Applied Machine Learning Project

Pages 1–23
2022

Tox21 project report

Arash Sajjadi

Applied Machine Learning

Shahid Beheshti University

February 23, 2022

Keywords: Machine learning, Machine learning algorithms, K-NN,K-nearest neighbors,Support vector machine,svm,Gridsearchcv, Boosting, Bagging ,AdaBoost, Ensemble, Random forest, Decision Tree, Python, MLP, Balanced accuracy, Confusion matrix

ABSTRACT

The raw Tox21 data in this report are related to the toxicity of various chemicals. The number of raw data is 7830. Labels are 12-dimensional vectors, each dimension indicating whether the chemical molecule is toxic or not from the point of view of that dimension. My task is to find a function that can compute the prediction of this 12-dimensional vector. [1] book, I tried to figure out the relation between the data and whether they were effective in treating HIV or not. I hope you enjoy the following content.

CONTENTS

PRE-INTRODUCTION

Toxicity is an adverse health effect caused by a drug. The adverse effect can range from a minor unpleasant side effect to a major threat to the life quality, health, or survival of the patient. [2]

You may be would like to know why it is essential for us to check the toxicity of chemicals. Knowing whether a chemical can cause cancer, allergic reactions, or abnormalities in unborn children is vital to human health. The process of discovering this information is known as the assessment of hazard.

1 INTRODUCTION

My data contains two files named `tox21_global_cdf_rdkit` and `tox21`. The first one is for the properties of each data, and the second one is for the labels. Also, the chemical formula of molecules is written in strings format, called Smiles format. For example, the eighth data refers to the chemical molecule in Figure 1.[3]

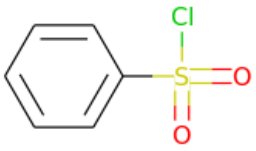

Figure 1: The chemical formula of the eighth data

The features file contains a small number of empty variables. For this reason, I can delete the rows containing the empty variable from the beginning. However, the labels file contains many vectors that have at least one empty variable. If I want to delete all vectors containing at least one empty variable, only 3054 [1] data will remain. To train a good

model, I have to think of a better idea. I will mention this idea in the DESCRIPTION OF RAW DATA section.

2 DESCRIPTION OF RAW DATA

2.1 Labels distribution

As I mentioned before, each data label is a 12-dimensional vector. Each dimension informs us of the toxicity of that chemical in binary form. The labels of each dimension of data have an imbalanced distribution. Figure 2 shows the distribution of each dimension for us. Of course, these charts have been prepared after the clever removal of missing data, which I will explain in more detail in the next subsection.

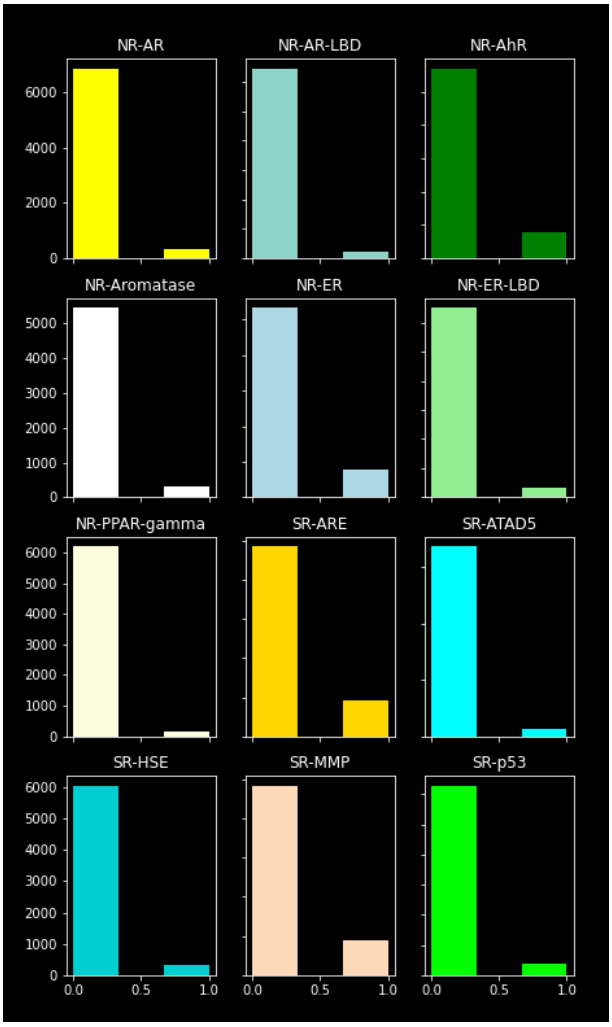

Figure 2: Labels distribution [4]

2.2 Missing data

I have probably been able to describe the challenge that missing data poses to us so far. To deal with this problem, instead of eliminating the rows containing empty data, we examine each dimension separately and remove only the empty data related to that dimension.

$$\begin{bmatrix} \text{NR-AR} & \text{NR-AR-LBD} & \dots & \text{SR-p53} \\ \not{\text{NaN}} & \not{\text{NaN}} & \dots & \not{\text{NaN}} \\ \not{\text{NaN}} & \not{\text{NaN}} & \dots & \not{\text{NaN}} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & \dots & 0 \end{bmatrix} \mapsto \begin{bmatrix} \text{NR-AR} \\ \not{\text{NaN}} \\ 0 \\ \vdots \\ 1 \end{bmatrix}, \begin{bmatrix} \text{NR-AR-LBD} \\ 1 \\ \not{\text{NaN}} \\ \vdots \\ 0 \end{bmatrix}, \dots, \begin{bmatrix} \text{SR-p53} \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

As you can see in Figure 2, none of the labels are distributed in a balanced manner. The data imbalance causes us to have other problems, for which a solution has been predicted. For example, in the case of separating the data set into the training set and test set, we must use the stratify data option when separating the data.[5] you can see more detail in Listing 1

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.simplefilter('ignore')
#importing pakcages
initial_Features=pd.read_csv('tox21_global_cdf_rdkit.
    csv')
initial_dataset=pd.read_csv('tox21.csv')
#import data
initial_Features=initial_Features.loc[:,
    initial_Features.apply(pd.Series.nunique) != 1]
initial_dataset=initial_dataset.iloc[initial_Features.
    dropna().index]
#algebratic dimension reduction
initial_dataset=initial_dataset.reset_index()
initial_Features=initial_Features.dropna()
initial_Features=initial_Features.reset_index()
index_array=[]
for i in np.arange(1,13):
    index_array.append(initial_dataset.iloc[:,i+1].
    dropna().index)
def label_ith(i):
    return pd.DataFrame(data=initial_dataset.iloc[
    index_array[i]].iloc[:,i+2])
def Feature_ith(i):
    return initial_Features.iloc[index_array[i]].drop(
    'index',axis=1)
#define functions to sync Nan removed labels with
    features
from sklearn.model_selection import train_test_split
X_training_data=[]
X_test=[]
y_training_data=[]
y_test=[]
for i in np.arange(0,12):
    X_training_data_tmp, X_test_tmp,
    y_training_data_tmp, y_test_tmp =train_test_split(
    Feature_ith(i),label_ith(i), stratify=label_ith(i)
    ,test_size=0.10,random_state=1234)
    X_training_data.append(X_training_data_tmp)
    X_test.append(X_test_tmp)
    y_training_data.append(y_training_data_tmp)
    y_test.append(y_test_tmp)
#split, sync data as we saw in the matrix before.
```

Listing 1: Import packages Deal with missing data Data separation [6] [7]

2.3 Algebraic dimension of features

Raw data is 200 dimensions. I quickly noticed that some of the columns of the matrix feature show a fixed number. By removing these columns, we have 190 algebraic dimensions.[Refer to Listing 1] Also, the data was normalized from the beginning. 201st column was related to the data normalization report, which I did not consider as a separate dimension from the beginning.

3 DATA SEPARATION

It is clear that in machine learning projects, one of the first tasks is to separate the test set. I must do this before any data analysis. If the data separation is to be done after PCA [2] or anything like that, we have practically used the test set data to find the best directions, which is a mistake. As a result, data separation generally must be done at the beginning of machine learning projects. [3] I also intended to allocate 10% of the data volume to the test set.

As Listing 1, there are functions called `label_ith` and `Feature_ith`, which give us labels and data synced with nonempty labels, respectively. Also, in each cell, the `X_training_data`, `y_training_data`, `X_test`, and `y_test` arrays, as their names, contain information synced to the same label dimension.

4 DIMENSIONALITY REDUCTION

High-dimensional data has some peculiar characteristics, some of which are counterintuitive. For example, in high dimensions the center of the space is devoid of points, with most of the points being scattered along the surface of the space or in the corners. There is also an apparent proliferation of orthogonal axes. Consequently, high-dimensional data can cause problems for data mining and analysis, although in some cases, high-dimensionality can help, for example, for nonlinear classification. Nevertheless, it is essential to check whether the dimensionality can be reduced while preserving the crucial properties of the full data matrix. One of the critical methods for dimension reduction is PCA.

Principal Component Analysis (PCA) is a technique that seeks an r-dimensional basis that best captures the variance in the data. The direction with the largest projected variance is called the first principal component. The orthogonal direction that captures the second largest projected variance is called the second principal component, and so on. As we shall see, the direction that maximizes the variance is also the one that minimizes the mean squared error [8]

Since I have prepared arrays such as `X_training_data`, etc., for each dimension of the label vector, applying the PCA algorithm separately for each member of these arrays is better. You can also see the results of applying this algorithm in Listing 2. It is also evident that the test set information for the PCA algorithm has not been used in any way. I found that my data with 71 principal components can maintain more than 95% of the data variance in each of the 12 data sets.

```
from sklearn.decomposition import PCA
X_training_data_pca=[]
X_test_pca=[]
for i in np.arange(0,12):
    pca = PCA(n_components=71)
    principalComponents = pca.fit_transform(
    X_training_data[i])
    X_training_data_PCA_tmp = pd.DataFrame(data =
    principalComponents)
    X_training_data_pca.append(X_training_data_PCA_tmp
    )
    X_test_pca_tmp=pd.DataFrame(data=pca.transform(
    X_test[i]))
    X_test_pca.append(X_test_pca_tmp)
    print('PCA with 71 principal components retains',
    np.sum(pca.explained_variance_ratio_)*100,'% of
    data VAR. (It is related for',i,'th label)')
```

Listing 2: PCA [5]

```
PCA with 71 principal components retains 95.14902477458284 % of data VAR. (It is
    related for 0 th label)
PCA with 71 principal components retains 95.18528509261213 % of data VAR. (It is
    related for 1 th label)
PCA with 71 principal components retains 95.23516716185081 % of data VAR. (It is
    related for 2 th label)
PCA with 71 principal components retains 95.24899205532745 % of data VAR. (It is
    related for 3 th label)
PCA with 71 principal components retains 95.19568949095981 % of data VAR. (It is
    related for 4 th label)
PCA with 71 principal components retains 95.18825061528679 % of data VAR. (It is
    related for 5 th label)
PCA with 71 principal components retains 95.13894464393015 % of data VAR. (It is
    related for 6 th label)
PCA with 71 principal components retains 95.21618305641543 % of data VAR. (It is
    related for 7 th label)
PCA with 71 principal components retains 95.18018989506741 % of data VAR. (It is
    related for 8 th label)
PCA with 71 principal components retains 95.16545227328498 % of data VAR. (It is
    related for 9 th label)
PCA with 71 principal components retains 95.22548243540908 % of data VAR. (It is
    related for 10 th label)
PCA with 71 principal components retains 95.17820123387547 % of data VAR. (It is
    related for 11 th label)
```

Listing 3: PCA results

5 CORRELATION OF LABELS

Before running the algorithms, I was personally asked how correlated the different dimensions of the labels were. I want to talk here as someone who has seen the end of the project. We were much more successful in predicting some labels in proportion to some others. So if we could find a high correlation between the labels, it would be possible to use the result of more successful predictions for less accurate predictions. But as you can see in Figures 3 and 4, there is not much correlation between the labels.

Is it not harmful to explain the difference between Figures 3 and 4. Figure 3 shows the correlation between the raw data. We know that the raw data contained a significant amount of empty data. Figure 4 shows the correlation of non-empty data labels. What is clear is that the maximum correlation between the data is 0.551974651, which clearly indicates that different data labels are not going to help us predict other labels.

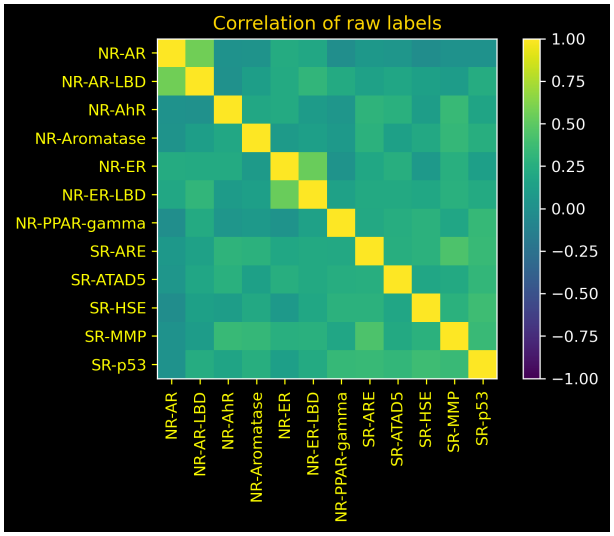

Figure 3: Correlation of the raw labels [4]

6 ALGORITHMS

In this section, I will execute each classification algorithm for each dimension of the label vectors with the help of cross-validation [4] and share the results with you. One of my efforts during this project has been to keep the code short for better readability. I hope you enjoy these beautiful algorithms.

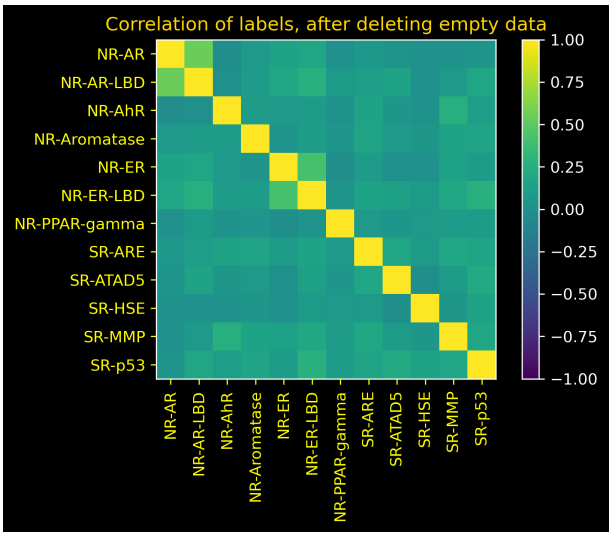

Figure 4: Correlation of labels, after deleting empty data [4]

Scoring meter

Given the nature of the imbalanced data, I decided to choose the balanced accuracy criterion to evaluate the results of my algorithms. This accuracy measurement is defined as follows.

$$\textbf{Balanced Accuracy} = \frac{1}{2}\left(\frac{\text{TP}}{\text{P}} + \frac{\text{TN}}{\text{N}}\right) \quad 5$$

Of course, if necessary, I will use the confusion matrix [Refer to Figure 5] to analyze the algorithms.

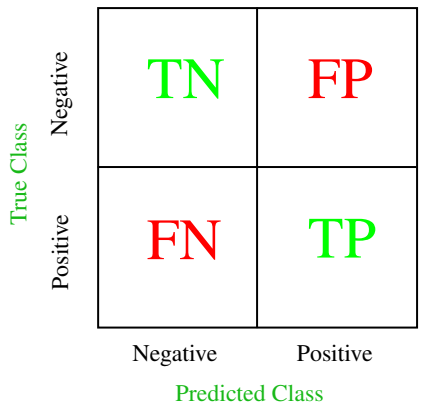

Figure 5: confusion matrix[9][6]

6.1 K Nearest Neighbors

Nearest Neighbor algorithms are among the simplest of all machine learning algorithms. The idea is to memorize the training set and then to predict the label of any new instance based on the labels of its closest neighbors in the training set. The rationale behind such a method is based on the assumption that the features that are used to describe the domain points are relevant to their labelings in a way that makes close-by points likely to have the same label. Furthermore, in some situations, even when the training set is immense, finding the nearest neighbor can be done extremely fast [7]. [1] You can see the execution of KNN on 77-dimensional data (after PCA application) in Listing 4.

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix,
    plot_confusion_matrix
print("KNeighborsClassifier:")
for i in np.arange(0,12,1):
    from sklearn.neighbors import KNeighborsClassifier
    knn = KNeighborsClassifier()
    param_grid={'n_neighbors':np.arange(1,11,2),'
    metric':['euclidean','cosine','manhattan']}
    grid=GridSearchCV(knn,param_grid=param_grid,cv=10,
    scoring='balanced_accuracy',return_train_score=
    False)
    grid.fit(X_training_data_pca[i].to_numpy(),
    y_training_data[i])
    print("---This information is for the ",i,"th
    label.---")
    print("-----------------Including PCA
    .-----------------")
    print("best mean cv score= ",grid.best_score_)
    print("best parameters= ",grid.best_params_)
    y_pred_knn_pca=grid.best_estimator_.predict(
    X_test_pca[i])
    print(confusion_matrix(y_test[i],np.transpose(np.
    matrix(y_pred_knn_pca))))
    print(grid.score(X_test_pca[i],y_test[i]))
```

