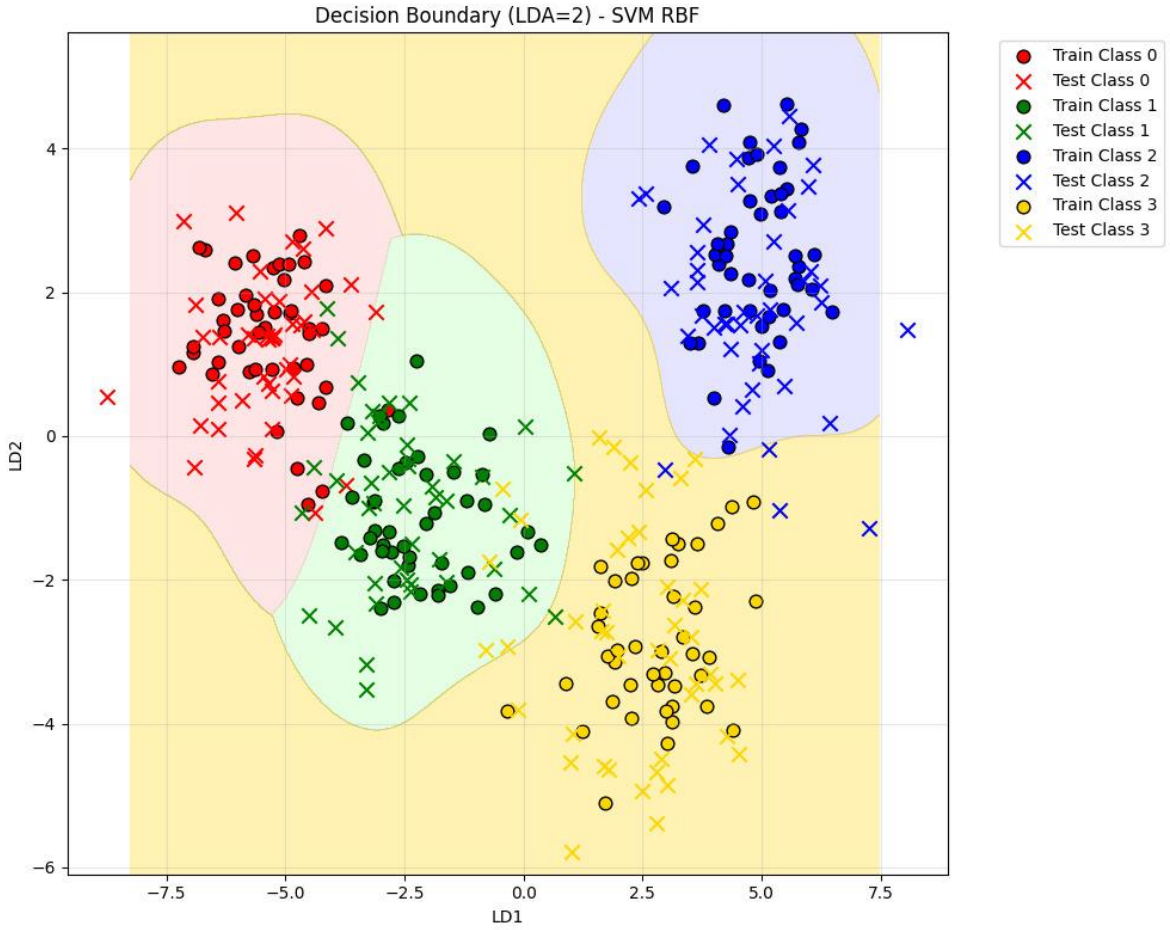


Figure 3 – SVM RBF Decision Boundary (C = 0.01)

How Hyperparameters Were Set

- **KNN** → n_neighbors=3, p=1, weights='distance', algorithm='auto'
  Chosen after small trials, Manhattan distance, and weighted neighbors.

- **SVM (Linear)** → C=1000, kernel='linear', max_iter=5000, tol=0.001
  Large C to reduce underfitting, and more iterations for stable training.

- **SVM (RBF)** → C=0.01, kernel='rbf', gamma='auto', max_iter=1000, tol=0.001
  Small C for stronger regularization, and automatic scaling for gamma.

- Hyperparameters were chosen through a mix of **default values and small trials** to find a good balance between **accuracy and training efficiency**.

