A black text on a white background

Description automatically generated

Bachelor of Engineering Honours in Robotics and Artificial Intelligence

**6FTC2057 Visual and Spoken Interfaces**

Report Title:

Spoken Interfaces Report

|  |  |
| --- | --- |
| Name | Student ID |
| Jawad Bin Joha | UH Student ID: 23029411  PSB Student ID: 530BIYKP |

Contents

[Abstract 1](#_Toc181739593)

[Introduction 2](#_Toc181739594)

[Methodology 3](#_Toc181739595)

[Results 4](#_Toc181739596)

[GMM Model Figures 4](#_Toc181739597)

[KNN Model Figures 6](#_Toc181739598)

[SVM Model 8](#_Toc181739599)

[Evaluation 10](#_Toc181739600)

[Conclusion 10](#_Toc181739601)

[References 11](#_Toc181739602)

[Appendix 12](#_Toc181739603)

# Abstract

The following report focuses on the development and evaluation of a speaker identification system that can identify speakers from audio recordings. From the TIMIT voice dataset, five speakers were selected and fifty audio files were processed. The system accomplished feature extraction and capturing key audio characteristics using Mel-Frequency Cepstral Coefficients (MFCCs).

A Gaussian Mixture Model (GMM), K-Nearest Neighbours Model (KNN), and Support Vector Machine Model (SVM) were implemented to classify speakers with reference to extracted features. Each model was trained using the audio files from the five speakers. The classification accuracy was calculated using file and frame level confusion matrices.

The GMM and KNN Models achieved 100% file level accuracy, while the SVM model showed a file accuracy of 60%. The frame-level accuracy was varied among the 3 models, with KNN and GMM showing robust classification accuracy across most classes, whereas SVM exhibited more misclassification.

Features 14, 9 and 12 were identified as the most significant across all the models which emphasized the relevance of MFCC features in speaker identification. The results indicate that in terms of effectiveness, the GMM model is ranked the highest, followed by the KNN model with the SVM model coming in last.

# Introduction

Speaker Identification is the process of identifying the speaker from a given utterance by comparing the voice biometrics of the utterance with those utterance models stored beforehand [1]. These systems have been widely applied in security, forensics, and human-computer interfaces to both maintain security protocol and enhance personalized experiences.

The development of a robust speaker identification system hinges on the recognition of specific voice patterns unique to each speaker, such as accent, pitch, and intonation along with the pre processing and feature extraction methods that can assess these voice patterns reliably. The TIMIT database, with its diverse speech samples, provided an ideal dataset from which five speakers could be selected to create a controlled but representative sample voices to gauge the strengths and shortcomings of different classifiers.

The Mel-Frequency Cepstral Coefficients (MFCCs) have emerged as one of the most effective feature representations for speaker identification due to their alignment with the human auditory perception of sound [2] and have been utilized widely in speech recognition and identification applications as they mimic human hearing in the way they capture the spectral properties of audio signals which this project intends to leverage so that raw audio can be transformed into useful feature vectors for each model to use.

The three models, Gaussian Mixture Model (GMM), K-Nearest Neighbours Model (KNN), and Support Vector Machine Model (SVM), had their performances evaluated. The purpose of implementing multiple classifiers was to compare their effectiveness and understand the strengths and limitations of each model for speaker identification.

# Methodology

The development of the speaker identification system comprised three main stages: pre-processing, feature extraction, and classification [3]. In the pre-processing stage, the audio files were standardized by down sampling all files to a 16 kHz sampling rate. This standardization ensured consistency across the dataset, as 16 kHz is considered a commonly used sampling rate in speech processing with reference to the Nyquist-Shannon sampling theorem [4].

MATLAB’s filtering techniques were used to minimize background noise to produce cleaner input data for feature extraction. The audio files were then normalized in amplitude to maintain consistent volume levels across all recordings, which further helped in reducing variability that could negatively impact classification.

Mel-Frequency Cepstral Coefficients (MFCCs) were chosen for feature extraction due to their effectiveness in capturing the timbral qualities of speech [5]. Each audio file was divided into overlapping frames of 25-millisecond, with a 10-millisecond overlap, to ensure that the temporal details of the speech signal were captured adequately.

To avoid spectral leakage, which can distort the frequency representation of the signal, a hamming window was applied to each frame. Fourteen MFCCs were computed per frame, capturing key characteristics of the speech signal’s spectral properties.

The GMM Model was selected for its ability to model complex distributions of MFCC features through a mixture of Gaussians representing distinct speaker characteristics. The GMM was trained using the Expectation-Maximization (EM) algorithm [6], which optimizes the model parameters by maximizing the likelihood of observed features.

The KNN Model was also chosen for its simplicity and effectiveness, particularly for tasks involving distance-based feature similarity. The Euclidean distance metric was used to compare feature vectors, and a K-value of 5 was selected to balance bias and variance in classification.

Finally, the SVM Model with a radial basis function (RBF) kernel was implemented to handle complex, non-linear relationships in the feature space. The regularization parameter was optimized through cross-validation to prevent overfitting.

Confusion matrices were used to evaluate each model by providing both the per-class and overall classification accuracy. Each model’s performance was assessed based on file-level and frame-level classification accuracy, where file-level accuracy measured correct classification per audio file and frame-level accuracy assessed accuracy on a per-frame basis.