Listing 4: GridSearchCV-KNN-PCA [5]

```
KNeighborsClassifier:
---This information is for the  0 th label.---
-----------------Including PCA.-----------------
best mean cv score=  0.7235540265998088
best parameters=  {'metric': 'manhattan', 'n_neighbors': 7}
[[682   4]
 [ 15  16]]
0.7551490642339885
---This information is for the  1 th label.---
-----------------Including PCA.-----------------
best mean cv score=  0.7850023245692129
best parameters=  {'metric': 'manhattan', 'n_neighbors': 1}
[[637   9]
 [  6  17]]
0.8625992731188585
---This information is for the  2 th label.---
-----------------Including PCA.-----------------
best mean cv score=  0.7259460692370788
best parameters=  {'metric': 'cosine', 'n_neighbors': 1}
[[544  27]
 [ 33  44]]
0.7620715536652489
---This information is for the  3 th label.---
-----------------Including PCA.-----------------
best mean cv score=  0.6768973816233019
best parameters=  {'metric': 'cosine', 'n_neighbors': 1}
[[530  17]
 [ 13  17]]
0.767794028031688
---This information is for the  4 th label.---
-----------------Including PCA.-----------------
best mean cv score=  0.639949171815369
best parameters=  {'metric': 'euclidean', 'n_neighbors': 3}
[[512  22]
 [ 58  21]]
0.6123121414687337
---This information is for the  5 th label.---
-----------------Including PCA.-----------------
best mean cv score=  0.7037870122495336
best parameters=  {'metric': 'euclidean', 'n_neighbors': 1}
[[636  18]
 [ 20  13]]
0.6832082290797887
---This information is for the  6 th label.---
-----------------Including PCA.-----------------
best mean cv score=  0.5970295738998852
best parameters=  {'metric': 'manhattan', 'n_neighbors': 1}
```

```
[[613   8]
 [ 11   6]]
0.670029364402766
---This information is for the  7 th label.---
-----------------Including PCA.-----------------
best mean cv score=  0.6628518308101344
best parameters=  {'metric': 'cosine', 'n_neighbors': 1}
[[431  55]
 [ 45  46]]
0.696162890607335
---This information is for the  8 th label.---
-----------------Including PCA.-----------------
best mean cv score=  0.6327067238634185
best parameters=  {'metric': 'euclidean', 'n_neighbors': 1}
[[660  15]
 [ 17   8]]
0.6488888888888888
---This information is for the  9 th label.---
-----------------Including PCA.-----------------
best mean cv score=  0.6296254938216598
best parameters=  {'metric': 'manhattan', 'n_neighbors': 1}
[[578  26]
 [ 22  13]]
0.6641911069063386
---This information is for the  10 th label.---
-----------------Including PCA.-----------------
best mean cv score=  0.7639088766366591
best parameters=  {'metric': 'cosine', 'n_neighbors': 1}
[[443  41]
 [ 32  58]]
0.7798668503213958
---This information is for the  11 th label.---
-----------------Including PCA.-----------------
best mean cv score=  0.6857203682733718
best parameters=  {'metric': 'cosine', 'n_neighbors': 1}
[[608  21]
 [ 23  18]]
0.7028190313699639
```

Listing 5: GridSearchCV-KNN-PCA results

I implemented the KNN algorithm without reducing the PCA dimension on the data, but the desired result is usually better on the KNN algorithm with the PCA dimension reduction algorithm. Although PCA improves the result in some algorithms such as KNN, on the other hand, some algorithms such as The Decision Tree algorithm can worsen the results. The reason for this is almost apparent given the nature of these two algorithms. Anyway, this is just my guess, and it does not mean that it is true. So I did not use PCA for the algorithms that I guessed PCA would not help.

In Listing 6, you can see the code related to running the KNN algorithm on the data without running the PCA algorithm. You can also compare the results of this performance in Listing 7 with the written results in the Listing 5.

```
print("KNeighborsClassifier:")
for i in np.arange(0,12,1):
    knn = KNeighborsClassifier()
    param_grid={'n_neighbors':np.arange(1,11,2),'metric':['euclidean','cosine','manhattan']}
    grid=GridSearchCV(knn,param_grid=param_grid,cv=10,scoring='balanced_accuracy',return_train_score=False)
    grid.fit(X_training_data[i].to_numpy(),y_training_data[i])
    print("---This information is for the ",i,"th label.---")
    print("best mean cv score= ",grid.best_score_)
    print("best parameters= ",grid.best_params_)
    y_pred_knn=grid.best_estimator_.predict(X_test[i])
    print(confusion_matrix(y_test[i],np.transpose(np.matrix(y_pred_knn))))
    print(grid.score(X_test[i],y_test[i]))
```

Listing 6: GridSearchCV-KNN [5]

```
KNeighborsClassifier:
---This information is for the  0 th label.---
best mean cv score=  0.7309100650238083
best parameters=  {'metric': 'manhattan', 'n_neighbors': 9}
[[683   3]
 [ 15  16]]
0.7558779272077494
---This information is for the  1 th label.---
best mean cv score=  0.7846450333353117
best parameters=  {'metric': 'manhattan', 'n_neighbors': 1}
[[634  12]
 [  6  17]]
0.8602772916947099
---This information is for the  2 th label.---
best mean cv score=  0.7272648019793841
best parameters=  {'metric': 'manhattan', 'n_neighbors': 1}
[[545  26]
 [ 32  45]]
0.7694407169013123
---This information is for the  3 th label.---
best mean cv score=  0.6781417541315211
best parameters=  {'metric': 'manhattan', 'n_neighbors': 1}
[[528  19]
 [ 13  17]]
0.7659658744667885
---This information is for the  4 th label.---
best mean cv score=  0.6430757718521802
best parameters=  {'metric': 'manhattan', 'n_neighbors': 3}
[[505  29]
 [ 59  20]]
0.5994287204285782
---This information is for the  5 th label.---
best mean cv score=  0.7049780678881433
best parameters=  {'metric': 'euclidean', 'n_neighbors': 1}
[[637  17]
 [ 20  13]]
0.6839727550736725
---This information is for the  6 th label.---
best mean cv score=  0.6020852568911459
best parameters=  {'metric': 'manhattan', 'n_neighbors': 1}
[[611  10]
 [ 11   6]]
0.6684190584446339
---This information is for the  7 th label.---
best mean cv score=  0.6631432384485604
best parameters=  {'metric': 'cosine', 'n_neighbors': 1}
[[434  52]
 [ 45  46]]
0.6992493103604215
---This information is for the  8 th label.---
best mean cv score=  0.6377311937912507
best parameters=  {'metric': 'manhattan', 'n_neighbors': 1}
[[660  15]
 [ 19   6]]
0.6088888888888888
---This information is for the  9 th label.---
best mean cv score=  0.6371383700958552
best parameters=  {'metric': 'manhattan', 'n_neighbors': 1}
[[579  25]
 [ 23  12]]
0.6507332071901608
---This information is for the  10 th label.---
best mean cv score=  0.7758465460133529
best parameters=  {'metric': 'manhattan', 'n_neighbors': 1}
[[447  37]
 [ 31  59]]
0.78955463728191
---This information is for the  11 th label.---
best mean cv score=  0.6895577029999881
best parameters=  {'metric': 'manhattan', 'n_neighbors': 1}
[[610  19]
 [ 22  19]]
```

```
0.7166039784404203
```

Listing 7: GridSearchCV-KNN results

6.2 Decision Tree

Decision Tree learning or induction of Decision Trees is one of the predictive modelling approaches used in statistics, data mining and machine learning. It uses a Decision Tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). Tree models where the target variable can take a discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision Trees where the target variable can take continuous values (typically real numbers) are called regression trees. Decision Trees are among the most popular machine learning algorithms given their intelligibility and simplicity. [10]

One of the algorithms I applied to the data (without PCA interference) was the Decision Tree algorithm. It should be noted that, as I mentioned earlier, my results show that PCA dimensionally reduced data do not make outstanding results with the Decision Tree. I think it's because the Decision Tree itself has a feature selection, but all the features are somehow important when we use PCA.However, because the results were not desirable, I decided to avoid writing the results of the Decision Tree algorithm on the data with PCA.

```
print("DecisionTreeClassifier:")
from sklearn.tree import DecisionTreeClassifier,
    export_graphviz
for i in np.arange(0,12,1):
    DecisionTree = DecisionTreeClassifier()
    param_grid={'criterion':['gini','entropy'],'
    max_depth':np.arange(4,30,1)}
    grid=GridSearchCV(DecisionTree,param_grid=
    param_grid,cv=10,scoring='balanced_accuracy',
    return_train_score=False)
    grid.fit(X_training_data[i].to_numpy(),
    y_training_data[i])
    print("---This information is for the ",i,"th
    label.--------")
    print("best mean cv score= ",grid.best_score_)
    print("best parameters= ",grid.best_params_)
    y_pred_dt=grid.best_estimator_.predict(X_test[i])
    print(confusion_matrix(y_test[i],np.transpose(np.
    matrix(y_pred_dt))))
    print(grid.score(X_test[i],y_test[i]))
```

Listing 8: GridSearchCV-Decision Tree [5]

```
DecisionTreeClassifier:
----------------------This information is for the  0 th label
    .----------------------------
best mean cv score=  0.7343343187519519
best parameters=  {'criterion': 'entropy', 'max_depth': 4}
[[682   4]
 [ 15  16]]
0.7551490642339885
----------------------This information is for the  1 th label
    .----------------------------
best mean cv score=  0.7493516924842225
best parameters=  {'criterion': 'gini', 'max_depth': 27}
[[637   9]
 [  7  16]]
0.8408601426840758
----------------------This information is for the  2 th label
    .----------------------------
best mean cv score=  0.715454431723965
best parameters=  {'criterion': 'gini', 'max_depth': 27}
[[534  37]
 [ 34  43]]
0.7468214797461732
----------------------This information is for the  3 th label
    .----------------------------
best mean cv score=  0.6357916597143412
best parameters=  {'criterion': 'gini', 'max_depth': 22}
[[523  24]
 [ 19  11]]
0.6613954905545399
----------------------This information is for the  4 th label
    .----------------------------
best mean cv score=  0.6307871144050721
best parameters=  {'criterion': 'entropy', 'max_depth': 13}
[[487  47]
 [ 53  26]]
0.6205494713886124
----------------------This information is for the  5 th label
    .----------------------------
best mean cv score=  0.6988894876519602
best parameters=  {'criterion': 'entropy', 'max_depth': 28}
[[627  27]
 [ 23  10]]
0.6308729496802892
----------------------This information is for the  6 th label
    .----------------------------
best mean cv score=  0.5994250886439558
best parameters=  {'criterion': 'gini', 'max_depth': 28}
[[607  14]
 [ 10   7]]
0.6946102112342522
----------------------This information is for the  7 th label
    .----------------------------
best mean cv score=  0.6527794346669296
best parameters=  {'criterion': 'gini', 'max_depth': 21}
[[431  55]
 [ 53  38]]
0.6522068466512911
----------------------This information is for the  8 th label
    .----------------------------
best mean cv score=  0.617416934552133
best parameters=  {'criterion': 'gini', 'max_depth': 28}
[[652  23]
 [ 18   7]]
0.6229629629629629
----------------------This information is for the  9 th label
    .----------------------------
best mean cv score=  0.6111641916613375
best parameters=  {'criterion': 'entropy', 'max_depth': 21}
[[575  29]
 [ 24  11]]
0.6331362346263009
----------------------This information is for the  10 th
    label.----------------------------
best mean cv score=  0.7517233591876147
best parameters=  {'criterion': 'gini', 'max_depth': 20}
[[448  36]
 [ 37  53]]
0.7572543617998164
----------------------This information is for the  11 th
    label.----------------------------
best mean cv score=  0.6584764787503303
best parameters=  {'criterion': 'gini', 'max_depth': 28}
[[589  40]
 [ 28  13]]
0.6267400829811159
```

Listing 9: GridSearchCV-Decision Tree results

6.3 Random forest

A Random forest is a classifier consisting of a collection of Decision Trees, where each tree is constructed by applying an algorithm A on the training set S and an additional random vector, θ, where θ is sampled i.i.d. [8] from some distribution. The prediction of the Random forest is obtained by a majority vote over the predictions of the individual trees.