Conclusion

- SVM Linear turned out to be best model among KNN, SVM Linear and SVM RBF achieved the highest accuracy (91.66%).
- KNN performance was competitive but slightly lower, potentially due to sensitivity to class boundaries in the reduced 2D LDA space.
- ROI-based spectrogram statistics + LDA yields strong separation between commands.

Model	Accuracy
KNN	90%
SVM Linear	91.66%
SVM RBF	89.44%

Table: Accuracy matrices

6. Reduced spectrogram approach for classification

PCA – Principal Component Analysis

- PCA is a method for reducing the number of features in data.
- It keeps the most important information (variance) while removing noise.
- Helps make data simpler, faster to process, and easier to visualize.

How PCA Works

- Standardize the data (so all features are equal).
- Find the directions (called principal components) where data varies most.
- Keep only the top components that carry the most information.

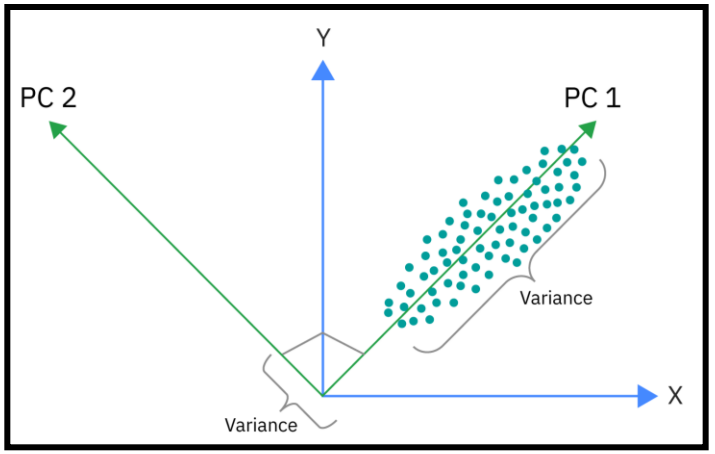


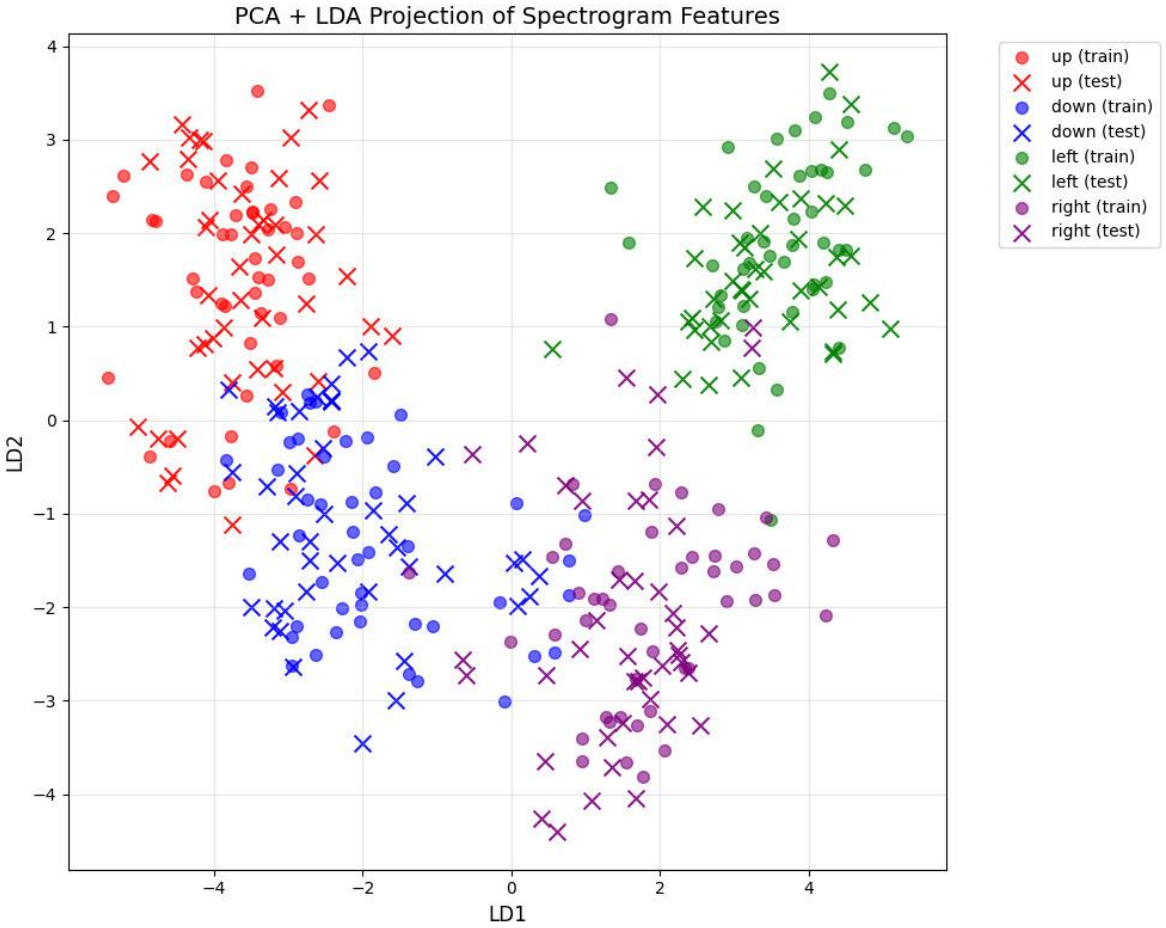
Fig: PCA variance

LDA + PCA Stage

Standardize features → apply PCA

- Reduce **36-D → 10-D** while retaining maximum variance
- Input: PCA-compressed 10-D data
- Output: 2-D subspace maximizing class separation