# Results

The following sections present the results for each classification model, including the confusion matrices and feature importance rankings.

|  |  |
| --- | --- |
| GMM Model Figures | |
|  | The GMM model achieved 100% classification accuracy for all test audio files, demonstrating high file accuracy. This result indicates that the GMM model generalizes well when classifying entire audio files, making it highly reliable for this dataset. |
|  | The frame-level confusion matrix for the GMM model demonstrates that class 3 had the highest accuracy at 90.5%, while classes 1 and 5 had lower accuracies, with more instances of misclassification. This suggests that the model struggled with these classes on a per-frame basis, possibly due to similarities in speaker characteristics or noise in the audio signal. |
|  | The feature importance graph for the GMM model identified features 14, 9, and 12 as the most significant. These features are likely to contain the most distinguishing information for each speaker, while features with lower importance scores contributed minimally to classification decisions. |
|  | The histogram of predicted labels for the GMM model shows a higher frequency of predictions for classes 3 and 5, indicating a preference for these classes. This could be due to a greater number of samples or more distinctive patterns in these classes. |
|  | The validation accuracy confusion chart shows that the GMM model performed well on the validation set, with minimal misclassification. The lowest validation accuracy was 87.3% for class 5, while other classes had higher accuracies, indicating strong generalization. |

|  |  |
| --- | --- |
| KNN Model Figures | |
|  | The KNN model also achieved perfect file-level classification accuracy, with 100% accuracy across all classes. This indicates that KNN is effective at recognizing the overall pattern of each speaker’s voice in complete audio files. |
|  | At the frame level, KNN’s performance varied across classes. Class 3 showed the highest frame-level accuracy at 84.1%, while classes 1 and 5 had lower accuracies, indicating potential challenges with these classes when classifying individual frames. |
|  | Similar to the GMM model, the KNN feature importance plot shows features 14, 9, and 12 as the most significant. This consistency across models reinforces the effectiveness of these features in distinguishing between speakers. |
|  | The histogram of predicted labels indicates a strong prediction bias towards class 5, which may suggest overfitting or an imbalance in the data. |
|  | The validation accuracy for KNN was high, with most classes achieving over 98% accuracy. This indicates that the KNN model generalizes well to unseen data. |

|  |  |
| --- | --- |
| SVM Model Figures | |
|  | The SVM model, in contrast to GMM and KNN, showed limited file-level accuracy, with only 60% accuracy for the first file-level class. This suggests that SVM struggles with classifying audio files compared to the other models. |
|  | At the frame level, SVM showed moderate performance across classes, with class 3 achieving the highest frame-level accuracy at 95.1%. Classes 1 and 5, however, had lower accuracies, indicating more misclassification. |
|  | Feature importance analysis revealed that features 14, 9, and 12 were again the most important, aligning with the results from the GMM and KNN models. |
|  | The histogram of predicted labels for SVM shows a more balanced distribution compared to KNN, with noticeable peaks for classes 3 and 5. |
|  | The validation accuracy was generally high, although class 5 had the lowest accuracy at 70.4%, suggesting some room for improvement in SVM’s generalization ability. |

# Evaluation

The comparison of the three models shows that the GMM and KNN models perform better than SVM in terms of file-level and frame-level accuracy. The high performance of GMM can be traced to its ability to model complex distributions of MFCC features, which allows it to effectively capture variations in speaker characteristics.

KNN, known for its simplicity in classification tasks, also showed strong performance, effectively using proximity in feature space to classify speakers accurately. Both models demonstrated the ability to generalize well to unseen data, particularly at the file level, which is extremely important for real-world applications of speaker identification.

The SVM model, although generally effective for many classification tasks, faced limitations in this context. Its lower performance may be attributed to its sensitivity to imbalances in the dataset and difficulty handling overlapping feature distributions among different speakers. As shown by the lower file-level accuracy, SVM may require further touch ups, such as adjusting the regularization parameter or incorporating additional features, to improve its performance in speaker identification.

Additionally, across all models, features 14, 9, and 12 consistently ranked as the most significant, highlighting the importance of specific MFCC features in capturing unique speaker traits.

# Conclusion

The results of this study show that the GMM model is the most effective classifier for speaker identification for the chosen dataset, achieving perfect file-level accuracy and consistently high frame-level accuracy across most classes. GMM’s ability to model complex distributions of MFCC features allows it to capture the unique attributes of each speaker reliably, leading to effective generalization to new data. The high accuracy of GMM suggests its reliability for real-world applications where precise speaker identification is crucial.

The KNN model achieved 100% file-level accuracy and strong frame-level accuracy for most classes as well. This effectiveness in identifying patterns based on feature similarity points to the MFCC features that provide a suitable representation for distinguishing between speakers. However, KNN displayed a slight prediction bias toward certain classes, which could affect its output on more diverse or imbalanced datasets. This bias suggests that KNN may benefit from additional data balancing techniques or further tuning to enhance its robustness.

The SVM model showed some limitations in this area, particularly at the file level, where it struggled to achieve the same accuracy as GMM and KNN. The model’s challenges likely stem from overlapping feature distributions among speakers and the class imbalance in the dataset, but despite these limitations, SVM demonstrated reasonable frame-level accuracy for certain classes, indicating its potential for specific scenarios. Future improvements for SVM could include feature engineering beyond MFCCs and data balancing methods to improve generalization. Overall, GMM and KNN proved to be the most effective models for speaker identification, with potential improvements exploring model combinations or ensemble approaches to further enhance accuracy.

# References

[1] ScienceDirect, 2023. Speaker Identification. Available at: https://www.sciencedirect.com/topics/computer-science/speaker-identification#:~:text=Speaker%20Identification%20(SI)%20is%20the,those%20utterance%20models%20stored%20beforehand [Accessed 5 November 2024].