With the argument I made for the Decision Tree algorithm, I will omit the Random forest algorithm and the rest of the algorithms written based on the Decision Tree on PCA data. Of course, I checked the background results and just ignored them for reporting. Finally, I would like to draw your attention to the implementation of the Random forest algorithm on the data in Listings 10 and 11.

```
print("RandomForestClassifier:")
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
for i in np.arange(0,12,1):
    RFC = RandomForestClassifier()
    param_grid={'n_estimators':[500,700],'criterion':['gini','entropy'],'max_depth':[4,6,8,10,15,20], 'max_features':['auto','sqrt','log2']}
    grid=GridSearchCV(RFC,param_grid=param_grid,cv=5, scoring='balanced_accuracy',return_train_score=False)
    grid.fit(X_training_data[i].to_numpy(), y_training_data[i])
    print("-----------------------This information is for the ",i,"th label.----------------------------")
    print("best mean cv score= ",grid.best_score_)
    print("best parameters= ",grid.best_params_)
    y_pred_RFC=grid.best_estimator_.predict(X_test[i])
    print(confusion_matrix(y_test[i],np.transpose(np.matrix(y_pred_RFC))))
    print(grid.score(X_test[i],y_test[i]))
```

Listing 10: GridSearchCV-Random forest [5]

```
RandomForestClassifier:
-----------------------This information is for the  0 th label.----------------------------
best mean cv score=  0.7216258089347315
best parameters=  {'criterion': 'gini', 'max_depth': 15, 'max_features': 'auto', 'n_estimators': 700}
[[684   2]
 [ 15  16]]
0.7566067901815103
-----------------------This information is for the  1 th label.----------------------------
best mean cv score=  0.739999752677349
best parameters=  {'criterion': 'entropy', 'max_depth': 10, 'max_features': 'sqrt', 'n_estimators': 500}
[[640   6]
 [  6  17]]
0.8649212545430072
-----------------------This information is for the  2 th label.----------------------------
best mean cv score=  0.6722003183198237
best parameters=  {'criterion': 'entropy', 'max_depth': 20, 'max_features': 'auto', 'n_estimators': 700}
[[560  11]
 [ 46  31]]
0.6916664771305752
-----------------------This information is for the  3 th label.----------------------------
best mean cv score=  0.5665454586815841
best parameters=  {'criterion': 'gini', 'max_depth': 20, 'max_features': 'sqrt', 'n_estimators': 500}
[[544   3]
 [ 20  10]]
0.6639244363193175
-----------------------This information is for the  4 th label.----------------------------
best mean cv score=  0.6182522327025808
best parameters=  {'criterion': 'entropy', 'max_depth': 20, 'max_features': 'auto', 'n_estimators': 500}
[[526   8]
 [ 57  22]]
0.631749869624994
-----------------------This information is for the  5 th label.----------------------------
best mean cv score=  0.6767173252279635
best parameters=  {'criterion': 'entropy', 'max_depth': 15, 'max_features': 'auto', 'n_estimators': 500}
[[652   2]
 [ 23  10]]
0.649986099527384
-----------------------This information is for the  6 th label.----------------------------
best mean cv score=  0.509426763957477
best parameters=  {'criterion': 'gini', 'max_depth': 20, 'max_features': 'auto', 'n_estimators': 500}
[[621   0]
 [ 17   0]]
0.5
-----------------------This information is for the  7 th label.----------------------------
best mean cv score=  0.5917580102590566
best parameters=  {'criterion': 'gini', 'max_depth': 20, 'max_features': 'sqrt', 'n_estimators': 500}
[[480   6]
 [ 61  30]]
0.6586623253289919
-----------------------This information is for the  8 th label.----------------------------
best mean cv score=  0.5349558244652022
best parameters=  {'criterion': 'entropy', 'max_depth': 20, 'max_features': 'auto', 'n_estimators': 500}
[[674   1]
 [ 23   2]]
0.5392592592592592
-----------------------This information is for the  9 th label.----------------------------
best mean cv score=  0.5541034155597723
best parameters=  {'criterion': 'gini', 'max_depth': 20, 'max_features': 'auto', 'n_estimators': 500}
[[603   1]
 [ 30   5]]
0.570600756859035
-----------------------This information is for the  10 th label.----------------------------
best mean cv score=  0.7399732819343566
best parameters=  {'criterion': 'gini', 'max_depth': 20, 'max_features': 'auto', 'n_estimators': 500}
[[469  15]
 [ 41  49]]
0.7567263544536271
-----------------------This information is for the  11 th label.----------------------------
best mean cv score=  0.5619639353507783
best parameters=  {'criterion': 'entropy', 'max_depth': 20, 'max_features': 'sqrt', 'n_estimators': 500}
[[628   1]
 [ 35   6]]
0.5723758191476986
```

Listing 11: GridSearchCV-Random forest results

6.4 XGBoost

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. The same code runs on major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of examples. [11] In the following, I have implemented this beautiful algorithm on my data.

```
from xgboost import XGBClassifier
print("XGBClassifier:")
for i in np.arange(0,12,1):
```

```
    param_grid={'n_estimators':[700,2000],'max_depth'
    :[3,4,5,6,7],'eval_metric':['mlogloss']}
    grid=GridSearchCV(XGBClassifier(),param_grid=
    param_grid,cv=10,scoring='balanced_accuracy',
    return_train_score=False)
    grid.fit(X_training_data[i].to_numpy(),
    y_training_data[i])
    print("---This information is for the ",i,"th
    label.--------")
    print("best mean cv score= ",grid.best_score_)
    print("best parameters= ",grid.best_params_)
    y_pred_xgbc=grid.best_estimator_.predict(X_test[i
    ])
    print(confusion_matrix(y_test[i],np.transpose(np.
    matrix(y_pred_xgbc))))
    print(grid.score(X_test[i],y_test[i]))
```

Listing 12: GridSearchCV-XGBoost [5]

```
XGBClassifier:
---This information is for the  0 th label.--------
best mean cv score=  0.7220923302944042
best parameters=  {'eval_metric': 'mlogloss', 'max_depth': 3,
    'n_estimators': 700}
[[683   3]
 [ 16  15]]
0.7397488949496849
---This information is for the  1 th label.--------
best mean cv score=  0.7588862345935466
best parameters=  {'eval_metric': 'mlogloss', 'max_depth': 6,
    'n_estimators': 2000}
[[640   6]
 [  8  15]]
0.821442993673442
---This information is for the  2 th label.--------
best mean cv score=  0.7333850964654818
best parameters=  {'eval_metric': 'mlogloss', 'max_depth': 5,
    'n_estimators': 2000}
[[557  14]
 [ 36  41]]
0.7539745718379693
---This information is for the  3 th label.--------
best mean cv score=  0.6181325689242583
best parameters=  {'eval_metric': 'mlogloss', 'max_depth': 5,
    'n_estimators': 700}
[[540   7]
 [ 20  10]]
0.6602681291895186
---This information is for the  4 th label.--------
best mean cv score=  0.6417879130291102
best parameters=  {'eval_metric': 'mlogloss', 'max_depth': 3,
    'n_estimators': 2000}
[[519  15]
 [ 54  25]]
0.6441829042810412
---This information is for the  5 th label.--------
best mean cv score=  0.6939448539211255
best parameters=  {'eval_metric': 'mlogloss', 'max_depth': 7,
    'n_estimators': 700}
[[650   4]
 [ 22  11]]
0.6636085626911314
---This information is for the  6 th label.--------
best mean cv score=  0.5692212315899486
best parameters=  {'eval_metric': 'mlogloss', 'max_depth': 3,
    'n_estimators': 700}
[[621   0]
 [ 16   1]]
0.5294117647058824
---This information is for the  7 th label.--------
best mean cv score=  0.6742232870501532
best parameters=  {'eval_metric': 'mlogloss', 'max_depth': 5,
    'n_estimators': 2000}
[[462  24]
 [ 48  43]]
0.7115723782390448
---This information is for the  8 th label.--------
best mean cv score=  0.5970327471111124
best parameters=  {'eval_metric': 'mlogloss', 'max_depth': 4,
    'n_estimators': 700}
[[669   6]
 [ 21   4]]
0.5755555555555556
---This information is for the  9 th label.--------
best mean cv score=  0.5932812079563602
best parameters=  {'eval_metric': 'mlogloss', 'max_depth': 6,
    'n_estimators': 700}
[[601   3]
 [ 27   8]]
0.611802270577105
---This information is for the  10 th label.--------
best mean cv score=  0.7958197053234901
best parameters=  {'eval_metric': 'mlogloss', 'max_depth': 3,
    'n_estimators': 700}
[[462  22]
 [ 33  57]]
0.7939393939939394
---This information is for the  11 th label.--------
best mean cv score=  0.619645691038506
best parameters=  {'eval_metric': 'mlogloss', 'max_depth': 3,
    'n_estimators': 700}
[[624   5]
 [ 30  11]]
0.6301717786653225
```

Listing 13: GridSearchCV-XGBoost results

6.5 C-Support Vector Classification

In machine learning, support-vector machines (SVMs, also support-vector networks[12]) are supervised learning models with associated learning algorithms that analyze data for classification and regression analysis. Developed at ATT Bell Laboratories by Vladimir Vapnik with colleagues.

```
from sklearn.svm import SVC
print("SVC:")
for i in np.arange(0,12,1):
    param_grid={'kernel':['poly'],'degree':np.arange
    (1,15,1)}
    grid=GridSearchCV(SVC(),param_grid=param_grid,cv
    =10,scoring='balanced_accuracy',return_train_score
    =False)
    grid.fit(X_training_data[i].to_numpy(),
    y_training_data[i])
    print("---This information is for the ",i,"th
    label.--------")
    print("best mean cv score= ",grid.best_score_)
    print("best parameters= ",grid.best_params_)
    y_pred_svm=grid.best_estimator_.predict(X_test[i])
    print(confusion_matrix(y_test[i],np.transpose(np.
    matrix(y_pred_svm))))
    print(grid.score(X_test[i],y_test[i]))
```

Listing 14: GridSearchCV-SVC [5]

```
SVC:
---This information is for the  0 th label.--------
best mean cv score=  0.7230681963217643
best parameters=  {'degree': 4, 'kernel': 'poly'}
[[680   6]
 [ 15  16]]
0.7536913382864667
---This information is for the  1 th label.--------
best mean cv score=  0.7546105110095553
best parameters=  {'degree': 5, 'kernel': 'poly'}
[[638   8]
 [  8  15]]
0.8198950060573429
```

```
---This information is for the  2 th label.--------
best mean cv score=  0.7242156849858815
best parameters=  {'degree': 5, 'kernel': 'poly'}
[[546  25]
 [ 38  39]]
0.7313553346828303
---This information is for the  3 th label.--------
best mean cv score=  0.6645036425232391
best parameters=  {'degree': 7, 'kernel': 'poly'}
[[534  13]
 [ 17  13]]
0.7047836684948202
---This information is for the  4 th label.--------
best mean cv score=  0.630630697298831
best parameters=  {'degree': 11, 'kernel': 'poly'}
[[490  44]
 [ 56  23]]
0.6043711183805054
---This information is for the  5 th label.--------
best mean cv score=  0.6947422904425824
best parameters=  {'degree': 8, 'kernel': 'poly'}
[[645   9]
 [ 20  13]]
0.6900889630247429
---This information is for the  6 th label.--------
best mean cv score=  0.5878664257410506
best parameters=  {'degree': 9, 'kernel': 'poly'}
[[612   9]
 [ 11   6]]
0.6692242114236999
---This information is for the  7 th label.--------
best mean cv score=  0.6656936034198049
best parameters=  {'degree': 14, 'kernel': 'poly'}
[[433  53]
 [ 54  37]]
0.6487699543255099
---This information is for the  8 th label.--------
best mean cv score=  0.5954265276466763
best parameters=  {'degree': 10, 'kernel': 'poly'}
[[667   8]
 [ 18   7]]
0.634074074074074
---This information is for the  9 th label.--------
best mean cv score=  0.6096192244051426
best parameters=  {'degree': 10, 'kernel': 'poly'}
[[583  21]
 [ 24  11]]
0.6397587511825922
---This information is for the  10 th label.--------
best mean cv score=  0.7740261034794257
best parameters=  {'degree': 5, 'kernel': 'poly'}
[[454  30]
 [ 36  54]]
0.7690082644628099
---This information is for the  11 th label.--------
best mean cv score=  0.6622435935009551
best parameters=  {'degree': 7, 'kernel': 'poly'}
[[618  11]
 [ 24  17]]
0.6985730350149288
```

Listing 15: GridSearchCV-SVC results

6.6 AdaBoost

AdaBoost, short for Adaptive Boosting, is a statistical classification meta-algorithm formulated by Yoav Freund and Robert Schapire, who won the 2003 Gödel Prize for their work. It can be used in conjunction with many other types of learning algorithms to improve performance. The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. In some problems it can be less susceptible to the overfitting problem than other learning algorithms. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing, the final model can be proven to converge to a strong learner. [13]

```
from sklearn.ensemble import AdaBoostClassifier
print("AdaBoostClassifier:")
for i in np.arange(0,12,1):
    param_grid={'n_estimators':[8000,1000,500,700],'
    algorithm':['SAMME.R','SAMME'],'learning_rate'
    :[1]}
    grid=GridSearchCV(AdaBoostClassifier(),param_grid=
    param_grid,cv=5,scoring='balanced_accuracy',
    return_train_score=False)
    grid.fit(X_training_data[i].to_numpy(),
    y_training_data[i])
    print("---This information is for the ",i,"th
    label.--------")
    print("best mean cv score= ",grid.best_score_)
    print("best parameters= ",grid.best_params_)
    y_pred_AdaBoost=grid.best_estimator_.predict(
    X_test[i])
    print(confusion_matrix(y_test[i],np.transpose(np.
    matrix(y_pred_AdaBoost))))
    print(grid.score(X_test[i],y_test[i]))
```

Listing 16: GridSearchCV-AdaBoost [5]

```
AdaBoostClassifier:
---This information is for the  0 th label.--------
best mean cv score=  0.7157613477540118
best parameters=  {'algorithm': 'SAMME.R', 'learning_rate': 1,
      'n_estimators': 8000}
[[674  12]
 [ 16  15]]
0.7331891281858365
---This information is for the  1 th label.--------
best mean cv score=  0.7341166085078775
best parameters=  {'algorithm': 'SAMME.R', 'learning_rate': 1,
      'n_estimators': 500}
[[640   6]
 [  6  17]]
0.8649212545430072
---This information is for the  2 th label.--------
best mean cv score=  0.7308967630172937
best parameters=  {'algorithm': 'SAMME.R', 'learning_rate': 1,
      'n_estimators': 8000}
[[535  36]
 [ 35  42]]
0.7412036299952236
---This information is for the  3 th label.--------
best mean cv score=  0.6154114251647196
best parameters=  {'algorithm': 'SAMME.R', 'learning_rate': 1,
      'n_estimators': 8000}
[[534  13]
 [ 16  14]]
0.7214503351614869
---This information is for the  4 th label.--------
best mean cv score=  0.6233451764286004
best parameters=  {'algorithm': 'SAMME.R', 'learning_rate': 1,
      'n_estimators': 1000}
[[511  23]
 [ 57  22]]
0.6177049258047693
---This information is for the  5 th label.--------
best mean cv score=  0.6699627297727602
best parameters=  {'algorithm': 'SAMME.R', 'learning_rate': 1,
      'n_estimators': 8000}
[[643  11]
 [ 21  12]]
0.6734083958854601
---This information is for the  6 th label.--------
best mean cv score=  0.5620798156865388
best parameters=  {'algorithm': 'SAMME', 'learning_rate': 1, '
     n_estimators': 8000}
```

```
[[617   4]
 [ 17   0]]
0.4967793880837359
---This information is for the  7 th label.--------
best mean cv score=  0.6537217100575029
best parameters=  {'algorithm': 'SAMME.R', 'learning_rate': 1, 'n_estimators': 8000}
[[432  54]
 [ 54  37]]
0.6477411477411477
---This information is for the  8 th label.--------
best mean cv score=  0.5969505584559676
best parameters=  {'algorithm': 'SAMME.R', 'learning_rate': 1, 'n_estimators': 8000}
[[658  17]
 [ 18   7]]
0.6274074074074074
---This information is for the  9 th label.--------
best mean cv score=  0.5858266150853018
best parameters=  {'algorithm': 'SAMME.R', 'learning_rate': 1, 'n_estimators': 700}
[[600   4]
 [ 29   6]]
0.58240302743614
---This information is for the  10 th label.--------
best mean cv score=  0.7804093635981373
best parameters=  {'algorithm': 'SAMME.R', 'learning_rate': 1, 'n_estimators': 1000}
[[458  26]
 [ 36  54]]
0.7731404958677686
---This information is for the  11 th label.--------
best mean cv score=  0.624099446415163
best parameters=  {'algorithm': 'SAMME.R', 'learning_rate': 1, 'n_estimators': 1000}
[[617  12]
 [ 32   9]]
0.6002171468455543
```