- LDA projection emphasizes differences between the 4 commands while minimizing within-class spread

PCA + LDA Projection of Spectrogram Features



This shows how well LDA + PCA separates commands

Classifiers & Training

- **Input Features**: LDA-transformed spectrogram statistics (2D representation).
- **KNN**: varied k, uniform/distance weights.
- **SVM (Linear)**: varied C.
- **SVM (RBF)**: varied C and gamma.
- **Parameter tuning**: GridSearchCV (5-fold CV, accuracy metric).
- Hyperparameters were chosen through a **mix of default values and small trials** to find a good balance between **accuracy and training efficiency**.

Dataset & Setup

- Audio Format: 16 kHz sampling rate
- A sampling rate of 16 kHz was selected because it provides enough frequency detail for speech and command recognition while significantly reducing computational requirements.
- It is the lowest rate (found by trial and error) that preserves essential information for the model's purpose, making it ideal for real-time and live system performance.
- Commands: 4 classes (up=0, down=1, left=2, right=3)
- Train/Test Split: 50% / 50%
- Train Dataset: 180 Test Dataset: 180

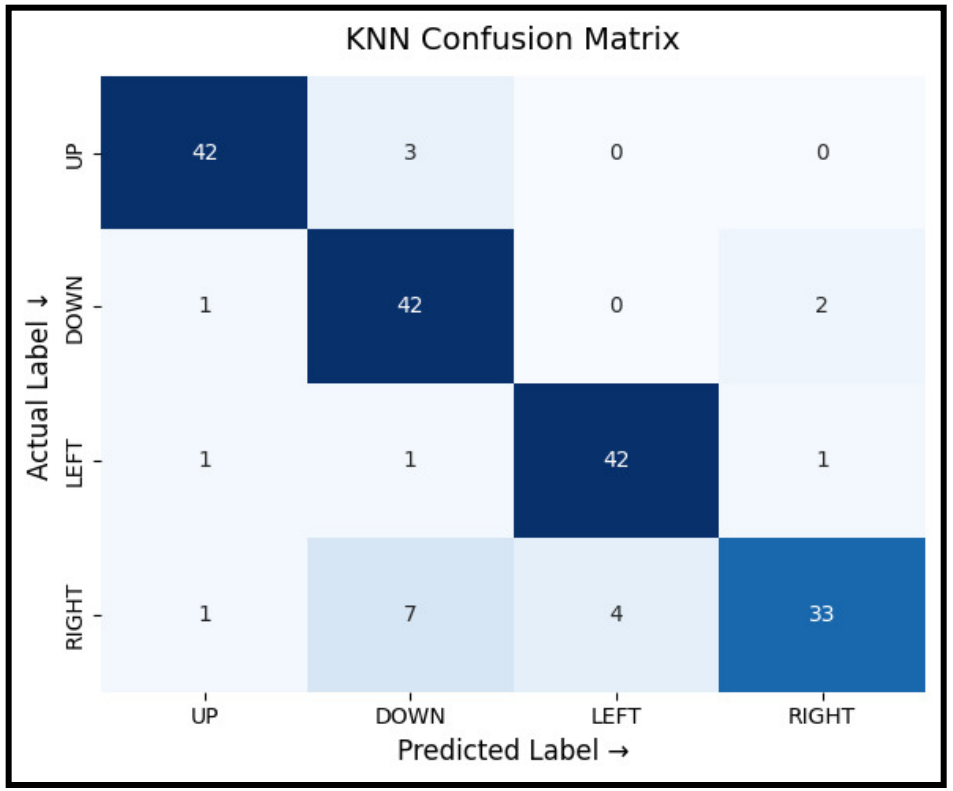
Performance Metrics: K-Nearest Neighbors