[2] Davis, S. & Mermelstein, P., 1980. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. IEEE Transactions on Acoustics, Speech, and Signal Processing, 28(4), pp. 357-366. Available at: https://ieeexplore.ieee.org/document/1163420 [Accessed 5 November 2024].

[3] Al-Dulaimi, H.W., Aldhahab, A. & Al Abboodi, H.M., 2023. Recent Advances in Speaker Recognition Using Machine Learning Techniques. International Association of Scientists in the Field of Engineering. Available at: https://inass.org/wp-content/uploads/2023/05/2023103130-2.pdf [Accessed 5 November 2024].

[4] Picovoice, 2024. Audio Sampling and Sample Rate. Available at: https://picovoice.ai/blog/audio-sampling-and-sample-rate/#:~:text=The%20Nyquist%2DShannon%20sampling%20theorem,using%20human%20speech%20and%20voice [Accessed 5 November 2024].

[5] ScienceDirect, 2023. Cepstral Coefficient. Available at: https://www.sciencedirect.com/topics/computer-science/cepstral-coefficient [Accessed 5 November 2024].

[6] Bi, C., 2019. The EM Algorithm Explained. Medium. Available at: https://medium.com/@chloebee/the-em-algorithm-explained-52182dbb19d9 [Accessed 5 November 2024].