Listing 17: GridSearchCV-AdaBoost results

6.7 Bagging

Bootstrap aggregating, also called bagging (from bootstrap aggregating), is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid overfitting [9] . Although it is usually applied to Decision Tree methods, it can be used with any type of method. Bagging is a special case of the model averaging approach. [14]

```
from sklearn.ensemble import BaggingClassifier
print("BaggingClassifier:")
for i in np.arange(0,12,1):
    param_grid={'n_estimators':[10,100,500],'max_samples':[1,0.5],'max_features':[1,0.8,0.6]}
    grid=GridSearchCV(BaggingClassifier(),param_grid=param_grid,cv=5,scoring='balanced_accuracy',return_train_score=False)
    grid.fit(X_training_data[i].to_numpy(),y_training_data[i])
    print("---This information is for the ",i,"th label.--------")
    print("best mean cv score= ",grid.best_score_)
    print("best parameters= ",grid.best_params_)
    y_pred_bag=grid.best_estimator_.predict(X_test[i])
    print(confusion_matrix(y_test[i],np.transpose(np.matrix(y_pred_bag))))
    print(grid.score(X_test[i],y_test[i]))
```

Listing 18: GridSearchCV-Bagging [5]

```
AdaBoostClassifier:
---This information is for the  0 th label.--------
best mean cv score=  0.7269508786019065
best parameters=  {'max_features': 0.6, 'max_samples': 0.5, 'n_estimators': 500}
[[683   3]
 [ 15  16]]
0.7558779272077494
---This information is for the  1 th label.--------
best mean cv score=  0.7326826795066174
best parameters=  {'max_features': 0.8, 'max_samples': 0.5, 'n_estimators': 100}
[[640   6]
 [  8  15]]
0.821442993673442
---This information is for the  2 th label.--------
best mean cv score=  0.6652733608745625
best parameters=  {'max_features': 0.6, 'max_samples': 0.5, 'n_estimators': 100}
[[560  11]
 [ 48  29]]
0.6786794641435623
---This information is for the  3 th label.--------
best mean cv score=  0.559236688270835
best parameters=  {'max_features': 0.8, 'max_samples': 0.5, 'n_estimators': 500}
[[545   2]
 [ 24   6]]
0.5981718464351006
---This information is for the  4 th label.--------
best mean cv score=  0.6169766928272998
best parameters=  {'max_features': 0.6, 'max_samples': 0.5, 'n_estimators': 100}
[[524  10]
 [ 57  22]]
0.629877210448964
---This information is for the  5 th label.--------
best mean cv score=  0.6717171804892169
best parameters=  {'max_features': 0.8, 'max_samples': 0.5, 'n_estimators': 100}
[[653   1]
 [ 20  13]]
0.6962051709758132
---This information is for the  6 th label.--------
best mean cv score=  0.5288716095018979
best parameters=  {'max_features': 0.8, 'max_samples': 0.5, 'n_estimators': 10}
[[621   0]
 [ 17   0]]
0.5
---This information is for the  7 th label.--------
best mean cv score=  0.5930241135390106
best parameters=  {'max_features': 0.8, 'max_samples': 0.5, 'n_estimators': 100}
[[478   8]
 [ 62  29]]
0.6511102066657621
---This information is for the  8 th label.--------
best mean cv score=  0.529686773652454
best parameters=  {'max_features': 0.8, 'max_samples': 0.5, 'n_estimators': 10}
[[674   1]
 [ 24   1]]
0.5192592592592592
---This information is for the  9 th label.--------
best mean cv score=  0.5476517181011051
best parameters=  {'max_features': 0.8, 'max_samples': 0.5, 'n_estimators': 500}
[[603   1]
 [ 29   6]]
0.5848864711447493
---This information is for the  10 th label.--------
best mean cv score=  0.7392036815039424
best parameters=  {'max_features': 0.8, 'max_samples': 0.5, 'n_estimators': 500}
[[466  18]
 [ 42  48]]
```

```
0.7480716253443527
---This information is for the  11 th label.--------
best mean cv score=  0.5421860682323457
best parameters=  {'max_features': 0.8, 'max_samples': 0.5, '
    n_estimators': 100}
[[628   1]
 [ 35   6]]
0.5723758191476986
```

Listing 19: GridSearchCV-Bagging results

6.8 MLP

A multilayer perceptron (MLP) is a class of feedforward artificial neural network (ANN). The term MLP is used ambiguously, sometimes loosely to mean any feedforward ANN, sometimes strictly to refer to networks composed of multiple layers of perceptrons (with threshold activation); Multilayer perceptrons are sometimes colloquially referred to as "vanilla" neural networks, especially when they have a single hidden layer.[15]

An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training.[16][17] Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.[18]

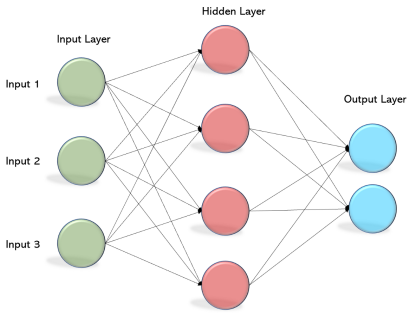

Figure 6: MLP neural network graph

```
from sklearn.neural_network import MLPClassifier
print("MLPClassifier:")
for i in np.arange(0,12,1):
    param_grid={'activation':['identity', 'logistic',
    'tanh', 'relu'],'hidden_layer_sizes':[100,500]}
    grid=GridSearchCV(MLPClassifier(),param_grid=
    param_grid,cv=5,scoring='balanced_accuracy',
    return_train_score=False)
    grid.fit(X_training_data[i].to_numpy(),
    y_training_data[i])
    print("---This information is for the ",i,"th
    label.--------")
    print("best mean cv score= ",grid.best_score_)
    print("best parameters= ",grid.best_params_)
    y_pred_bag=grid.best_estimator_.predict(X_test[i])
    print(confusion_matrix(y_test[i],np.transpose(np.
    matrix(y_pred_bag))))
    print(grid.score(X_test[i],y_test[i]))
```

Listing 20: GridSearchCV-MLP [5]

```
MLPClassifier:
---This information is for the  0 th label.--------
best mean cv score=  0.7190012640264282
best parameters=  {'activation': 'logistic', '
    hidden_layer_sizes': 500}
[[684   2]
 [ 15  16]]
0.7566067901815103
---This information is for the  1 th label.--------
best mean cv score=  0.7713046063735265
best parameters=  {'activation': 'relu', 'hidden_layer_sizes':
    100}
[[639   7]
 [  7  16]]
0.8424081303001749
---This information is for the  2 th label.--------
best mean cv score=  0.7485781227406563
best parameters=  {'activation': 'tanh', 'hidden_layer_sizes':
    500}
[[544  27]
 [ 36  41]]
0.7425910341847295
---This information is for the  3 th label.--------
best mean cv score=  0.6675002192944698
best parameters=  {'activation': 'relu', 'hidden_layer_sizes':
    500}
[[540   7]
 [ 18  12]]
0.6936014625228519
---This information is for the  4 th label.--------
best mean cv score=  0.6513103184676161
best parameters=  {'activation': 'tanh', 'hidden_layer_sizes':
    500}
[[511  23]
 [ 57  22]]
0.6177049258047693
---This information is for the  5 th label.--------
best mean cv score=  0.6968680706325083
best parameters=  {'activation': 'relu', 'hidden_layer_sizes':
    100}
[[646   8]
 [ 20  13]]
0.6908534890186266
---This information is for the  6 th label.--------
best mean cv score=  0.5938001655740702
best parameters=  {'activation': 'tanh', 'hidden_layer_sizes':
    500}
[[617   4]
 [ 13   4]]
0.6144264469072653
---This information is for the  7 th label.--------
best mean cv score=  0.6845563149901454
best parameters=  {'activation': 'tanh', 'hidden_layer_sizes':
    100}
[[461  25]
 [ 55  36]]
0.6720820331931443
---This information is for the  8 th label.--------
best mean cv score=  0.6304795924052781
best parameters=  {'activation': 'tanh', 'hidden_layer_sizes':
    100}
[[663  12]
 [ 19   6]]
0.6111111111111112
---This information is for the  9 th label.--------
best mean cv score=  0.6378664632826452
best parameters=  {'activation': 'tanh', 'hidden_layer_sizes':
    500}
[[594  10]
 [ 27   8]]
0.6060075685903501
---This information is for the  10 th label.--------
best mean cv score=  0.798803902979208
best parameters=  {'activation': 'tanh', 'hidden_layer_sizes':
    100}
[[452  32]
 [ 34  56]]
0.7780532598714417
---This information is for the  11 th label.--------
```

```
best mean cv score=  0.6682045890004455
best parameters=  {'activation': 'tanh', 'hidden_layer_sizes':
      500}
[[622   7]
 [ 23  18]]
0.7139478072046221
```

Listing 21: GridSearchCV-MLP results

7 EVALUATION

In this section, I want to compare the accuracy of different algorithms for each label that we have obtained based on balanced accuracy. First of all, let's have a summary of what was done. I prepared the data to be processed by different algorithms. Then I ran different algorithms on the data. The results included cv score, confusion matrix, and the absolute accuracy of the algorithm on the test set. In this section, I will select the best predictor algorithm for each dimension of the label vector based on the results.

For the head dimension of the label vector, I prepared a figure containing three diagrams [such as Figure 7]. The first graph is the accuracy obtained from the test set based on balanced accuracy. The second diagram contains the cv score output of each algorithm. [10] The third diagram compares the previous two, giving us great information about overfitting or underfitting. In general, the most important parameter to choose the best algorithm will be balanced accuracy, then the analysis of the third diagram, cv score, and confusion matrix can help us in selecting the best choice.

7.1 1st dimension of the labels vector

Figure 7 shows us that almost all algorithms have been able to be accurate at practically the same level. However, the highest accuracy was on the test set for Random forest and multi layer perceptron algorithms. Fortunately, all test set results are better than cv scores, which means we do not have overfitting. The AdaBoost algorithm also had the lowest accuracy on the test set.

Which is better between Random forest and multi layer perceptron algorithms? Random forest seems to be a good option due to the use of fewer features and a higher cv score. But the execution runtime of the multilayer perceptron algorithm is significantly shorter. Assuming we do not have a computation time problem, I select the Random forest algorithm with the hyperparameters mentioned in Listing 11.

7.2 2nd dimension of the labels vector

Figure 8 also shows us that almost all algorithms have the same level of accuracy. However, the highest accuracy on the test set was for Random forest and AdaBoost algorithms. I'm glad that in the second dimension, as in the first dimension, all the results related to the test set are better than the cv scores, and this, fortunately, means that we are not overfitting. Also, the SVC algorithm had the lowest accuracy on the test set.

Which is better between Random forest and AdaBoost algorithms? The Random forest seems to be a good option due to the use of fewer features and a higher cv score. Also, the execution time of the AdaBoost algorithm is not much different from the Random forest. I chose the Random forest algorithm with the hyperparameters mentioned in Listing 11 as the best algorithm for estimating the second dimension of the label vector.

Finally, I should note that the selected algorithm has achieved 86.49212545430072 % balanced accuracy.

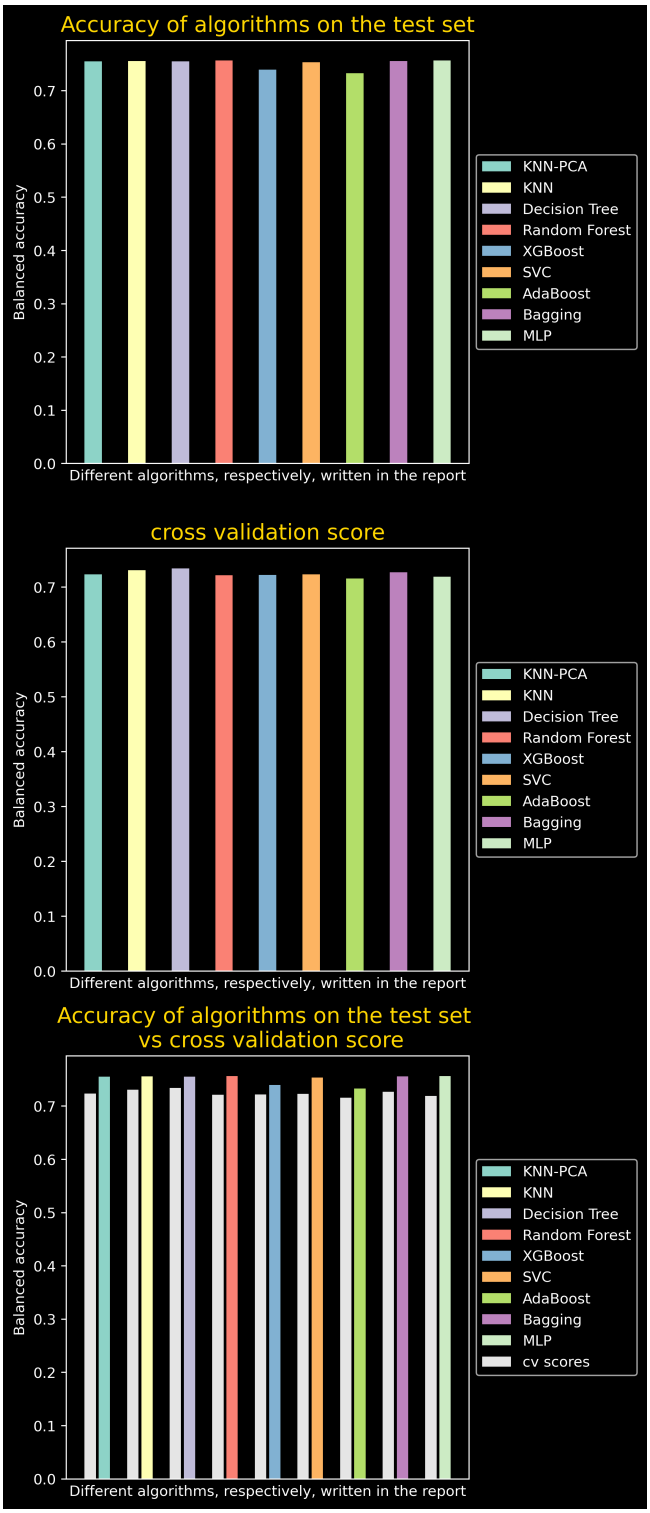

Figure 7: Diagrams of the 1st dimension of the labels vectors[4]

7.3 3rd dimension of the labels vector

Figure 9 shows that the accuracy of algorithms has more variance than the first and second dimensions. However, the highest accuracy has been on the test set for KNN algorithm. Except for the MLP algorithm, the test set results are better than the cv scores in the third dimension, which means we are not likely to see overfitting in this dimension. The Bagging algorithm also had the lowest accuracy on the test set.