Input: PCA-compressed 10-D data

Class	Precision	Recall	F1-Score
Up	0.93	0.93	0.93
Down	0.79	0.93	0.86
Left	0.91	0.93	0.92
Right	0.92	0.73	0.81

Hyperparameters

Parameter	Value
n_neighbors	3
weights	uniform



Accuracy: 88.30%

Decision Boundary for K-Nearest Neighbors

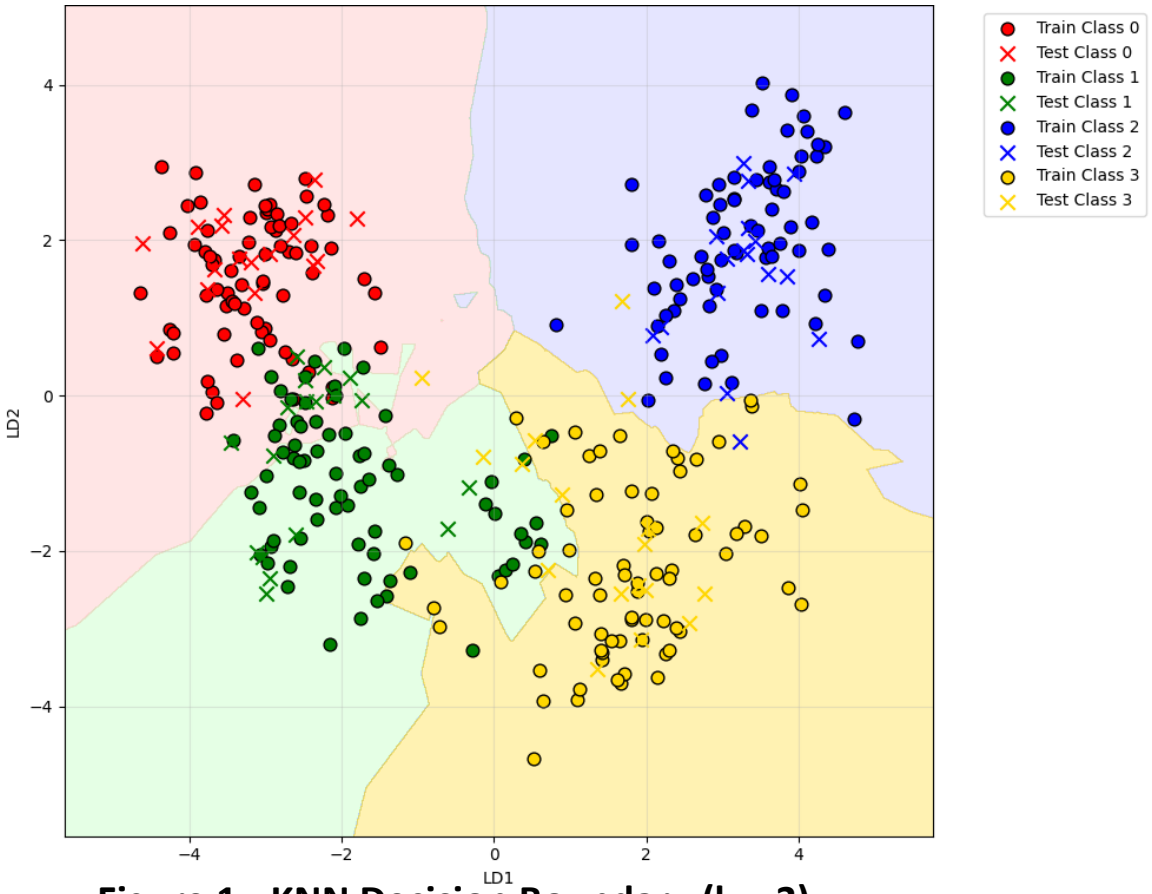


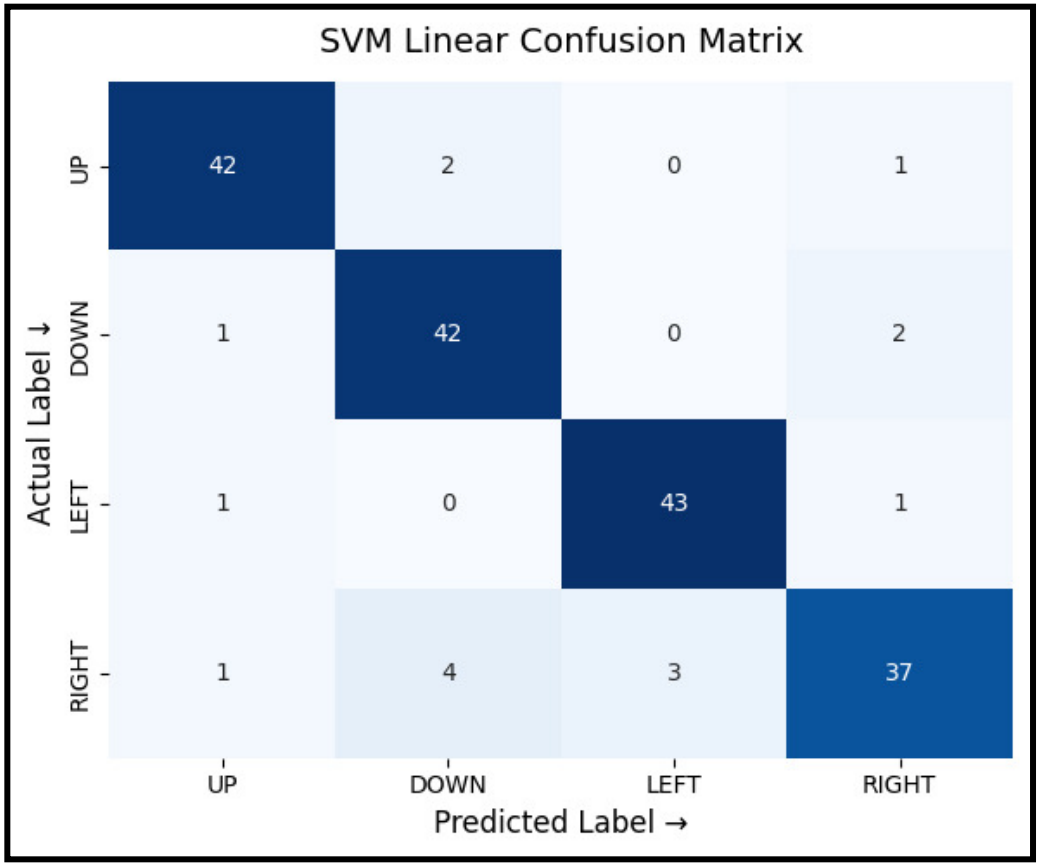
Figure 1 - KNN Decision Boundary (k = 3)

Performance Metrics: SVM (Linear Kernel)

Class	Precision	Recall	F1-Score
Up	0.93	0.93	0.93
Down	0.88	0.93	0.90
Left	0.93	0.96	0.95
Right	0.90	0.82	0.86