# Appendix

|  |
| --- |
| GMM Source Code |
| % GMM\_Classifier\_Model  clear all; close all; clc;  % Define the data folder  dataFolder = "C:\Education matters\PSB related\PSB Subjects\PSB Term 4\6FTC2057 Visual and Spoken Interfaces\Jawad\_530BIYKP\_Spoken\_Interfaces\timit\";  % Define the specific folders you want to include  specificFolders = { ...  'dr1-mcpm0', ... % Folder for Speaker 1  'dr4-maeb0', ... % Folder for Speaker 2  'dr5-mbgt0', ... % Folder for Speaker 3  'dr7-madd0', ... % Folder for Speaker 4  'dr8-mbcg0' ... % Folder for Speaker 5  };  % Create an empty cell array to hold the file paths  fileList = {};  % Loop through each specific folder and get the .wav files  for i = 1:length(specificFolders)  folderPath = fullfile(dataFolder, specificFolders{i});  wavFiles = dir(fullfile(folderPath, '\*.wav')); % Get all .wav files in the folder  fileList = [fileList; fullfile(folderPath, {wavFiles.name})']; % Append to the fileList  end  % Create the audio datastore using the custom file list  ads = audioDatastore(fileList, 'LabelSource', 'foldernames');  % Split the datastore into training and testing sets  [adsTrain, adsTest] = splitEachLabel(ads, 0.6); % split files - 60% train 40% test  % Display the datastore and the number of speakers in the test datastore  trainDatastoreCount = countEachLabel(adsTrain);  testDatastoreCount = countEachLabel(adsTest);  [sampleTrain, dsInfo] = read(adsTrain); % provide a sample  sound(sampleTrain, dsInfo.SampleRate);  reset(adsTrain);  %% Feature Extraction  fs = dsInfo.SampleRate;  windowLength = round(0.03 \* fs);  overlapLength = round(0.025 \* fs);  afe = audioFeatureExtractor(SampleRate=fs, ...  Window=hamming(windowLength, "periodic"), OverlapLength=overlapLength, ...  zerocrossrate=true, shortTimeEnergy=true, pitch=true, mfcc=true);  %%  featureMap = info(afe); % with afe, one can seek mfcc  %%  % Extract features from the training set.  features = [];  labels = [];  energyThreshold = 0.005;  zcrThreshold = 0.2;  allFeatures = extract(afe, adsTrain);  allLabels = adsTrain.Labels;  for ii = 1:numel(allFeatures)  thisFeature = allFeatures{ii};  isSpeech = thisFeature(:, featureMap.shortTimeEnergy) > energyThreshold;  isVoiced = thisFeature(:, featureMap.zerocrossrate) < zcrThreshold;  voicedSpeech = isSpeech & isVoiced;  thisFeature(~voicedSpeech, :) = [];  thisFeature(:, [featureMap.zerocrossrate, featureMap.shortTimeEnergy]) = [];  label = repelem(allLabels(ii), size(thisFeature, 1));  features = [features; thisFeature]; % adding up the features  labels = [labels, label]; % adding up the labels  end  %% Normalize features  M = mean(features, 1);  S = std(features, [], 1);  features = (features - M) ./ S;  %% - GMM  uniqueLabels = unique(labels); % Get the unique labels  numLabels = length(uniqueLabels); % Number of unique classes  gmModel = cell(numLabels, 1); % Initialize a cell array for GMMs  % Fit GMM for each label  for i = 1:numLabels  label = uniqueLabels(i); % Current label  X\_label = features(labels == label', :); % Features for the current label  numComponents = 2; % Number of Gaussian components  gmModel{i} = fitgmdist(X\_label, numComponents); % Fit GMM  end  %% Extract features from the test set  % Extract features from the data set.  features = [];  labels = [];  energyThreshold = 0.005;  zcrThreshold = 0.2;  allFeatures = extract(afe,adsTrain);  allLabels = adsTrain.Labels;  for ii = 1:numel(allFeatures)  thisFeature = allFeatures{ii};  isSpeech = thisFeature(:,featureMap.shortTimeEnergy) > energyThreshold;  isVoiced = thisFeature(:,featureMap.zerocrossrate) < zcrThreshold;  voicedSpeech = isSpeech & isVoiced;  thisFeature(~voicedSpeech,:) = [];  thisFeature(:,[featureMap.zerocrossrate,featureMap.shortTimeEnergy]) = [];  label = repelem(allLabels(ii),size(thisFeature,1));    features = [features;thisFeature]; % adding up the features  labels = [labels,label]; % adding up the labels  end  %%  % Pitch and MFCC are not on the same scale. This will bias the classifier. Normalize  % the features by subtracting the mean and dividing the standard deviation.  M = mean(features,1);  S = std(features,[],1);  features = (features-M)./S;  %%  [idx,scores] = fscmrmr(features,labels);  % Create a bar graph  figure;  bar(scores(idx));  xlabel('Features');  ylabel('Importance Score');  title('Feature Importance (GMM)');  % Optional: Add feature names on the x-axis if they exist  xticks(1:length(idx));  xticklabels(idx); % replace featureNames with your actual feature names  xtickangle(45); % Angle the x-axis labels if they are overlapping  %% - GMM  % X is your data matrix of size [n, d], where n is the number of data points and d is the number of features  % Y is your label vector of size [n, 1], where each element corresponds to the class label of the feature vector in X  % Get the unique labels (classes) in your data  uniqueLabels = unique(labels); % e.g., if Y has 16 unique labels, this will contain those labels  numLabels = length(uniqueLabels); % Number of unique classes (e.g., 16)  % Initialize a cell array to store GMMs for each label  gmModel = cell(numLabels, 1);  % Loop over each unique label (class)  for i = 1:numLabels  label = uniqueLabels(i); % Get the current label (class)  % Extract the feature vectors corresponding to the current label  X\_label = features(labels == label', :); % Data points that belong to the current label  % Choose the number of Gaussian components for the GMM (e.g., 2 for each class)  numComponents = 2;  % Fit the GMM to the data points for this label  gmModel{i} = fitgmdist(X\_label, numComponents);  