[IAS Project] Ancri Carlo

Program behaviour

The program extracts features from audio files belonging to three different "classes" (crow, wind, and engine) and then trains classification models on this data. Features are extracted in both the time and frequency domains to analyze the signal from multiple perspectives. This program utilizes several scripts to ensure proper data processing: file_separator initializes the dataset in the correct format, extractAllFeats extracts all features in a compact and reliable manner, safe_normalize normalizes the data matrices and knnTrainer trains the models for classification.

file_separator

```
file_separator(masterFolderPath, folderNames)
```

Initializes the dataset by creating, for each considered class, two subfolders: 'training' and 'testing'. The files of each class are randomly shuffled, with 70% allocated to the 'training' folder and the remaining 30% to the 'testing' folder. The entire dataset and the newly created folders (if they did not already exist) are then added to the MATLAB path.

extractAllFeats

```
[TrFeatF, TrFeatT, TeFeatF, TeFeatT] = extractAllFeats(subFolPath, wL, sL)
```

It utilizes the functions freqFeatures and timedomainFeats to extract the relevant features in an organized manner from the given path *'subFolPath'*. The function returns four matrices, separating training and testing features as well as frequency-domain features from time-domain features.

safe_normalize

```
[matrix, mn, st] = safe_normalize(matrix)
```

This function normalizes the input matrix by transposing it, subtracting the mean *'mn'* from each value, and dividing by the standard deviation *'std'*. It returns the normalized matrix along with the mean and standard deviation, which can be reused in subsequent code.

knnTrainer

```
[recognRate, Mdl] = knnTrainer(trainFeat, testFeat, labels, ground_truth, k)
```

By associating *'trainFeat'* with corresponding *'labels'* that identify the class of each feature, a classification model (*Mdl*) is trained. The model is then tested on *'testFeat'*, which are linked to labels not visible to the model (*ground_truth*). The model's performance is recorded in the *recognRate* vector for a given number of neighbors *k(ind)*.

Algorithm used

k-NN

The k-Nearest Neighbors (k-NN) algorithm is a classifier that uses proximity to make classifications or predictions about the grouping of a single data point. This algorithm belongs to the family of **"lazy learning"** models, meaning that it only stores a training dataset instead of undergoing a dedicated training phase; all computations are performed only when a classification or prediction is needed. The key aspect of this algorithm is training the classification model on a quality dataset, as all future calculations will rely on it. k-NN is one of the simplest yet most accurate algorithms; however, as the dataset grows, k-NN becomes increasingly inefficient, which can compromise the overall performance of the model.

Testing

The goal of the k-NN algorithm is to identify the nearest neighbors of a given frame (by evaluating all its features) to assign a class label to that point. The most common method used to compute the distance between frames is the Euclidean distance, which is limited to real-valued vectors.
The parameter k in the k-NN algorithm defines the number of neighbors considered when determining the classification of a specific query point. Setting the k value is a crucial step, as lower values of k may result in high variance but low bias, while higher values may lead to high bias and lower variance (this information will be useful later). One major limitation of this algorithm in terms of efficiency is dimensionality reduction: once an "optimal" number of features is reached, where the model performs correctly, adding more features could increase classification errors instead of improving performance.

PCA

The PCA algorithm summarizes the informational content of large datasets into a smaller set of uncorrelated variables, known as principal components. These components are called "principal" because they capture as much information as possible from the original dataset.
PCA is highly effective for visualizing and exploring high-dimensional datasets or data with many features, as it can easily identify trends, patterns, or outliers.