Hyperparameters

Parameter	Value
C	0.01



Accuracy: 91.11%

Decision Boundary SVM Linear

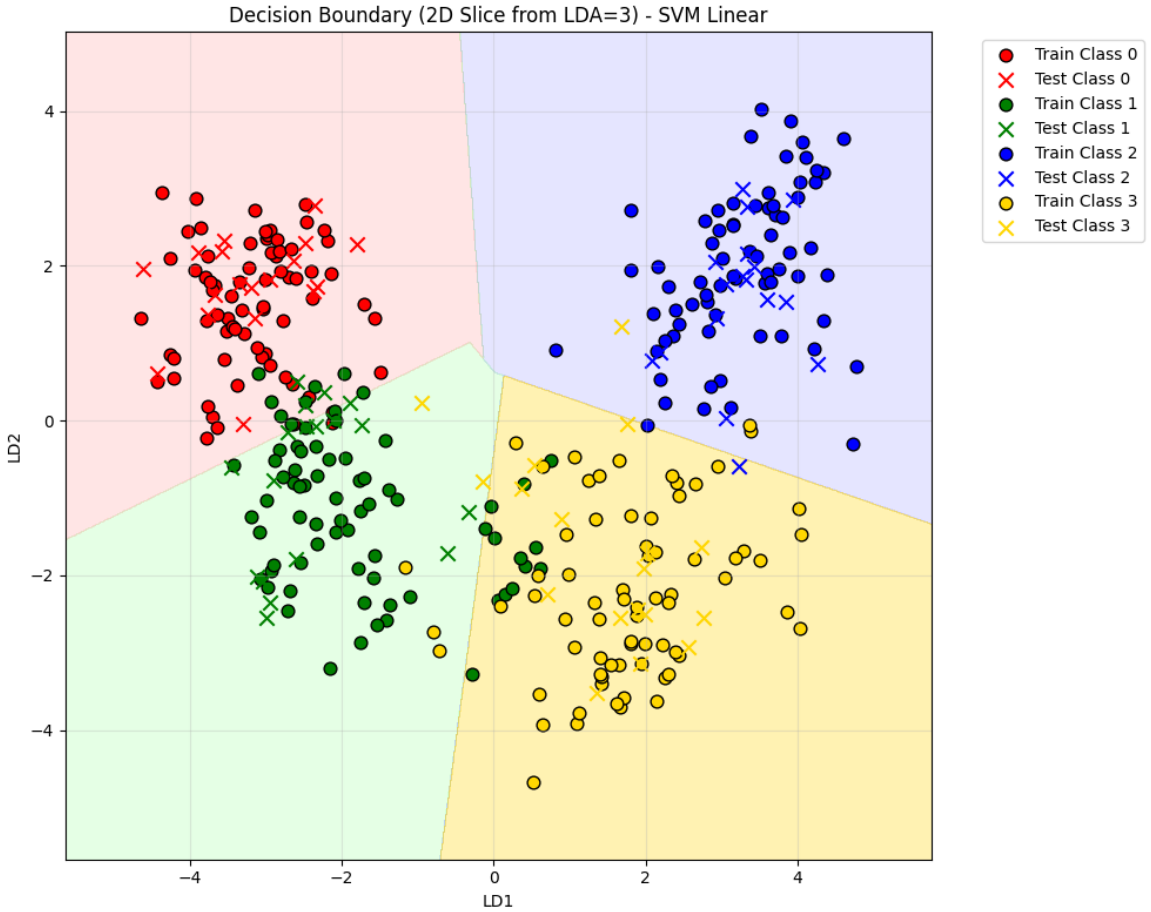


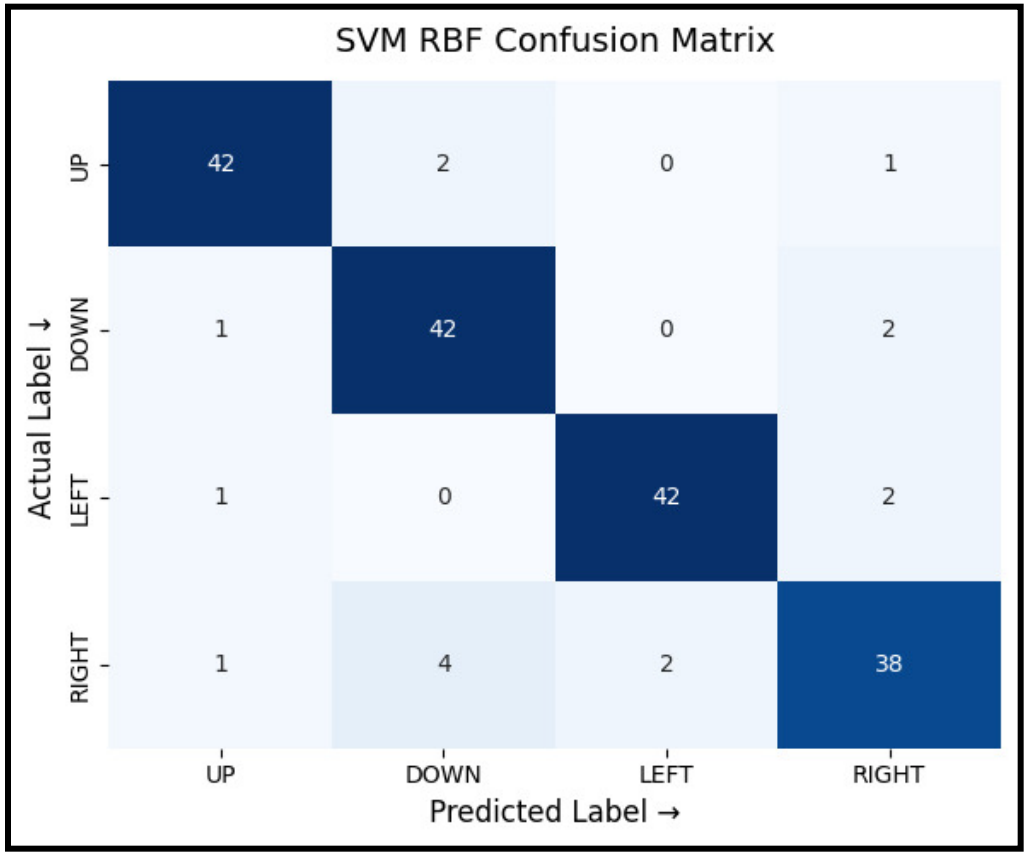
Figure 2 – SVM Linear Decision Boundary (C = 0.01)