end  %%  features = [];  labels = [];  numVectorsPerFile = [];  allFeatures = extract(afe,adsTest);  allLabels = adsTest.Labels;  for ii = 1:numel(allFeatures)  thisFeature = allFeatures{ii};  isSpeech = thisFeature(:,featureMap.shortTimeEnergy) > energyThreshold;  isVoiced = thisFeature(:,featureMap.zerocrossrate) < zcrThreshold;  voicedSpeech = isSpeech & isVoiced;  thisFeature(~voicedSpeech,:) = [];  numVec = size(thisFeature,1);  thisFeature(:,[featureMap.zerocrossrate,featureMap.shortTimeEnergy]) = [];  label = repelem(allLabels(ii),numVec);  numVectorsPerFile = [numVectorsPerFile,numVec];  features = [features;thisFeature];  labels = [labels,label];  end  features = (features-M)./S;  %%  % Initialize an array to store predicted labels  Y\_pred = zeros(size(features, 1), 1); % Preallocate array for predicted labels  % Loop over the test data points  for j = 1:size(features, 1)  x\_test = features(j, :); % Extract one test data point (feature vector)  % Initialize an array to store likelihoods for each label  likelihood = zeros(numLabels, 1);  % Compute likelihood under each GMM  for i = 1:numLabels  likelihood(i) = pdf(gmModel{i}, x\_test); % Compute likelihood under GMM for label i  end  % Assign the label with the highest likelihood  [~, predictedLabelIdx] = max(likelihood);  Y\_pred(j) = uniqueLabels(predictedLabelIdx); % Predicted label for the current test data point  end  %%  % Calculate the confusion matrix  % Y\_test = uint8(labels);  % Y\_pred = uint8(Y\_pred');  %  % confusionMatrix = confusionmat(Y\_test, Y\_pred);  % confusionchart(Y\_test, Y\_pred); % Visualize the confusion matrix  % % Calculate accuracy  % accuracy = sum(Y\_test == Y\_pred) / length(Y\_test);  % disp(['Accuracy: ', num2str(accuracy \* 100), '%']);  %% Per File Accuracy  % Calculate the per-file prediction by taking the mode of predictions for each file.  file\_start = 1;  Y\_test\_file = [];  Y\_pred\_file = [];  for i = 1:length(numVectorsPerFile)  % Get the frame indices for the current file  numFrames = numVectorsPerFile(i);  file\_frames = Y\_pred(file\_start:file\_start + numFrames - 1);  true\_label = labels(file\_start); % True label for the entire file    % Store the mode of the predictions as the file-level prediction  Y\_pred\_file = [Y\_pred\_file; mode(file\_frames)];  Y\_test\_file = [Y\_test\_file; true\_label];    % Update the starting frame index for the next file  file\_start = file\_start + numFrames;  end  %% - Manual Testing with specific wav file  testfeatures = [];  predictedLabel = [];  % Define only 5 speakers  labelArray = [1, 2, 3, 4, 5]; % Adjust this if necessary for your specific speakers  stringArray = [...  "dr1-mcpm0/sa1", ... % Speaker 1  "dr4-maeb0/sa1", ... % Speaker 2  "dr5-mbgt0/sa1", ... % Speaker 3  "dr7-madd0/sa1", ... % Speaker 4  "dr8-mbcg0/sa1" ... % Speaker 5  ];  % Initialize arrays to store results for all speakers  Y\_true = [];  Y\_predicted = [];  % Loop through all speakers for playback and prediction  for select\_Wav = 1:length(stringArray)  str1 = stringArray(select\_Wav);  str2 = '.wav';  str123 = strcat(dataFolder, str1, str2);  % Check if the file exists before attempting to read  if exist(str123, 'file')  [audioIn, fs] = audioread(str123);  sound(audioIn, fs);  pause(length(audioIn) / fs + 1);  % Extract features from the test audio  myfeatures = extract(afe, audioIn);  testfeatures = [];  % Process features for speech detection and normalization  for ii = 1:size(myfeatures, 1)  thisFeature = myfeatures(ii, :);  isSpeech = thisFeature(:, featureMap.shortTimeEnergy) > energyThreshold;  isVoiced = thisFeature(:, featureMap.zerocrossrate) < zcrThreshold;  voicedSpeech = isSpeech & isVoiced;  thisFeature(~voicedSpeech, :) = [];  thisFeature(:, [featureMap.zerocrossrate, featureMap.shortTimeEnergy]) = [];  testfeatures = [testfeatures; thisFeature];  end  % Normalize the features  testfeatures = (testfeatures - M) ./ S;  % Predict label for each feature frame  predictedLabel = zeros(size(testfeatures, 1), 1);  for j = 1:size(testfeatures, 1)  likelihood = zeros(numLabels, 1);  for i = 1:numLabels  likelihood(i) = pdf(gmModel{i}, testfeatures(j, :));  end  [~, predictedLabelIdx] = max(likelihood);  predictedLabel(j) = uniqueLabels(predictedLabelIdx);  end  % Append true and predicted labels for this speaker  Y\_true = [Y\_true; uint8(labelArray(select\_Wav) \* ones(size(predictedLabel)))];  Y\_predicted = [Y\_predicted; uint8(predictedLabel)];  else  error('Audio file does not exist: %s', str123);  end  end  myfeatures = extract(afe, audioIn);  % Start processing features  for ii = 1:size(myfeatures, 1)  thisFeature = myfeatures(ii, :);  isSpeech = thisFeature(:, featureMap.shortTimeEnergy) > energyThreshold;  isVoiced = thisFeature(:, featureMap.zerocrossrate) < zcrThreshold;  voicedSpeech = isSpeech & isVoiced;  thisFeature(~voicedSpeech, :) = []; % adding empty  thisFeature(:, [featureMap.zerocrossrate, featureMap.shortTimeEnergy]) = [];  testfeatures = [testfeatures; thisFeature]; % injecting 3  end  testfeatures = (testfeatures - M) ./ S; % Normalize features  % Now gmModel contains a GMM for each label/class  % You can use these models for classification of new data points  % Initialize an array to store the likelihoods for each label  likelihood = zeros(numLabels, 1);  % Loop over each test feature  for j = 1:size(testfeatures, 1)  for i = 1:numLabels  likelihood(i) = pdf(gmModel{i}, testfeatures(j, :)); % Compute the likelihood under the GMM for label i  end  % Assign the label with the highest likelihood  [~, predictedLabelIdx] = max(likelihood);  predictedLabel(j) = uniqueLabels(predictedLabelIdx); % Predicted label for the new feature vector  end  % Display the predicted label for the selected wav  fprintf("By Mode, for Selected Wav Index, %d, the predicted label is %d\n", select\_Wav, mode(predictedLabel));  % Create a histogram of the predicted labels  figure;  histogram(predictedLabel);  xlabel('Predicted Labels');  ylabel('Frequency');  title('Histogram of Predicted Labels (GMM)');  % Y\_true = uint8(labelArray(select\_Wav) \* ones(1, length(testfeatures)));  % Y\_predicted = uint8(predictedLabel);  figure;  confusionchart(Y\_true, Y\_predicted, title="Validation Accuracy (GMM)", ...  