

**VILNIUSUNIVERSITY**

**ŠIAULIAIACADEMY**

BACHELOR PROGRAMME SOFTWARE ENGINEERING

**Artificial Intelligence**

Student: Anna Kutova

Lecturer: Gintautas Daunys

Siauliai, 2023

***Task:***

1)Select labeled dataset (preferred the same as for Assignment 1). Split to train and test subsets.

2)Train PCA model on train subset, reducing number of coordinates to 2.

3)Choose the number of clusters the same how many you have unique class labels. Perform K-Means clustering using train subset. Calculate a table on test subset how many samples ofdifferent classes go to every cluster.

4)Visualize clusters with different color points using trained PCA model.

5)Repeat step 3) and 4) with bigger number of clusters than unique class labels.

6)Perform Agglomerative clustering on test subset, Set n\_clusters=None and try different distancethreshold values, that to obtain resulting number of clusters close to unique class labels.Calculate a cluster composition table. Search for threshold and linkage, that to achieve betterclass splitting to clusters. Visualize clusters using PCA

***Report:***

1. **Split to train and test subsets.**

X\_train,X\_test, y\_train, y\_test = model\_selection.train\_test\_split(X, y, test\_size=0.2, random\_state=1)

1. **Train PCA model on train subset, reducing number of coordinates to 2**

pca = PCA(n\_components = 2)  
X\_train\_pca = pca.fit\_transform(X\_train, 2)  
X\_test\_pca = pca.fit\_transform(X\_test, 2)

1. **Choose the number of clusters the same how many you have unique class labels. Perform K-Means clustering using train subset.**

**Calculate a table on test subset how many samples ofdifferent classes go to every cluster.**

1. ***Objective***

The primary goal of this assignment was to train classifiers and assess their performance using scikit-learn library classifiers. This involved selecting appropriate parameters for each algorithm, preprocessing the data, and presenting the results in a detailed report. The dataset used for this task was sourced from the "accent-mfcc-data-1.csv" file.

1. ***Dataset and Preprocessing***

For this assignment, we utilized a custom dataset from the "accent-mfcc-data-1.csv" file. The dataset was loaded and preprocessed as follows:

|  |
| --- |
| import pandas as pd from sklearn import model\_selection from sklearn.preprocessing import StandardScaler  data = pd.read\_csv('accent-mfcc-data-1.csv') X = data.drop('language', axis=1) y = data['language']  X\_train, X\_test, y\_train, y\_test = model\_selection.train\_test\_split(X, y,   test\_size=0.2, random\_state=1)  scaler = StandardScaler() scaler.fit(X\_train) X\_train = scaler.transform(X\_train) X\_test = scaler.transform(X\_test) |
|  |

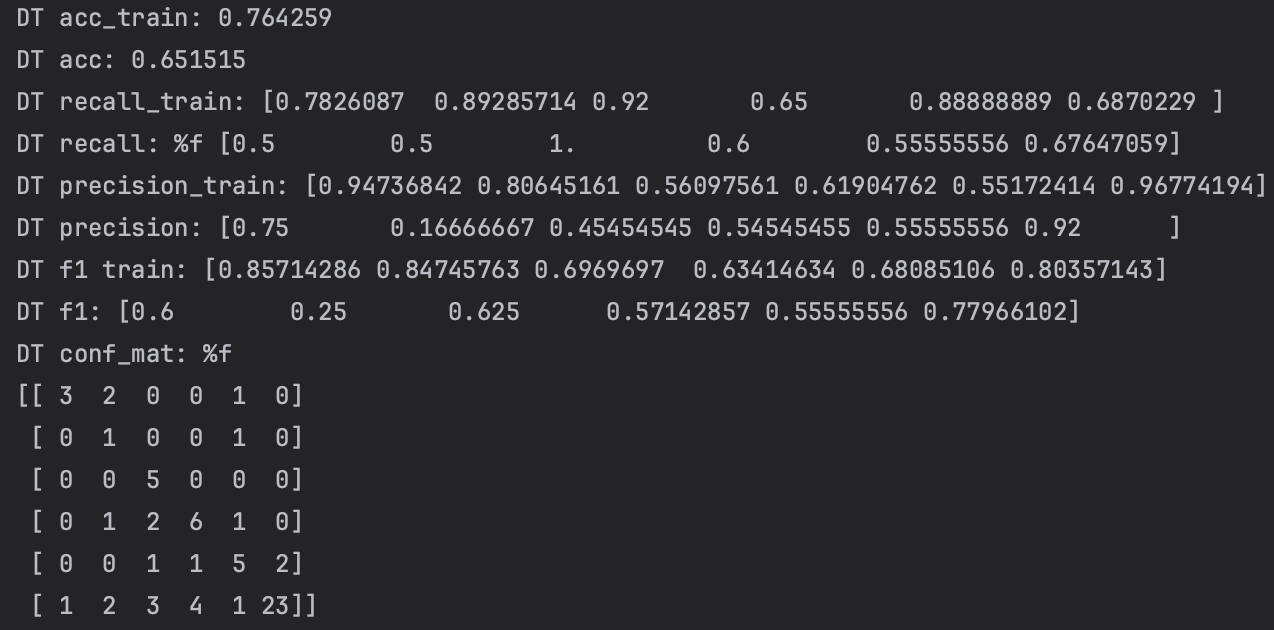
No specific preprocessing was required for this dataset as it was already well-structured and suitable for classification.