Performance Metrics: SVM (RBF)

Class	Precision	Recall	F1-Score
Up	0.91	0.93	0.98
Down	0.87	0.89	0.95
Left	0.93	0.88	0.94
Right	0.90	0.87	0.87

Hyperparameters

Parameter	Value
C	0.1
gamma	scale



Accuracy: 91.11%

Decision Boundary SVM RBF

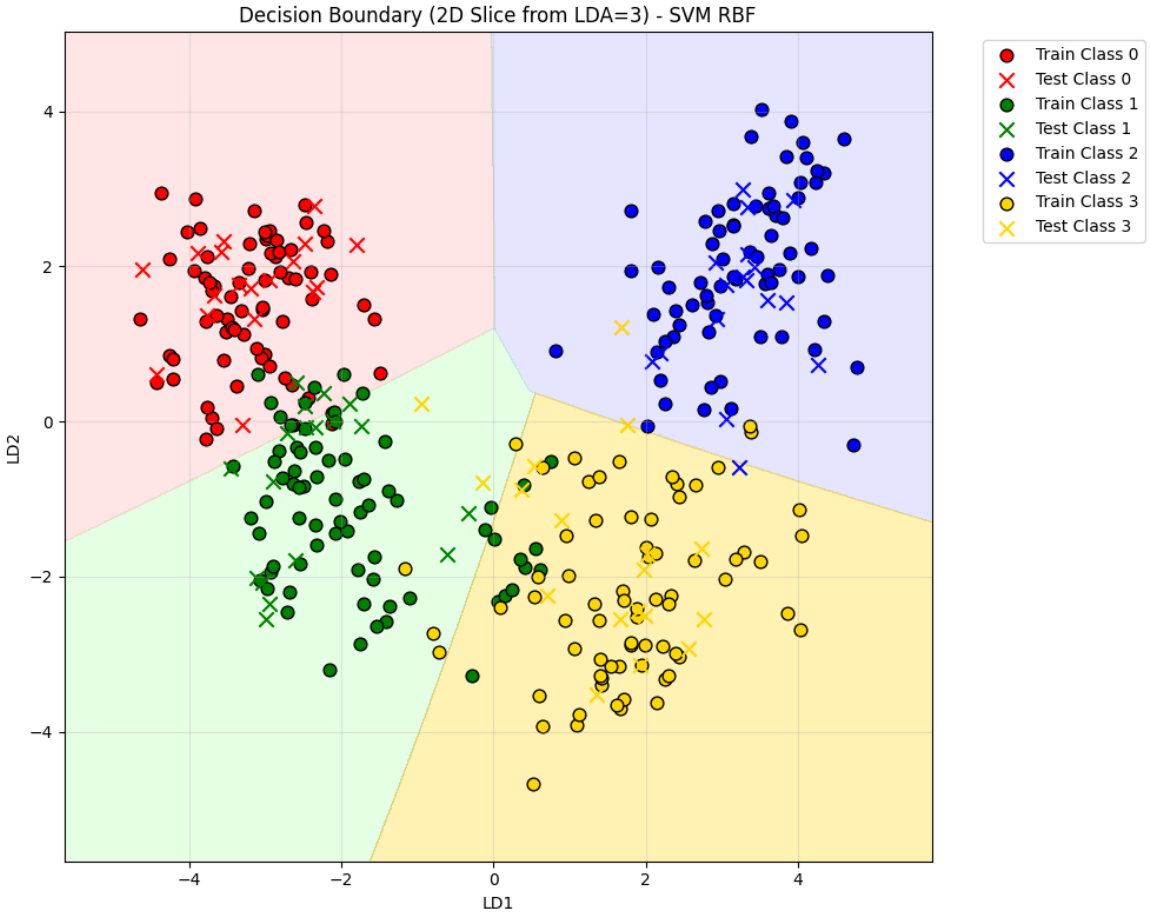


Figure 3 – SVM RBF Boundary (C = 0.01)