ColumnSummary="column-normalized", RowSummary="row-normalized");  validation\_accuracy = sum(Y\_true == Y\_predicted) / length(Y\_true);  disp(['Validation Accuracy: ', num2str(validation\_accuracy \* 100), '%']);  % Frame-level confusion chart  Y\_test = uint8(labels);  Y\_pred = uint8(Y\_pred');  figure;  confusionchart(Y\_test, Y\_pred, 'Title', 'Per Frame Accuracy (GMM)', ...  'ColumnSummary', 'column-normalized', 'RowSummary', 'row-normalized');  frame\_accuracy = sum(Y\_test == Y\_pred) / length(Y\_test);  disp(['Frame-Level Accuracy: ', num2str(frame\_accuracy \* 100), '%']);    Y\_test\_file = uint8(Y\_test\_file);  Y\_pred\_file = uint8(Y\_pred\_file);  % File-level confusion chart  figure;  confusionchart(Y\_test\_file, Y\_pred\_file, 'Title', 'Per File Accuracy (GMM)', ...  'ColumnSummary', 'column-normalized', 'RowSummary', 'row-normalized');  file\_accuracy = sum(Y\_test\_file == Y\_pred\_file) / length(Y\_test\_file);  disp(['File-Level Accuracy: ', num2str(file\_accuracy \* 100), '%'])  return; |
| KNN Source Code |
| %KNN\_Classifier\_Model  clear all; close all; clc;  % Define the data folder  dataFolder = "C:\Education matters\PSB related\PSB Subjects\PSB Term 4\6FTC2057 Visual and Spoken Interfaces\Jawad\_530BIYKP\_Spoken\_Interfaces\timit\";  % Define the specific folders you want to include  specificFolders = { ...  'dr1-mcpm0', ... % Folder for Speaker 1  'dr4-maeb0', ... % Folder for Speaker 2  'dr5-mbgt0', ... % Folder for Speaker 3  'dr7-madd0', ... % Folder for Speaker 4  'dr8-mbcg0' ... % Folder for Speaker 5  };  % Empty cell array to hold the file paths  fileList = {};  % Loop through each specific folder and get the .wav files  for i = 1:length(specificFolders)  folderPath = fullfile(dataFolder, specificFolders{i});  wavFiles = dir(fullfile(folderPath, '\*.wav')); % Get all .wav files in the folder  fileList = [fileList; fullfile(folderPath, {wavFiles.name})']; % Append to the fileList  end  % Audio datastore created using the custom file list  ads = audioDatastore(fileList, 'LabelSource', 'foldernames');  % Split the datastore into training and testing sets  [adsTrain, adsTest] = splitEachLabel(ads, 0.6); % split files - 60% train 40% test  % Display the datastore and the number of speakers in the test datastore  trainDatastoreCount = countEachLabel(adsTrain);  testDatastoreCount = countEachLabel(adsTest);  [sampleTrain, dsInfo] = read(adsTrain); % provide a sample  sound(sampleTrain, dsInfo.SampleRate);  reset(adsTrain);  %% Feature Extraction  fs = dsInfo.SampleRate;  windowLength = round(0.03 \* fs);  overlapLength = round(0.025 \* fs);  afe = audioFeatureExtractor(SampleRate=fs, ...  Window=hamming(windowLength, "periodic"), OverlapLength=overlapLength, ...  zerocrossrate=true, shortTimeEnergy=true, pitch=true, mfcc=true);  featureMap = info(afe)  %%  features = [];  labels = [];  energyThreshold = 0.005;  zcrThreshold = 0.2;  allFeatures = extract(afe,adsTrain);  allLabels = adsTrain.Labels;  for ii = 1:numel(allFeatures)  thisFeature = allFeatures{ii};  isSpeech = thisFeature(:,featureMap.shortTimeEnergy) > energyThreshold;  isVoiced = thisFeature(:,featureMap.zerocrossrate) < zcrThreshold;  voicedSpeech = isSpeech & isVoiced;  thisFeature(~voicedSpeech,:) = [];  thisFeature(:,[featureMap.zerocrossrate,featureMap.shortTimeEnergy]) = [];  label = repelem(allLabels(ii),size(thisFeature,1));    features = [features;thisFeature]; % adding up the features  labels = [labels,label]; % adding up the labels  end  M = mean(features,1);  S = std(features,[],1);  features = (features-M)./S;  %%  [idx,scores] = fscmrmr(features,labels);  % Create a bar graph  figure;  bar(scores(idx));  xlabel('Features');  ylabel('Importance Score');  title('Feature Importance (KNN)');  % Optional: Add feature names on the x-axis if they exist  xticks(1:length(idx));  xticklabels(idx); % replace featureNames with your actual feature names  xtickangle(45); % Angle the x-axis labels if they are overlapping  %% Train Classifier  trainedClassifier = fitcknn(features,labels, ...  Distance="euclidean", ...  NumNeighbors=5, ...  DistanceWeight="squaredinverse", ...  Standardize=false, ...  ClassNames=unique(labels));  k = 5;  group = labels;  c = cvpartition(group,KFold=k); % 5-fold stratified cross validation  partitionedModel = crossval(trainedClassifier,CVPartition=c);  %%  % Compute the validation accuracy.  validationAccuracy = 1 - kfoldLoss(partitionedModel,LossFun="ClassifError");  fprintf('\nValidation accuracy = %.2f%%\n', validationAccuracy\*100);  %%  % Visualize the confusion chart.  validationPredictions = kfoldPredict(partitionedModel);  figure(Units="normalized", Position=[0.4 0.4 0.4 0.4])  confusionchart(categorical(labels), categorical(validationPredictions), ...  'Title', "Validation Accuracy (KNN)", ...  'ColumnSummary', "column-normalized", ...  