Results

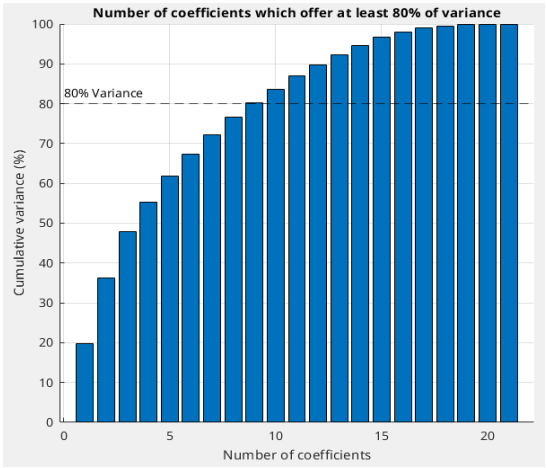
Optimization

The first step in achieving good results from feature extraction is selecting an appropriate window length (*wL*). This parameter significantly affects the DFT process applied to the signal. Longer wL values capture frequency-domain components with greater detail but at the cost of time-domain resolution. Conversely, a shorter wL (on the order of hundredths of a second) provides highly accurate results for time-domain variations but reduces frequency-domain detail. For the analyzed classes, a window length of *0.3s* was chosen, allowing for precise frequency-domain analysis. This decision was made based on the low temporal variation in the selected classes: Wind, Engine, and Crow signals exhibit stable frequency-domain characteristics.
The second parameter that can be optimized is the step length (*sL*), which defines the time interval (shorter than wL) between the start of one frame and the next. A small sL results in many overlapping frames, leading to redundant data. Conversely, if sL is too close in duration to wL, useful data may be lost, as only the central portion of each frame is analyzed. To balance these extremes, a step length of sL = wL/2 was chosen, ensuring an optimal trade-off between redundancy and data retention.

PCA

The PCA algorithm is useful for reducing the dimensionality of the dataset under analysis by representing the three classes using the three principal components with the highest variance. The function used in this project also returns an array called *explained*, which contains 21 values indicating the variance explained by each extracted feature type. In *Figure I*, the number of coefficients required to exceed 80% variance is illustrated. In this case, selecting only the 9 features that explain the most variance would allow for data representation and classification without losing essential information. This simplification process is particularly useful when working with a large number of features, as it helps prevent overfitting. **Overfitting** occurs when irrelevant information is analyzed, making the classification process unnecessarily complex and reducing model generalization.

Figure I



In the 3D plot shown alongside (*Figures II and III*), all frames are represented according to the three principal components. The first component accounts for 20% of the variance, the second for 16%, and the third for 11%. The three colors represent the three classes under analysis. Notably, for the Crow class (*red*), some frames form a distinct cluster in the lower right; this is indeed the only class with a different "timbre" compared to the others. The Wind (*yellow*) and Engine (*violet*) classes are positioned close to each other. Some Crow frames are depicted near the Wind class since, upon listening to the Crow audio files, there is an audible presence of wind. The plot also shows isolated frames known as **outliers**. These features can affect data analysis (mean, variance, etc.), so it is good practice to identify them and, depending on the situation, either remove or analyze them further as they may contain interesting information.

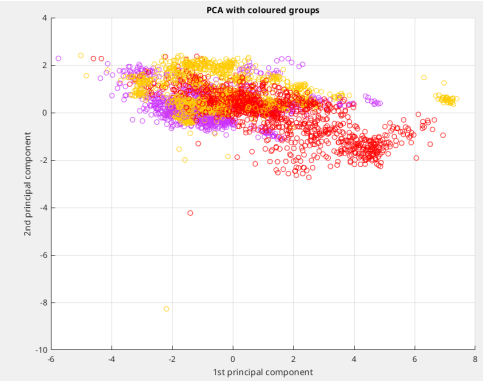


Figure II

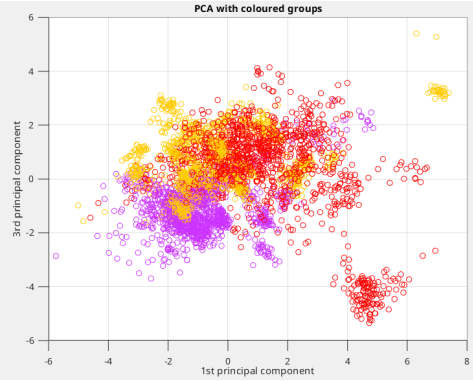


Figure III

KNN

The *Figure IV* below displays three graphs illustrating the recognition rate as a function of the number of neighbors used in the k-Nearest Neighbors algorithm.