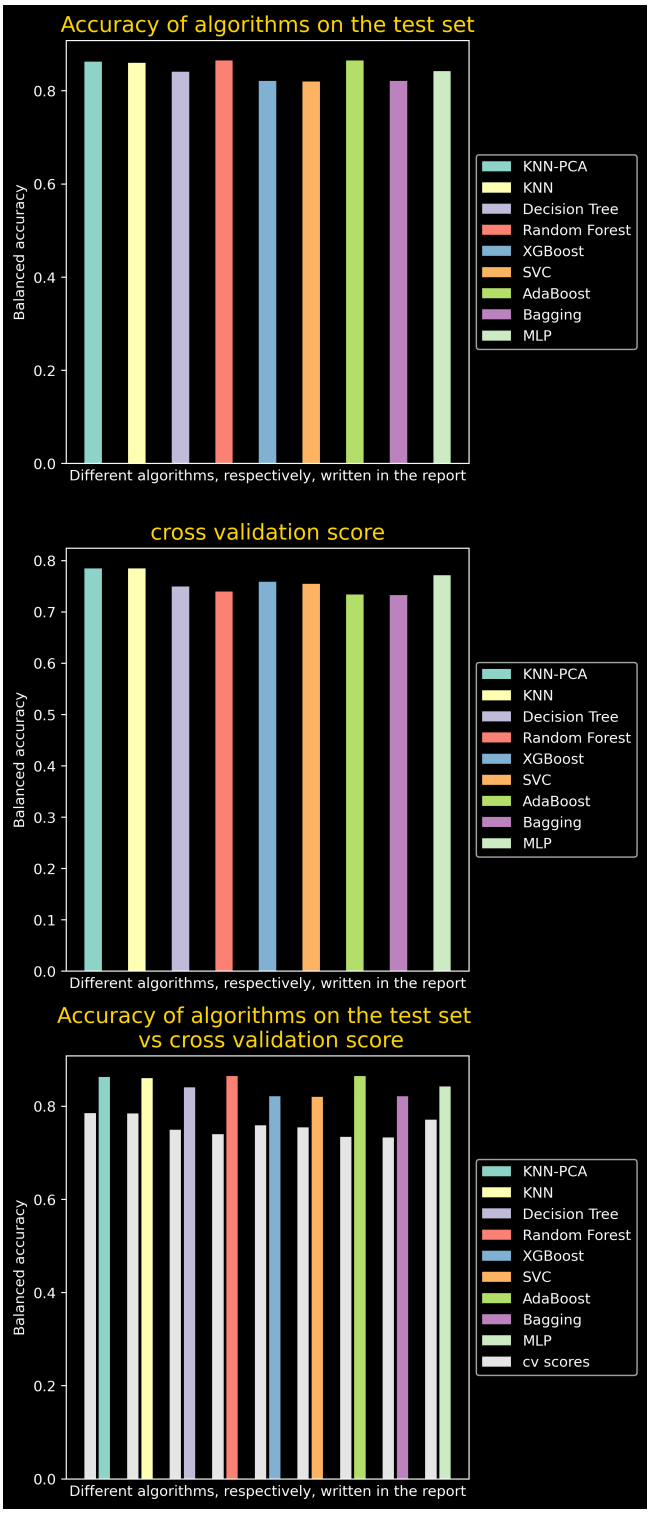

Figure 8: Diagrams of the 2nd dimension of the labels vectors[4]

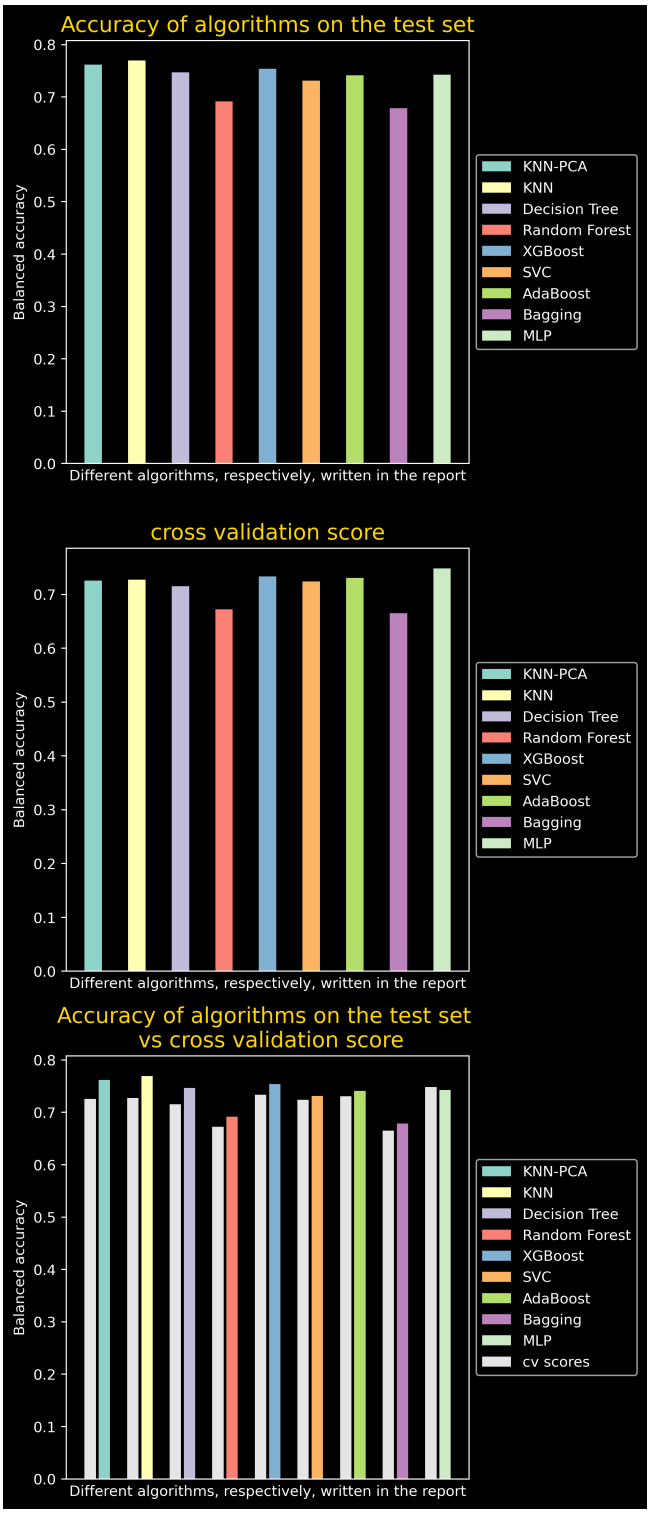

Figure 9: Diagrams of the 3rd dimension of the labels vectors[4]

7.4 4th dimension of the labels vector

Interestingly, KNN has performed better on this data dimensionFigure 11 shows us that the variance of algorithms accuracy in the fourth than KNN-PCA. So I was curious to see how KNN performed on alldimension is much more higher than in the previous dimension. However, dimensions of my data? I found that KNN did better in 6 dimensionsthe highest accuracy was on the test set for KNN-PCA algorithm. In the and worse in 6 dimensions. But adding up their error difference, I foundfourth dimension, in all algorithms, the test set results are better than the that overall KNN performance was slightly weaker than KNN-PCA. [Seecv scores, which means we are not trapped in overfitting. The Bagging Figure 10] algorithm also had the lowest accuracy on the test set.

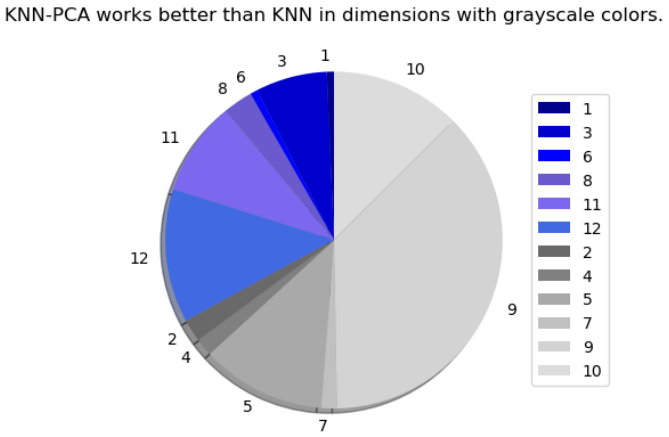

Figure 10: KNN vs. KNN-PCA performance in this project[4]

The interesting thing is that in the previous dimension where KNN performed well, Bagging still had the weakest performance.

7.5 5th dimension of the labels vector

Figure 12 shows almost a slight variance between the accuracy of the different algorithms. However, the most careful focus was on the test set for the XGBoost algorithm. Unfortunately, in the fifth dimension, we see some algorithms with a weak performance in test set score vs. cv scores. The KNN algorithm also had the lowest accuracy on the test set.

Of course, the highest cv score is related to the MLP algorithm, which, as you can see in Figure 12, had a poorer performance in the test set than the cv score. I think it smells overfitting. As a result, we leave this choice aside. In the dimension of the XGBoost algorithm achieved 64.41829042810412% balanced accuracy.

7.6 6th dimension ofthe labels vector

According to Figure 5, apart from the Bagging algorithm's highest test score in the sixth dimension, only the two Bagging and AdaBoost algorithms have obtained a higher test score than their cv score. This is while the average balanced accuracy of all algorithms is 67.357829%. And the AdaBoost algorithm has a slightly lower test score than the average. But the Bagging algorithm has a test score that is more than 2.2% higher than the average. For these reasons, it is enough to consider Bagging as the best predictor for sixth dimension labels.

7.7 7th dimension ofthe labels vector

Figure 14 shows the relative success of simpler algorithms compared to time-consuming algorithms. The seventh dimension is where the Decision Tree algorithm shows its power. Looking at Figure 22, you can see how the Decision Tree algorithm differs from other algorithms much better. The average balanced accuracy of all executed algorithms is more than 10% less than the accuracy obtained by the Decision Tree algorithm.

It is worth noting that even Decision Tree-based algorithms, such as Random forest and Bagging, have exactly the same confusion matrices. They could not even predict a True label number correctly.

7.8 8th dimension ofthe labels vector

Figure 15 shows the highest accuracy on the test set for XGBoost algorithms. In the eighth dimension, unfortunately, 33% of the algorithms in the test set were less accurate than the cv score. The

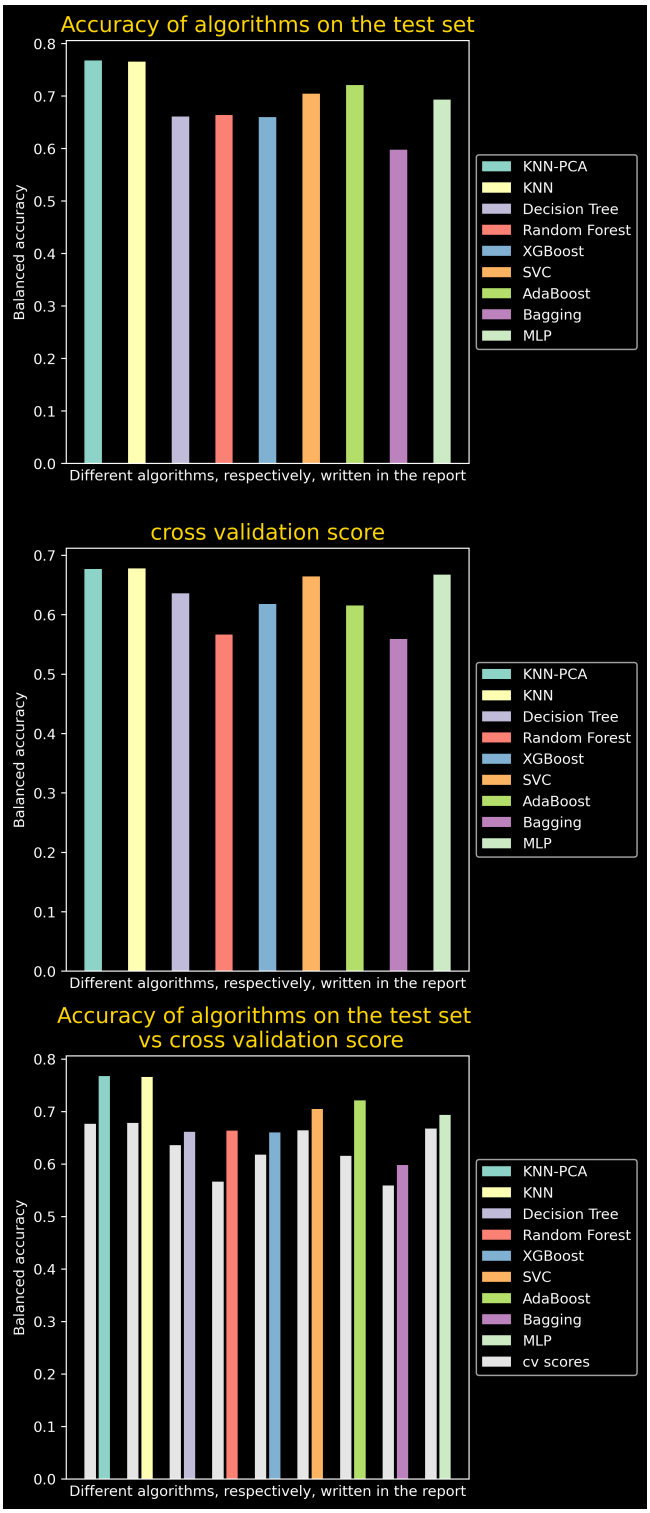

Figure 11: Diagrams of the 4th dimension of the labels vectors[4]

Bagging algorithm also had the lowest accuracy on the test set. The KNN-PCA and KNN algorithms performed well but were not the best. We have already seen the inverse relationship of the Bagging algorithm with these two algorithms.