'RowSummary', "row-normalized");    % Predict the label (speaker) for each frame by calling |predict| on |trainedClassifier|.  prediction = predict(trainedClassifier, features);  prediction = categorical(string(prediction));    %%  features = [];  labels = [];  numVectorsPerFile = [];  allFeatures = extract(afe,adsTest);  allLabels = adsTest.Labels;  for ii = 1:numel(allFeatures)  thisFeature = allFeatures{ii};  isSpeech = thisFeature(:,featureMap.shortTimeEnergy) > energyThreshold;  isVoiced = thisFeature(:,featureMap.zerocrossrate) < zcrThreshold;  voicedSpeech = isSpeech & isVoiced;  thisFeature(~voicedSpeech,:) = [];  numVec = size(thisFeature,1);  thisFeature(:,[featureMap.zerocrossrate,featureMap.shortTimeEnergy]) = [];    label = repelem(allLabels(ii),numVec);    numVectorsPerFile = [numVectorsPerFile,numVec];  features = [features;thisFeature];  labels = [labels,label];  end  features = (features-M)./S;  %%  % Predict the label (speaker) for each frame by calling |predict| on |trainedClassifier|.  prediction = predict(trainedClassifier,features);  prediction = categorical(string(prediction));  % Calculate accuracy per frame  frameAccuracy = sum(prediction == categorical(labels)) / numel(labels);    % Calculate average accuracy across all frames (if you have multiple frames to average)  averageFrameAccuracy = mean(frameAccuracy); % assuming frameAccuracy holds multiple values in the loop  fprintf('Average Frame Accuracy: %.2f%%\n', averageFrameAccuracy \* 100);    % Visualize the confusion chart for test predictions (per frame)  figure(Units="normalized", Position=[0.4 0.4 0.4 0.4])  confusionchart(categorical(labels(:)), prediction, ...  'Title', "Test Accuracy Per Frame (KNN)", ...  'ColumnSummary', "column-normalized", ...  'RowSummary', "row-normalized");        % Generate predictions for each file  r2 = prediction(1:numel(adsTest.Files));  idx = 1;  for ii = 1:numel(adsTest.Files)  r2(ii) = mode(prediction(idx:idx + numVectorsPerFile(ii) - 1));  idx = idx + numVectorsPerFile(ii);  end  % Calculate accuracy per file  fileAccuracy = sum(r2 == categorical(adsTest.Labels)) / numel(adsTest.Files);  fprintf('Test Accuracy (Per File): %.2f%%\n', fileAccuracy \* 100);  % Visualize the confusion chart for test predictions (per file)  figure(Units="normalized", Position=[0.4 0.4 0.4 0.4])  confusionchart(categorical(adsTest.Labels), categorical(r2), ...  'Title', "Test Accuracy Per File (KNN)", ...  'ColumnSummary', "column-normalized", ...  'RowSummary', "row-normalized");    %% - Manual Testing with specific wav file  testfeatures = [];  % Define the speakers used  labelArray = [1, 2, 3, 4, 5]; % Updated to reflect only 5 speakers  stringArray = [ ...  "dr1-mcpm0/sa1", ... % Speaker 1  "dr4-maeb0/sa1", ... % Speaker 2  "dr5-mbgt0/sa1", ... % Speaker 3  "dr7-madd0/sa1", ... % Speaker 4  "dr8-mbcg0/sa1" ... % Speaker 5  ];  % Loop through all speakers  for select\_Wav = 1:length(stringArray)  str1 = stringArray(select\_Wav);  str2 = '.wav';  str123 = strcat(dataFolder, str1, str2);  [audioIn, fs] = audioread([str123]); % Read the audio file  sound(audioIn, fs); % Play the audio  pause(length(audioIn) / fs + 1); % Pause to allow the audio to finish playing  end  str1 = stringArray(select\_Wav);  str2 = '.wav';  % ------------ End of Change ------------------  str123 = strcat(dataFolder, str1, str2);  [audioIn,fs] = audioread([str123]); % this is like the readwav function of voicebox  sound(audioIn,fs);  myfeatures = extract(afe, audioIn);  % start  for ii = 1:size(myfeatures,1)  thisFeature = myfeatures(ii,:);  isSpeech = thisFeature(:,featureMap.shortTimeEnergy) > energyThreshold;  isVoiced = thisFeature(:,featureMap.zerocrossrate) < zcrThreshold;  voicedSpeech = isSpeech & isVoiced;  thisFeature(~voicedSpeech,:) = []; % adding empty  thisFeature(:,[featureMap.zerocrossrate,featureMap.shortTimeEnergy]) = [];  testfeatures = [testfeatures;thisFeature]; % injecting 3  end  testfeatures = (testfeatures-M)./S; % has to be added  % ------------ Change ------------------  disp('Testing with my values:')  predictedLabel = predict(trainedClassifier,testfeatures);  fprintf("By Mode, for Selected Wav Index, %d, the predicted label is %d\n",select\_Wav, mode(uint8(predictedLabel)));  % Create a histogram of the predicted labels  figure;  histogram(predictedLabel);  xlabel('Predicted Labels');  ylabel('Frequency');  title('Histogram of Predicted Labels (KNN)');  return;  %%  % Add the following:  %%  disp("Clear KNN model");  clear trainedClassifier;  disp("Completed and Exit");  return;  %%  % This is specifically for manual prediction  disp("Saving KNN model and other variables");  save('KNN\_model.mat', 'trainedClassifier');  save('myVariables.mat', 'dataFolder', 'afe', 'fs', 'featureMap','energyThreshold', 'zcrThreshold','M','S');  disp("Saving completed");  return;  %%  % This is specifically for manual prediction  disp("Loading KNN model and other variables")  load('KNN\_model.mat'); % This will load the 'gmdist' variable back into the workspace  load('myVariables.mat'); % This will load the 'gmdist' variable back into the workspace  disp("Loading completed");  return;  %% |
| SVM Source Code |