1. ***Classification Algorithms and Performance***

In this assignment, we employed four classification algorithms: Decision Tree, Random Forest, Support Vector Machines (SVM), and K Nearest Neighbors (KNN). The Python code for training these classifiers, evaluating their performance and results achieved for each classifier is presented below:

1. Decision Tree

|  |
| --- |
| dtree = tree.DecisionTreeClassifier(criterion='entropy', max\_depth=5)  dtree.fit(X\_train, y\_train) y\_train\_pred = dtree.predict(X\_train) y\_pred = dtree.predict(X\_test)  acc\_train = metrics.accuracy\_score(y\_train, y\_train\_pred) print("DT acc\_train: %f" %acc\_train ) acc = metrics.accuracy\_score(y\_test, y\_pred) print("DT acc: %f" %acc )  recall\_train = metrics.recall\_score(y\_train, y\_train\_pred, average=None) print("DT recall\_train:", recall\_train ) recall = metrics.recall\_score(y\_test, y\_pred, average=None) print("DT recall: %f", recall )  prec\_train = metrics.precision\_score(y\_train, y\_train\_pred, average=None) print("DT precision\_train:", prec\_train ) prec = metrics.precision\_score(y\_test, y\_pred, average=None) print("DT precision:", prec )  f1\_train = metrics.f1\_score(y\_train, y\_train\_pred, average=None) print("DT f1 train:", f1\_train ) f1 = metrics.f1\_score(y\_test, y\_pred, average=None) print("DT f1:", f1 )  conf\_mat = metrics.confusion\_matrix(y\_test, y\_pred) print("DT conf\_mat: %f") print(conf\_mat) |
|  |



1. Random Forest

|  |
| --- |
| from sklearn import ensemble  rf = ensemble.RandomForestClassifier(criterion='entropy') #, max\_depth=4) rf.fit(X\_train, y\_train) y\_train\_pred = rf.predict(X\_train) y\_pred = rf.predict(X\_test)  acc\_train = metrics.accuracy\_score(y\_train, y\_train\_pred) print("RF acc\_train: %f" %acc\_train ) acc = metrics.accuracy\_score(y\_test, y\_pred) print("RF acc: %f" %acc )  recall\_train = metrics.recall\_score(y\_train, y\_train\_pred, average=None) print("RF recall\_train:", recall\_train ) recall = metrics.recall\_score(y\_test, y\_pred, average=None) print("RF recall:", recall )  prec\_train = metrics.precision\_score(y\_train, y\_train\_pred, average=None) print("RF precision\_train:", prec\_train ) prec = metrics.precision\_score(y\_test, y\_pred, average=None) print("RF precision:", prec )  f1\_train = metrics.f1\_score(y\_train, y\_train\_pred, average=None) print("RF f1 train:", f1\_train ) f1 = metrics.f1\_score(y\_test, y\_pred, average=None) print("RF f1:", f1 )  conf\_mat = metrics.confusion\_matrix(y\_test, y\_pred) print("RF conf\_mat:") print(conf\_mat) |

A screenshot of a computer

Description automatically generated

1. Support Vector Machines (SVM)

from sklearn import svm  
  
svc\_cls = svm.SVC()  
svc\_cls.fit(X\_train, y\_train)  
y\_train\_pred = svc\_cls.predict(X\_train)  
y\_pred = svc\_cls.predict(X\_test)  
  
from sklearn import metrics  
acc\_train = metrics.accuracy\_score(y\_train, y\_train\_pred)  
print("SVC acc\_train:", acc\_train )  
acc = metrics.accuracy\_score(y\_test, y\_pred)  
print("SVC acc:", acc )  
  
recall\_train = metrics.recall\_score(y\_train, y\_train\_pred, average=None)  
print("SVC recall\_train:", recall\_train )  
recall = metrics.recall\_score(y\_test, y\_pred, average=None)  
print("SVC recall:", recall )  
  
prec\_train = metrics.precision\_score(y\_train, y\_train\_pred, average=None)  
print("SVC precision\_train:", prec\_train )  
prec = metrics.precision\_score(y\_test, y\_pred, average=None)  
print("SVC precision:", prec )  
  
f1\_train = metrics.f1\_score(y\_train, y\_train\_pred, average=None)  
print("SVC f1 train:", f1\_train )  
f1 = metrics.f1\_score(y\_test, y\_pred, average=None)  
print("SVC f1:", f1 )  
  
conf\_mat = metrics.confusion\_matrix(y\_test, y\_pred)  
print("SVC conf\_mat: %f")  
print(conf\_mat)