It is not harmful to know that there is about a 99% correlation between the accuracy of KNN-PCA and KNN algorithms. Interestingly, the lowest correlation between the total accuracy of the implemented algorithms is

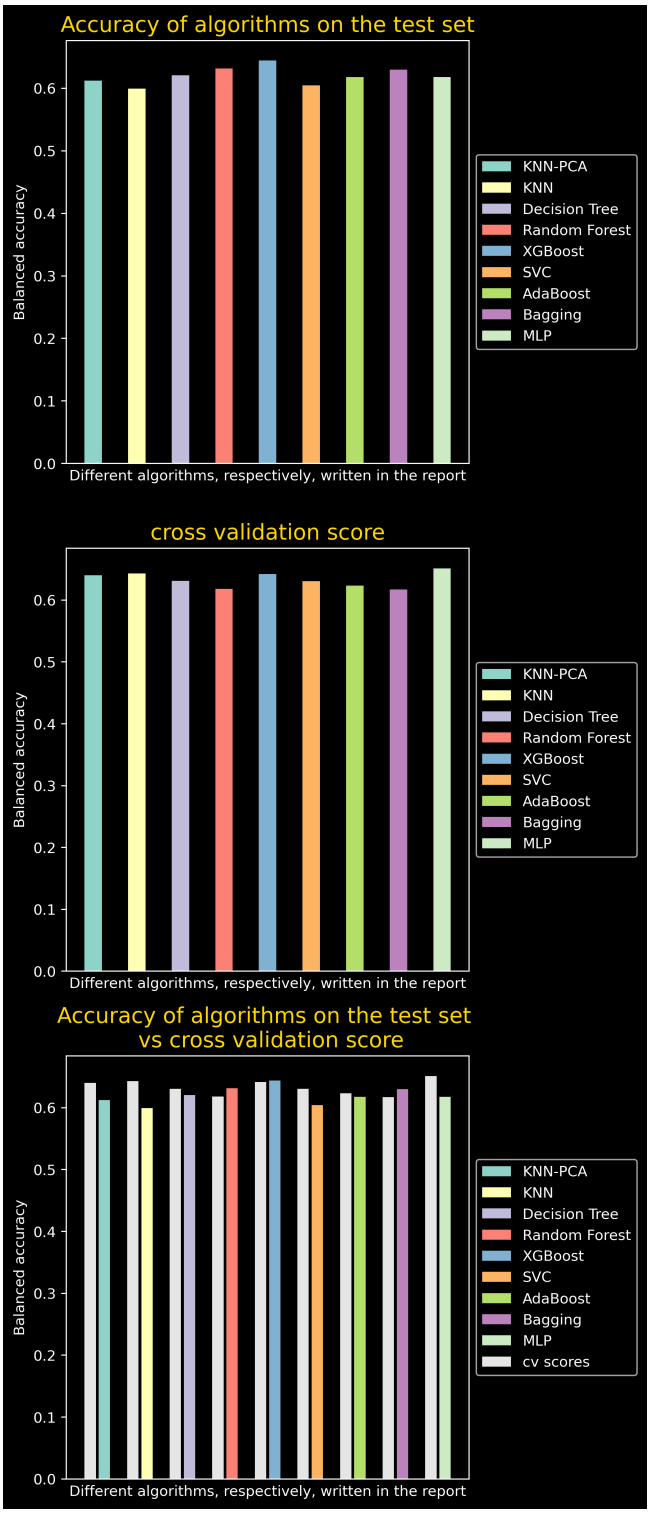

Figure 12: Diagrams of the 5th dimension of the labels vectors[4]

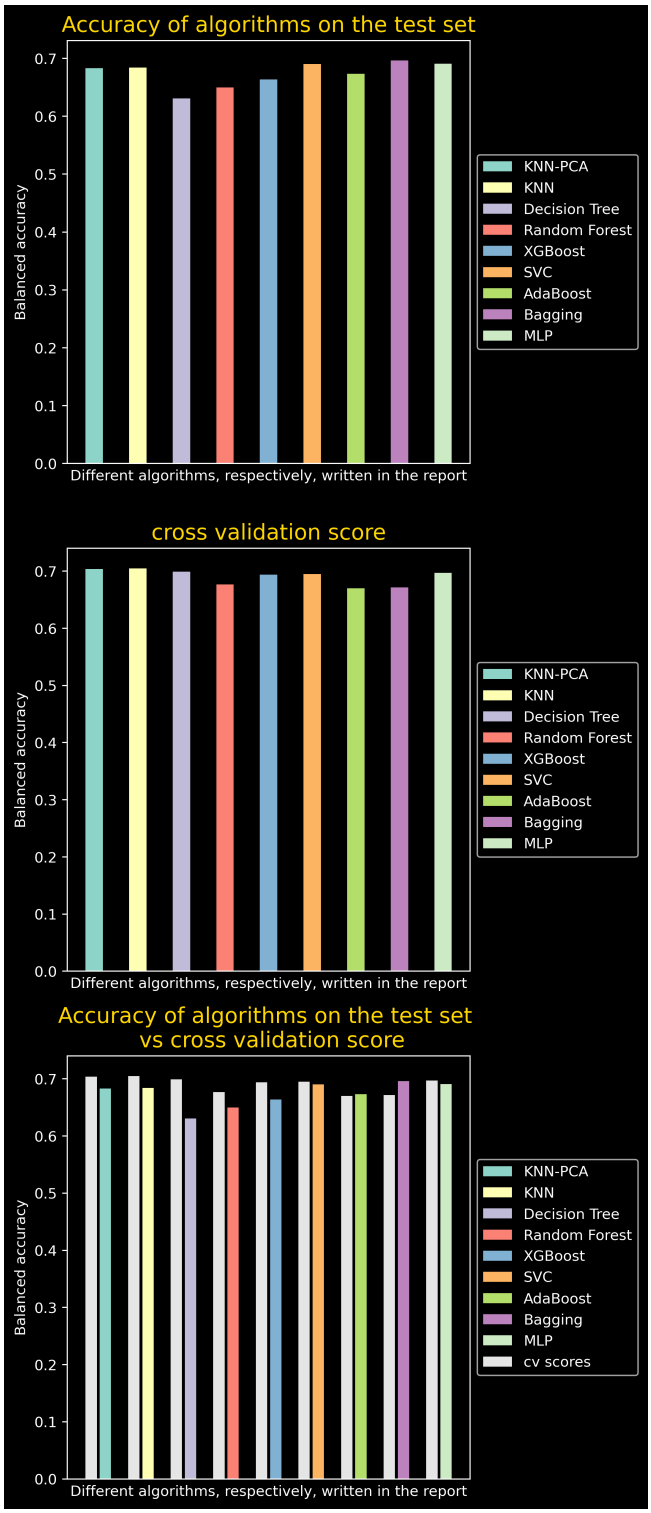

Figure 13: Diagrams ofthe 6th dimension ofthe labels vectors[4]

related to KNN-PCA and Bagging. For this reason, KNN-PCA and KNN have performed well or well, almost wherever Bagging has performed poorly. Interestingly, after KNN-PCA and KNN, the highest correlation with a high rate of 96% is related to Random forest and Bagging algorithms. So with a bit of carelessness, I can claim everything that I said about Bagging, about the Random forest algorithm.[Refer to Figure 16]

7.9 9th dimension ofthe labels vector

The results of the algorithms on the label vector's ninth dimension, which represents the toxicity of SR-ATAD5, can be seen in Figure 17. As shown in Figure 22, this dimension has one of the highest variances of algorithm accuracy compared to other dimensions. In this dimension, KNN-PCA has obtained the highest score based on the balanced accuracy measurement system. As expected, Bagging was the least accurate.

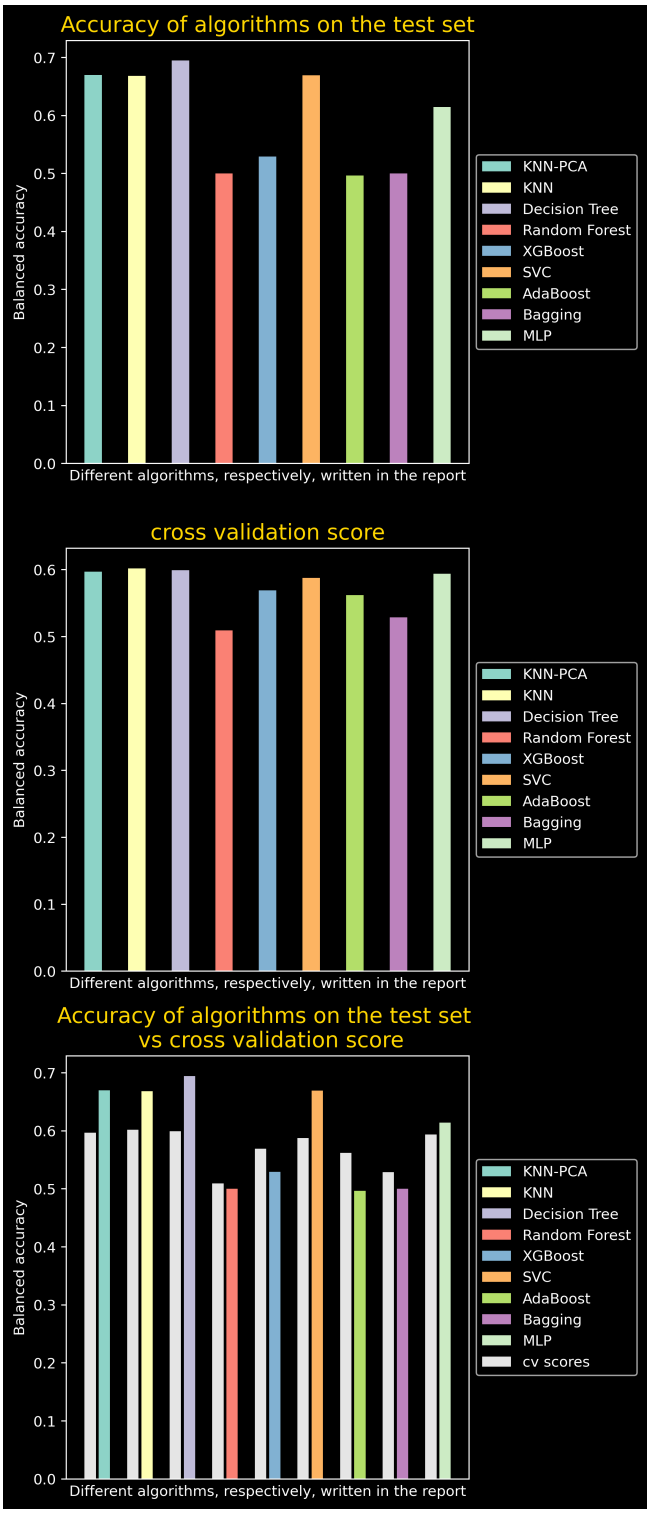

Figure 14: Diagrams ofthe 7th dimension ofthe labels vectors[4]

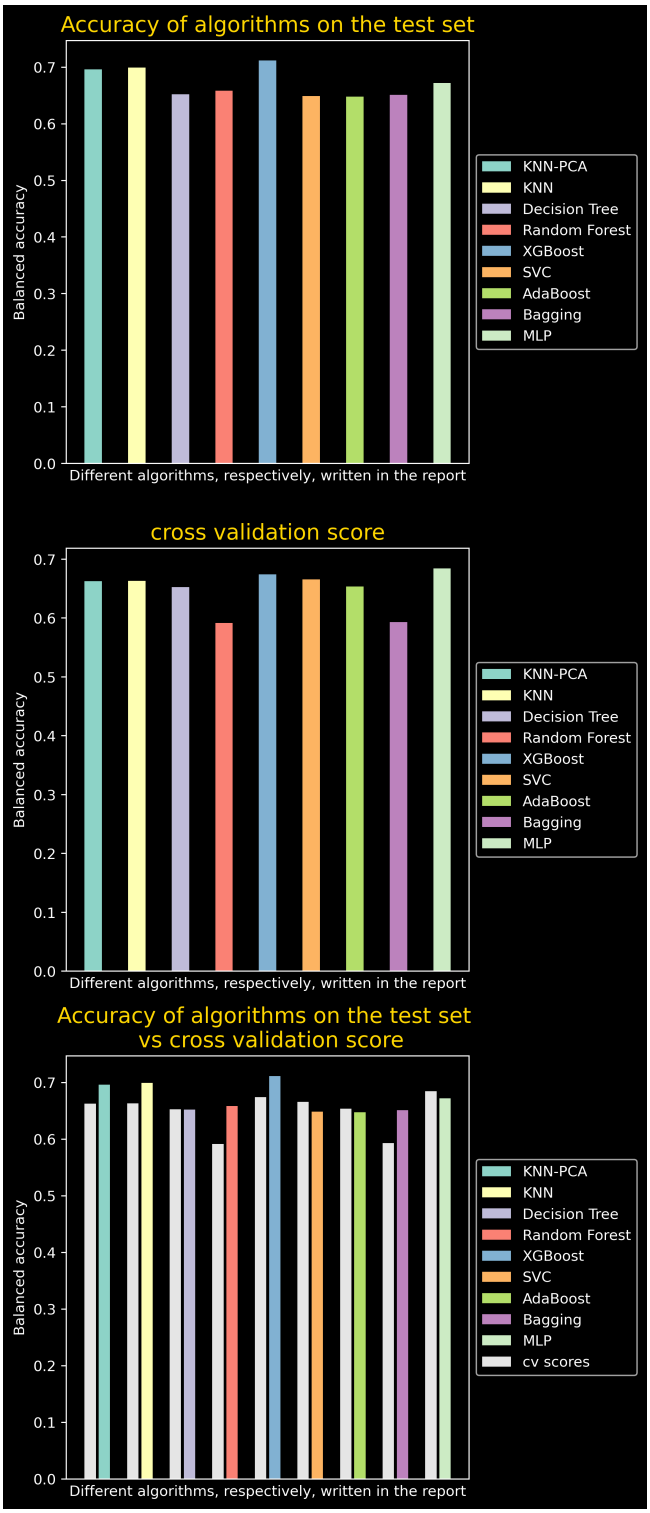

Figure 15: Diagrams ofthe 8th dimension ofthe labels vectors[4]

But given the high correlation between the labels predicted by KNN-PCA and KNN, the question is, did KNN also perform well? As shown in Figure 10, the ninth dimension shows the greatest advantage of KNN-PCA over KNN. Figure 17 also shows a small amount of overfitting in the KNN. Of course, the hyperparameters are selected in exactly the same in these algorithms, but KNN-PCA correctly predicts about 25% more True labels than KNN.

7.10 10th dimension ofthe labels vector

The results of performing algorithms on the tenth dimension are very similar to the ninth dimension. In this dimension, the label vectors representing SR-HSE toxicity can be seen in Figure 18. As shown in Figure 22, this dimension, like the ninth dimension, has a significant variance in the accuracy of the algorithms. KNN-PCA has obtained the highest score based on the balanced accuracy measurement system in the

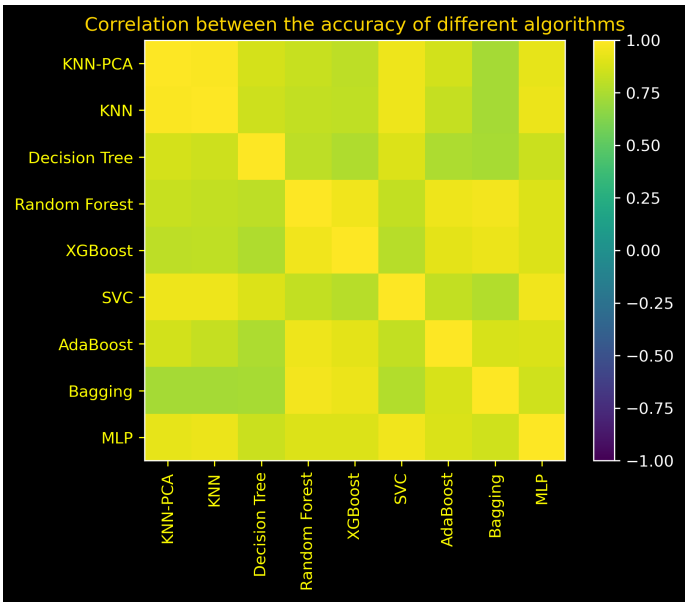

Figure 16: Correlation between the accuracy of different algorithms[4]

tenth dimension. As expected, the result of Bagging was not desirable, but the least accurate is related to the Random forest algorithm.

In Figure 23, you can see the accuracy range of the algorithm in these 12 dimensions (average, minimum and maximum). As the figure shows, the Random forest algorithm has a relatively high variance in balanced accuracy on these 12 dimensions. This algorithm had the highest score in predicting NR-AR toxicity, while it had the lowest accuracy here.

Also, despite the fact that the KNN-PCA hyperparameter [11] has been selected in this dimension (as well as in many other dimensions) in such a way that it seems to lead to overfitting, our test results show that we are not trapped by overfitting at all.