| % SVM\_Classifier\_Model  clear all; close all; clc;  % Define the data folder  dataFolder = "C:\Education matters\PSB related\PSB Subjects\PSB Term 4\6FTC2057 Visual and Spoken Interfaces\Jawad\_530BIYKP\_Spoken\_Interfaces\timit\";  % Define the specific folders you want to include  specificFolders = { ...  'dr1-mcpm0', ... % Folder for Speaker 1  'dr4-maeb0', ... % Folder for Speaker 2  'dr5-mbgt0', ... % Folder for Speaker 3  'dr7-madd0', ... % Folder for Speaker 4  'dr8-mbcg0' ... % Folder for Speaker 5  };  % Create an empty cell array to hold the file paths  fileList = {};  % Loop through each specific folder and get the .wav files  for i = 1:length(specificFolders)  folderPath = fullfile(dataFolder, specificFolders{i});  wavFiles = dir(fullfile(folderPath, '\*.wav')); % Get all .wav files in the folder  fileList = [fileList; fullfile(folderPath, {wavFiles.name})']; % Append to the fileList  end  % Create the audio datastore using the custom file list  ads = audioDatastore(fileList, 'LabelSource', 'foldernames');  % Split the datastore into training and testing sets  [adsTrain, adsTest] = splitEachLabel(ads, 0.6); % split files - 60% train 40% test  % Display the datastore and the number of speakers in the test datastore  trainDatastoreCount = countEachLabel(adsTrain);  testDatastoreCount = countEachLabel(adsTest);  [sampleTrain, dsInfo] = read(adsTrain); % provide a sample  sound(sampleTrain, dsInfo.SampleRate);  reset(adsTrain);  %% Feature Extraction  fs = dsInfo.SampleRate;  windowLength = round(0.03 \* fs);  overlapLength = round(0.025 \* fs);  afe = audioFeatureExtractor(SampleRate=fs, ...  Window=hamming(windowLength, "periodic"), OverlapLength=overlapLength, ...  zerocrossrate=true, shortTimeEnergy=true, pitch=true, mfcc=true);  featureMap = info(afe);  features = [];  labels = [];  energyThreshold = 0.005;  zcrThreshold = 0.2;  allFeatures = extract(afe, adsTrain);  allLabels = adsTrain.Labels;  for ii = 1:numel(allFeatures)  thisFeature = allFeatures{ii};  isSpeech = thisFeature(:, featureMap.shortTimeEnergy) > energyThreshold;  isVoiced = thisFeature(:, featureMap.zerocrossrate) < zcrThreshold;  voicedSpeech = isSpeech & isVoiced;  thisFeature(~voicedSpeech, :) = [];  thisFeature(:, [featureMap.zerocrossrate, featureMap.shortTimeEnergy]) = [];    label = repelem(allLabels(ii), size(thisFeature, 1));  label = label(:);  features = [features; thisFeature];  labels = [labels; label];  end  %% Normalize features  M = mean(features, 1);  S = std(features, [], 1);  features = (features - M) ./ S;  %% Feature Selection  [idx, scores] = fscmrmr(features, labels);  % Create a bar graph of feature importance  figure;  bar(scores(idx));  xlabel('Features');  ylabel('Importance Score');  title('Feature Importance (SVM)');  % Optional: Add feature names on the x-axis if they exist  xticks(1:length(idx));  xticklabels(idx); % replace featureNames with your actual feature names  xtickangle(45);  %% Train SVM Model with RBF Kernel  % Using a non-linear kernel for potentially better separation  template = templateSVM('KernelFunction', 'rbf', 'KernelScale', 'auto');  SVMModel = fitcecoc(features, labels, 'Learners', template);  %% Extract features from the test set  features = [];  labels = [];  allFeatures = extract(afe, adsTest);  allLabels = adsTest.Labels;  for ii = 1:numel(allFeatures)  thisFeature = allFeatures{ii};  isSpeech = thisFeature(:, featureMap.shortTimeEnergy) > energyThreshold;  isVoiced = thisFeature(:, featureMap.zerocrossrate) < zcrThreshold;  voicedSpeech = isSpeech & isVoiced;  thisFeature(~voicedSpeech, :) = [];  thisFeature(:, [featureMap.zerocrossrate, featureMap.shortTimeEnergy]) = [];    label = repelem(allLabels(ii), size(thisFeature, 1));  label = label(:);  features = [features; thisFeature];  labels = [labels; label];  end  % Normalize test features using the mean and std from training set  features = (features - M) ./ S;  %% Make Predictions  [Y\_pred, scores] = predict(SVMModel, features);  %% Confusion Matrices  % Frame-level confusion chart  figure;  confusionchart(labels, Y\_pred, 'Title', 'Frame-Level Accuracy (SVM)', ...  'ColumnSummary', 'column-normalized', 'RowSummary', 'row-normalized');  disp('Calculating frame-level accuracy...');  frame\_accuracy = sum(labels == Y\_pred) / length(labels);  disp(['Frame-Level Accuracy: ', num2str(frame\_accuracy \* 100), '%']);  %% Per File Accuracy  numVectorsPerFile = countEachLabel(adsTest).Count';  file\_start = 1;  Y\_test\_file = [];  Y\_pred\_file = [];  for i = 1:length(numVectorsPerFile)  numFrames = numVectorsPerFile(i);  if file\_start + numFrames - 1 <= length(Y\_pred) % Ensure we don't go out of bounds  file\_frames = Y\_pred(file\_start:file\_start + numFrames - 1);  true\_label = labels(file\_start);    % Use the mode or majority vote across frames for each file  if ~isempty(file\_frames) % Check if there are any predictions  Y\_pred\_file = [Y\_pred\_file; mode(file\_frames)];  else  Y\_pred\_file = [Y\_pred\_file; NaN]; % Or some default label  end    Y\_test\_file = [Y\_test\_file; true\_label];  end  file\_start = file\_start + numFrames;  end  % File-level confusion chart  figure;  confusionchart(Y\_test\_file, Y\_pred\_file, 'Title', 'File-Level Accuracy (SVM)', ...  'ColumnSummary', 'column-normalized', 'RowSummary', 'row-normalized');  file\_accuracy = sum(Y\_test\_file == Y\_pred\_file) / length(Y\_test\_file);  disp(['File-Level Accuracy: ', num2str(file\_accuracy \* 100), '%']);  %% Histogram of Predicted Labels  figure;  histogram(Y\_pred);  xlabel('Predicted Labels');  ylabel('Frequency');  title('Histogram of Predicted Labels (SVM)');  %% Validation Accuracy  Y\_train\_pred = predict(SVMModel, features);  figure;  confusionchart(labels, Y\_train\_pred, 'Title', 'Validation Accuracy (SVM)', ...  'ColumnSummary', 'column-normalized', 'RowSummary', 'row-normalized');  validation\_accuracy = sum(labels == Y\_train\_pred) / length(labels);  disp(['Validation Accuracy: ', num2str(validation\_accuracy \* 100), '%']); |