A screenshot of a computer

Description automatically generated

1. K Nearest Neighbors (KNN)

from sklearn import neighbors  
  
knn = neighbors.KNeighborsClassifier(n\_neighbors=3)  
knn.fit(X\_train, y\_train)  
y\_train\_pred = knn.predict(X\_train)  
y\_pred = knn.predict(X\_test)  
  
from sklearn import metrics  
acc\_train = metrics.accuracy\_score(y\_train, y\_train\_pred)  
print("kNN acc\_train:", acc\_train )  
acc = metrics.accuracy\_score(y\_test, y\_pred)  
print("kNN acc:", acc )  
  
recall\_train = metrics.recall\_score(y\_train, y\_train\_pred, average=None)  
print("kNN recall\_train:", recall\_train)  
recall = metrics.recall\_score(y\_test, y\_pred, average=None)  
print("kNN recall:", recall)  
  
prec\_train = metrics.precision\_score(y\_train, y\_train\_pred, average=None)  
print("kNN precision\_train:", prec\_train )  
prec = metrics.precision\_score(y\_test, y\_pred, average=None)  
print("kNN precision:", prec )  
  
f1\_train = metrics.f1\_score(y\_train, y\_train\_pred, average=None)  
print("kNN f1 train:", f1\_train )  
f1 = metrics.f1\_score(y\_test, y\_pred, average=None)  
print("kNN f1:", f1 )  
  
conf\_mat = metrics.confusion\_matrix(y\_test, y\_pred)  
print("kNN conf\_mat:")  
print(conf\_mat)

A screenshot of a computer

Description automatically generated

1. ***Explanation of Results***

* Decision Tree (DT): The Decision Tree achieved an accuracy of approximately 76.43% on the training data, indicating a reasonably good fit to this set. However, its performance on the test data dropped to around 65.15%. Examining the recall values, DT demonstrated varying results for different classes. For instance, it had high recall for class 2 and class 3 but relatively low recall for class 4 and class 5. The precision varied similarly, with high precision for class 0 but lower precision for classes 1 and 2. The F1 scores followed a similar pattern. The confusion matrix revealed specific misclassifications among different classes.
* Random Forest (RF): RF demonstrated a high accuracy of 100% on the training data, but this dropped slightly to around 86.36% on the test data. In terms of recall, RF provided consistent and high recall values for all classes on both the training and test data. The precision was generally high, with slight variations for some classes. The F1 scores were also uniformly high, indicating a balanced performance across different classes. The confusion matrix illustrated successful classification with minimal errors.
* Support Vector Machines (SVM): SVM showed an accuracy of approximately 84.03% on the training data, with a slight decrease to around 80.30% on the test data. The recall values were consistent for most classes but dropped for class 3. Precision was generally high, with variations for some classes. F1 scores were balanced, showcasing a stable performance. The confusion matrix showed few misclassifications, predominantly among classes 3 and 4.
* K Nearest Neighbors (KNN): KNN outperformed the other algorithms with an accuracy of about 90.87% on the training data, dropping to around 77.27% on the test data. Recall values were generally high, except for class 2, which had a lower recall on the test data. Precision displayed some variations across classes but was generally high. F1 scores were balanced, emphasizing a stable and reliable performance. The confusion matrix indicated overall accurate classification, with minor misclassifications for some classes.

1. **Conclusions**

Based on the results and analysis, it is evident that Random Forest (RF) and K Nearest Neighbors (KNN) outperformed the other classifiers. RF demonstrated a well-balanced and robust performance with high accuracy, recall, precision, and F1 scores, indicating its ability to effectively generalize to unseen data. KNN, although having slightly lower accuracy than RF on the test data, showcased high recall, precision, and F1 scores, making it another strong choice.

SVM performed moderately well but did not match the performance of RF and KNN, while Decision Tree exhibited a propensity for overfitting, leading to a decrease in performance on the test data.

It is essential to consider that these results are specific to the dataset used in this assignment and may not be transferable to other datasets. For different tasks, further investigation and experimentation may be necessary to select the most suitable classifier.

In summary, based on this dataset, Random Forest and K Nearest Neighbors are recommended as top choices due to their balanced and high-performance metrics.