7.11 11th dimension ofthe labels vector

In the eleventh dimension of the feature vector, we refer to the toxicity of SR-MMP [Refer to Figure 19]. The accuracy of the algorithms has been almost the same. The highest accuracy on the test set was related to the XGBoost algorithm. Unfortunately, the test set result was slightly lower than the cv score. But this difference is less than what we worry about overfitting. While the highest cv score was for MLP, the final accuracy on the test set was significantly different from the cv score, which makes me motivated to choose XGBoost.

The lowest accuracy of both test set results and the cv score result are related to the bagging algorithm. After Bagging, the weakest performance was associated with the Random forest algorithm. Of course, I emphasize again that the maximum difference in data accuracy was 4%, which means that all algorithms have performed approximately the same. That's why Random forest may be a good option because it works with fewer features, which makes it easier for us to collect data. But in terms of accuracy, XGBoost was a little better than the rest.

7.12 12th dimension ofthe labels vector

Figures 20 and 22 show us that not all algorithms in the twelfth dimension are alike. However, mathematically the highest balanced accuracy belongs to KNN. I want to break with tradition and choose MLP as the best model here. The MLP algorithm has about 0.2% less balanced accuracy than KNN, which is not a significant number at all. Because the test score of the MLP algorithm is much better than the cv score, I can firmly claim that in MLP, we taught the model even better than KNN.

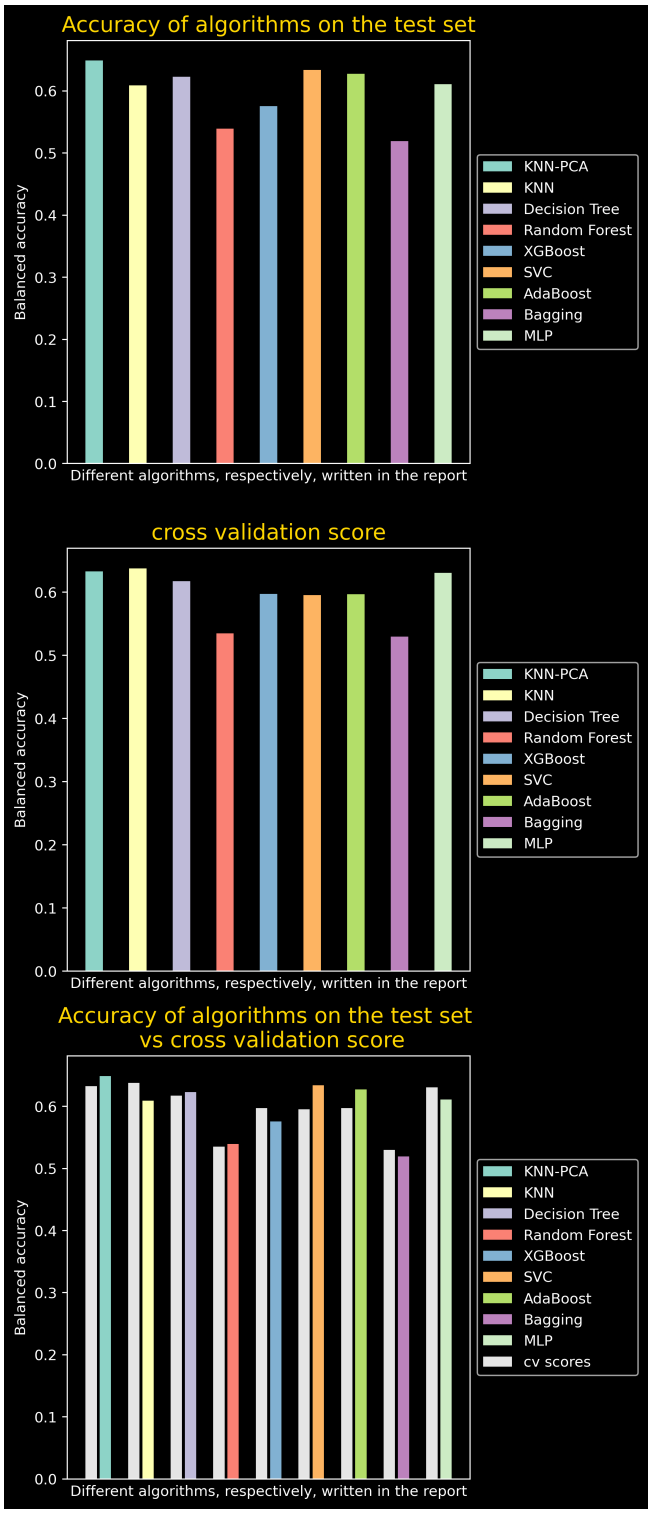

Figure 17: Diagrams ofthe 9th dimension ofthe labels vectors[4]

In SR-p53-related labels, such as the SR-MMP dimension, the weakest performance is in Bagging and Random forest, respectively. But in this dimension, unlike the eleventh dimension, the difference between these two algorithms and the best algorithms is much more than what can be ignored.

In general, I think it is easier for us to choose when the algorithms differ from the results of different algorithms. Because when the

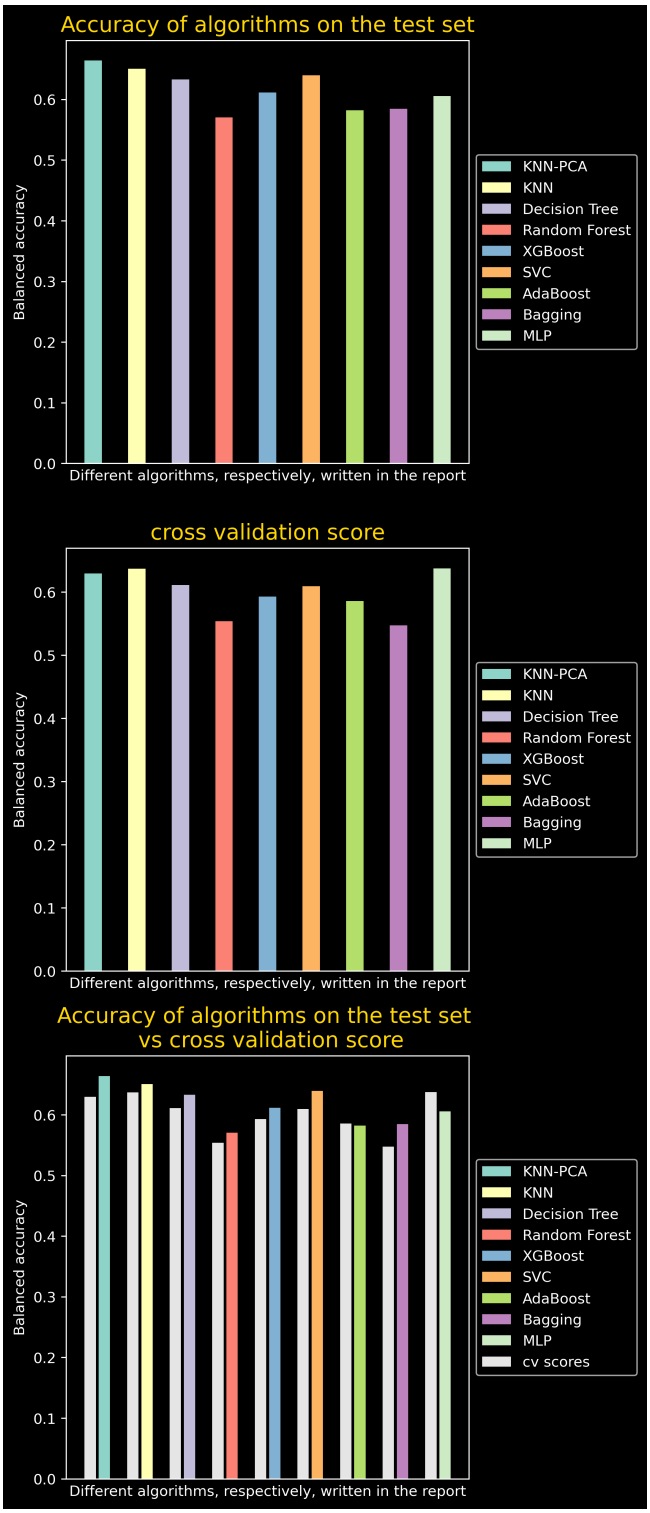

Figure 18: Diagrams ofthe 10th dimension ofthe labels vectors[4]

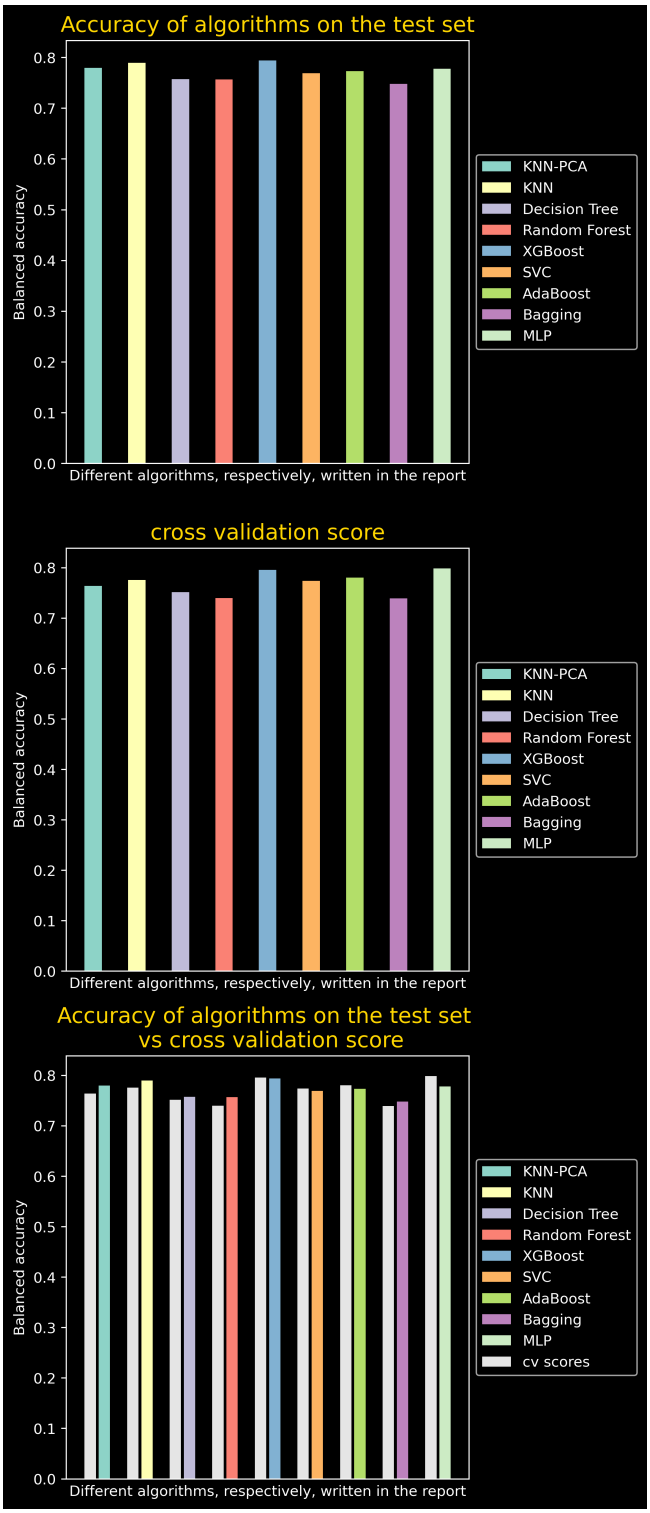

Figure 19: Diagrams ofthe 11th dimension ofthe labels vectors[4]

difference between the results of different algorithms is slight, we have to consider various parameters. For example, one algorithm maybe 0.002 percent weaker than another if it has hundreds of times simpler models or takes much less time to run this algorithm. Therefore, when one algorithm performs better than other algorithms with a big difference, we are less involved in these analyzes. As a result, the probability of our analytical error also decreases significantly.

8 CONCLUSION

8.1 Initial conclusion

First, I want to briefly rewrite the selected algorithms in each dimension and the accuracy of the information associated with them. Then I will provide an overview of which algorithms we used the most and which algorithms we did not use. Also, as a multi-label project, I must specify the final accuracy.

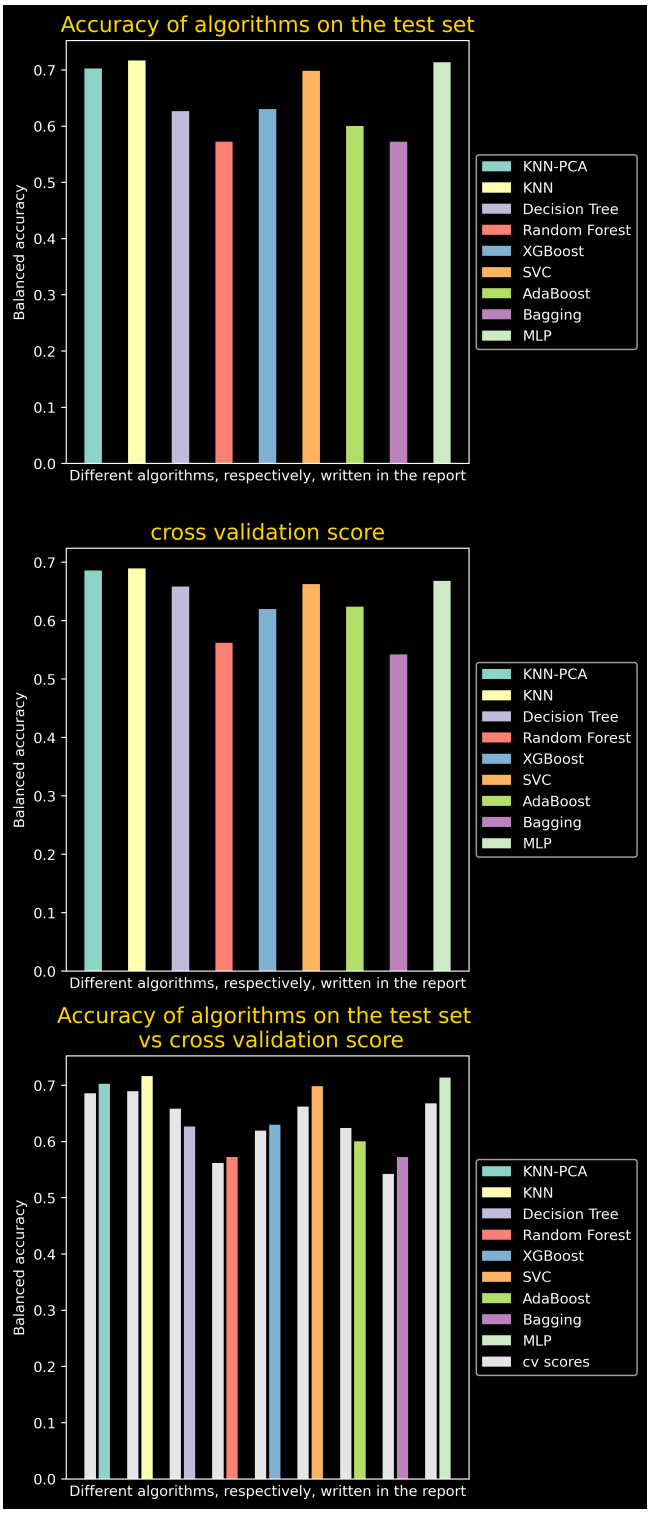

Figure 20: Diagrams ofthe 12th dimension ofthe labels vectors[4]