In the first graph, which relates to **time-domain features**, the recognition rate starts at a rather low value and shows some instability for small values of k. However, as k increases, the performance gradually improves, reaching a maximum value of around 61% for higher k values.

The second graph, representing the analysis based on **frequency-domain features**, displays a different trend. Here, the recognition rate quickly increases, reaching a maximum of about 83% for k values near 50. However, beyond this point, the performance progressively deteriorates, indicating that an excessively high number of neighbors may reduce the model's effectiveness, likely due to increased class overlap.

The third graph, which utilizes the **combined features**, follows a trend similar to that of the frequency-domain features alone. The recognition rate initially improves, reaching a peak of approximately 78% for k around 50. Yet, beyond this threshold, there is a significant drop in performance.

In **conclusion**, too low a k value may make the model overly sensitive to the training data, whereas too high a k value may lead to a loss of precision due to difficulty in accurately distinguishing between classes. Among the different feature configurations, the frequency-domain features appear to offer the best performance, achieving a higher maximum recognition rate compared to the other two approaches. Overall, a k value of around 50 seems to be the best compromise for obtaining good performance without sacrificing classification accuracy.

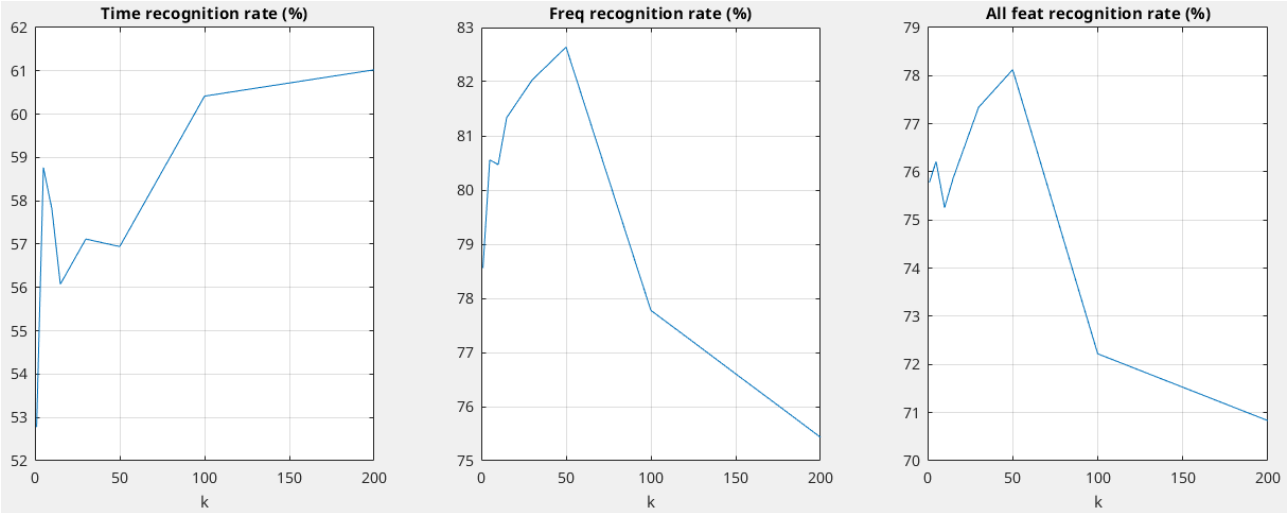


Figure IV

Documentation

PCA: https://www.ibm.com/it-it/topics/principal-component-analysis
kNN: https://www.ibm.com/it-it/topics/knn