Conclusion

Model	Accuracy	Remarks
KNN	88.30%	Slightly affected by PCA compression
SVM Linear	91.11%	0.55% reduction in accuracy after PCA compression
SVM RBF	91.11%	Increased accuracy with Pca+Lda pipeline

- **PCA + LDA** pipeline slightly affects the model performance
- This pipeline turned out to be better for the SVM RBF model
- ROI-based spectrogram statistics remain an effective feature choice.
- Across both 1.b and 1.c pipelines, SVM consistently outperforms KNN in accuracy and boundary robustness.

7. Live Classification

Live Classification Pipeline

- **Record Audio:** Captures 1-second voice samples (16 kHz).
- **Extract Features:** Converts sound to a **spectrogram** and computes key stats. (ROI)
- **Predict Command:** Uses trained **Scaler + LDA + Classifiers** models to classify direction.
- **Move Cursor:** A **Tkinter GUI** updates the cursor position based on prediction.
- **Runs Continuously:** Real-time loop handles recording, prediction, and movement.

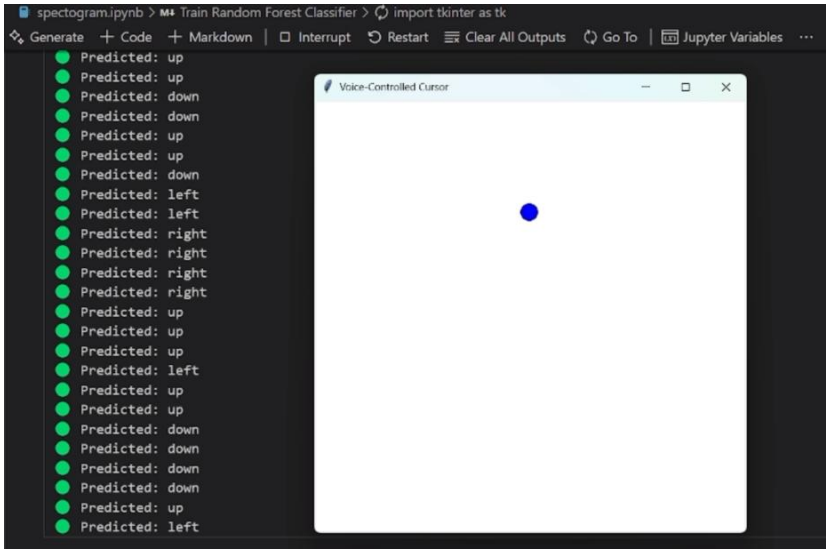


Fig: Live classification GUI

References

1. http://webapps.chem.uoa.gr/efs/applets/AppletNyquist/Appl_Nyquist2.html
2. https://brianmcfee.net/dstbook-site/content/ch02-sampling/Quantization.html
3. https://www.dsprelated.com/showarticle/938.php
4. https://docs.scipy.org/doc/scipy/tutorial/fft.html
5. https://sebastianraschka.com/Articles/2014_python_lda.html#summarizing-the-lda-approach-in-5-steps
6. https://scikit-learn.org/stable/modules/neighbors.html
7. https://scikit-learn.org/stable/modules/svm.html
8. https://scikit-learn.org/stable/modules/tree.html
9. Jolliffe, I. T., & Cadima, J. (2016). *Principal Component Analysis: A Review and Recent Developments. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences,* 374(2065), 20150202.
10. Shlens, J. (2014). *A Tutorial on Principal Component Analysis. arXiv preprint arXiv:1404.1100.*
11. K-Nearest Neighbors (KNN):Halder, R. K. (2024). Enhancing K-nearest neighbor algorithm. Journal of Big Data, 11(1), 1–20.
12. Ladicky, Lubor & Torr, Philip. (2011). Linear Support Vector Machines. 985-992.
13. Apostolidis-Afentoulis, Vasileios. (2015). SVM Classification with Linear and RBF kernels. 10.13140/RG.2.1.3351.4083.

Thank You for your attention!