1. **NR-AR**
    - Selected algorithm: Random forest
    - Hyperparameters: `criterion: gini, max_depth: 15, max_features: auto, n_estimators: 700`
    - Balanced accuracy: 75.66067901815103%
2. **NR-AR-LBD**
    - Selected algorithm: Random forest
    - Hyperparameters: `criterion: entropy, max_depth: 10, max_features: sqrt, n_estimators: 500`
    - Balanced accuracy: 86.49212545430072%
3. **NR-AhR**
    - Selected algorithm: KNN
    - Hyperparameters: `metric: manhattan, n_neighbors: 1`
    - Balanced accuracy: 76.94407169013123%
4. **NR-Aromatase**
    - Selected algorithm: KNN-PCA
    - Hyperparameters: `metric: cosine, n_neighbors: 1`
    - Balanced accuracy: 76.7794028031688%
5. **NR-ER**
    - Selected algorithm: XGBoost
    - Hyperparameters: `eval_metric: mlogloss, max_depth: 3, n_estimators: 2000`
    - Balanced accuracy: 64.41829042810411%
6. **NR-ER-LBD**
    - Selected algorithm: Bagging
    - Hyperparameters: `max_features: 0.8, max_samples: 0.5, n_estimators: 100`
    - Balanced accuracy: 69.62051709758133%
7. **NR-PPAR-gamma**
    - Selected algorithm: Decision Tree
    - Hyperparameters: `criterion: gini, max_depth: 28`
    - Balanced accuracy: 69.46102112342521%
8. **SR-ARE**
    - Selected algorithm: XGBoost
    - Hyperparameters: `eval_metric: mlogloss, max_depth: 5,n_estimators: 2000`
    - Balanced accuracy: 71.15723782390448%
9. **SR-ATAD5**
    - Selected algorithm: KNN-PCA
    - Hyperparameters: `metric: cosine, n_neighbors: 1`
    - Balanced accuracy: 64.88888888888889%
10. **SR-HSE**
    - Selected algorithm: KNN-PCA
    - Hyperparameters: `metric: manhattan, n_neighbors: 1`
    - Balanced accuracy:66.41911069063386%
11. **SR-MMP**
    - Selected algorithm:XGBoost

- Hyperparameters: `eval_metric: mlogloss, max_depth: 3,n_estimators: 700`
- Balanced accuracy: 79.393939393939%

12. **SR-p53**

- Selected algorithm: MLP [12]
- Hyperparameters: `activation: tanh , hidden_layer_sizes: 500`
- Balanced accuracy: 71.39478072046221%

As you can see in the histogram of Figure 21, the frequency of selection of KNN-PCA and XGBoost algorithms was three times. The Random Forest algorithm was then selected twice to predict the two dimensions of the label vector. The KNN, Bagging, Decision Tree, and MLP algorithms were in third place, each selected once. Also, the SVC and AdaBoost algorithms were never selected.

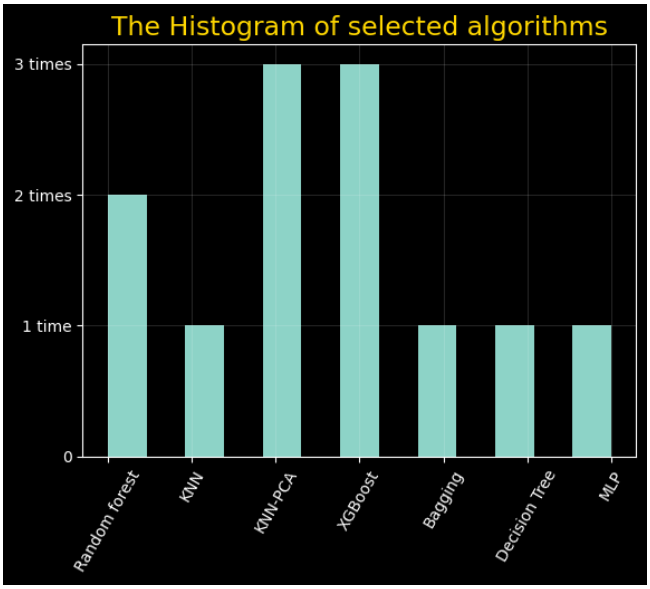

Figure 21: The Histogram of selected algorithms[4]

I also included Figure 22 so that you can see the performance of each algorithm in different dimensions of the labels vector in one frame.

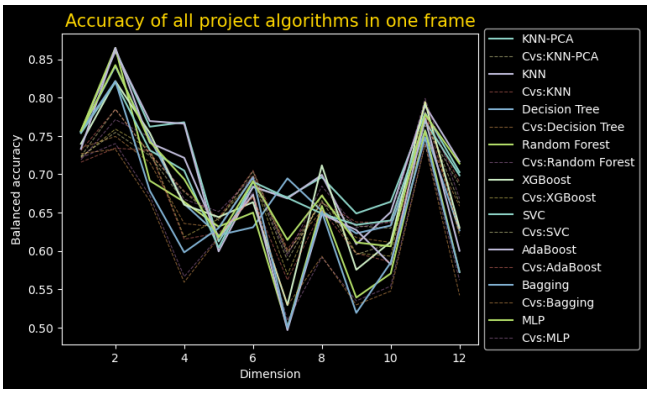

Figure 22: Accuracy of all project algorithms in one frame[4]

Since Figure 22 might seem a bit crowded, I prepared Figure 23 to evaluate the algorithms better so that you can see the accuracy range of each algorithm and the average accuracy of each algorithm separately.

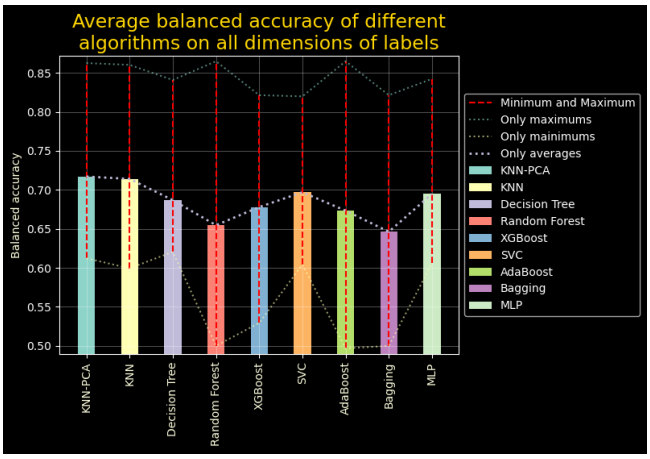

Figure 23: Average balanced accuracy of different algorithms on all dimensions of labels[4]

I want to say something informal in this part. Of course, the whole project had to be multitasking, and all 12 dimensions had to be applied to the data simultaneously without assuming independence. But according to the professor's orders, I decided to implement the project as above. Of course, we examined the correlation between labels, and the assumption of label independence is not very wrong.

However, if we were to study 12 dimensions simultaneously, we would probably have much better options of balanced accuracy to choose as a meter. Although, I want to introduce you to a very informal criterion based on balanced accuracy for evaluating the whole project, which was just made up by my own mind. After thinking about it, I saw many weaknesses, but I preferred to write this criterion in my report.

My goal from the beginning was to predict a 12-dimensional vector. I have said before that I use balanced accuracy to evaluate my algorithms. But I would like to have a number as the final accuracy of these algorithms on the whole project. First, this number must be good. This means that it cannot be interpreted as a liar. Secondly, it should be interpretable.

Since we had a confusion matrix for each selected algorithm, I guessed that summing these confusion matrices might give us good information. It should not be left unmentioned that average balancing is not a suitable parameter because the distribution of zeros and ones and the number of test set members in each label are different. I finally arrive at an interpretable matrix by summing the confusion matrices. This matrix explicitly answers the following questions.

1. How many True labels did I guess correctly?
2. How many False labels did I guess correctly?
3. How many True labels did I guess wrong?
4. How many False labels did I guess wrong?

Finally, I can calculate the balanced accuracy of the matrix and reach an excellent meter to measure the overall accuracy of the project.

$$\text{Confusion matrix} = \begin{bmatrix} 6880 & 175 \\ 293 & 279 \end{bmatrix}$$

$$\text{Balanced accuracy} = 73.14785674991204\%$$

In other words, I correctly predicted 97.51948972360028% of zeros in the total whole of the project. I also predicted 48.776223776223776% of the ones correctly. Clearly, our main challenge throughout the project has been to anticipate the ones.

There may be many reasons why this criterion is not good. For example, to say that we need a measure that examines every twelve-dimensional vector as a package of 12 binaries. Suppose we have a test set of 500 members. Our labels will be in the form of the following matrix.

$$\text{Labels}: \begin{bmatrix} 0 & \dots & 1 \\ 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix}_{500\times 12}, \text{Prediction}: \begin{bmatrix} 0 & \dots & 1 \\ 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{bmatrix}_{500\times 12}$$

My goal is to know how similar these two matrices are. As a result, we can define a confusion matrix for this matrix and know how accurate the prediction matrix was. Summing the confusion matrices I mentioned earlier is exactly the same as the confusion matrix obtained by comparing the above two matrices. I got the basic idea from studying a response in stackexchange.[19]

8.2 Final project conclusion

While editing the project report, I noticed that I could not select the final model using the test set. In competitions, to be fair, the test dataset is not revealed until you submit the final trained model.

Therefore, I decided to select the best models based on the cv score at the end of the report and compare the newly chosen models with the previous ones selected based on the test set results.

1. **NR-AR**
   - Selected algorithm: Decision Tree
   - CV score$_{\text{based on the balanced accuracy}}$: 73.43343187519518%
   - Balanced accuracy on the test set: 75.51490642339886%
2. **NR-AR-LBD**
   - Selected algorithm: KNN-PCA
   - CV score$_{\text{based on the balanced accuracy}}$: 78.5002324569213%
   - Balanced accuracy on the test set: 86.25992731188586%
3. **NR-AhR**
   - Selected algorithm: MLP
   - CV score$_{\text{based on the balanced accuracy}}$: 74.85781227406562%
   - Balanced accuracy on the test set: 74.25910341847295%
4. **NR-Aromatase**
   - Selected algorithm: KNN
   - CV score$_{\text{based on the balanced accuracy}}$: 67.81417541315211%
   - Balanced accuracy on the test set: 76.59658744667885%
5. **NR-ER**
   - Selected algorithm: MLP
   - CV score$_{\text{based on the balanced accuracy}}$: 65.13103184676162%
   - Balanced accuracy on the test set: 61.77049258047693%
6. **NR-ER-LBD**
   - Selected algorithm: KNN
   - CV score$_{\text{based on the balanced accuracy}}$: 70.49780678881433%
   - Balanced accuracy on the test set: 68.39727550736725%
7. **NR-PPAR-gamma**
   - Selected algorithm: KNN
   - CV score$_{\text{based on the balanced accuracy}}$: 60.20852568911459%
   - Balanced accuracy on the test set: 66.84190584446338%
8. **SR-ARE**
   - Selected algorithm: MLP
   - CV score$_{\text{based on the balanced accuracy}}$: 68.45563149901453%
   - Balanced accuracy on the test set: 67.20820331931444%
9. **SR-ATAD5**
   - Selected algorithm: KNN
   - CV score$_{\text{based on the balanced accuracy}}$: 63.77311937912506%
   - Balanced accuracy on the test set: 60.88888888888888%
10. **SR-HSE**
    - Selected algorithm: MLP
    - CV score$_{\text{based on the balanced accuracy}}$: 63.78664632826452%
    - Balanced accuracy on the test set: 60.60075685903501%
11. **SR-MMP**
    - Selected algorithm: MLP
    - CV score$_{\text{based on the balanced accuracy}}$: 79.8803902979208%
    - Balanced accuracy on the test set: 77.80532598714417%
12. **SR-p53**
    - Selected algorithm: KNN
    - CV score$_{\text{based on the balanced accuracy}}$: 68.9557702999988%
    - Balanced accuracy on the test set: 71.66039784404204%

With these interpretations, you can see the frequency of selected algorithms based on the Cv score in Figure 24.

I want to take this opportunity to try the metrics I suggested to evaluate the whole project here. We expect the accuracy to be a bit lower here, but we should not take it too seriously from the previous confusion matrix. I calculated the cumulative confusion matrix for the selected models based on the cross-validation score.

$$\text{Confusion matrix} = \begin{bmatrix} 6704 & 210 \\ 322 & 257 \end{bmatrix}$$

$$\text{Balanced accuracy} = 70.67477916449965\%$$

In other words, I correctly predicted 96.96268440844663% of zeros in the total whole of the project. I also predicted 44.38687392055268% of the ones correctly.

As you can see, we did not use the test set to re-select the algorithm. So it was apparent that our criterion called cumulative confusion matrix

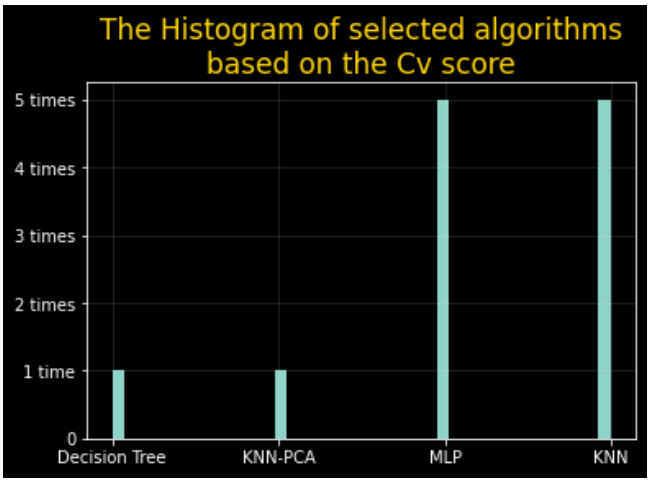

Figure 24: The Histogram of selected algorithms based on the Cv score[4]

should give a lower score than the whole project, which it did. This shows that this indicator does not seem very illogical.

The entire project run time (excluding research and programming) for me lasted about four consecutive days on a 48-core computer (each core was 2.4 GHz).

NOTES

[1] `initial_dataset.dropna().describe().iloc[0,0]` command in the Python environment after reading the label file can return us the number.

[2] **Principal component analysis (PCA)** is a technique for reducing the dimensionality of such datasets, increasing interpretability but at the same time minimizing information loss. It does so by creating new uncorrelated variables that successively maximize variance.[20]

[3] According to the group's agreement, the random number 1234 has been selected for train_test_split

[4] **GridSearchCV** implements a **fit** and a **score** method. It also implements `score_samples`, `predict`, `predict_proba`, `decision_function`, `transform` and `inverse_transform` if they are implemented in the estimator used.[5]

[5] TP:True Positive, P:Positive, TN:True Negative, N:Negative

[6] FP:False Positive ,FN: False Negative

[7] for example, when the training set is the entire Web and distances are based on links

[8] Independent and identically distributed

[9] In statistics, overfitting is "the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably". An overfitted model is a statistical model that contains more parameters than can be justified by the data. The essence of overfitting is to have unknowingly extracted some of the residual variation (i.e., the noise) as if that variation represented underlying model structure.

[10] As you know, the best hyper-parameters are selected based on this parameter.

[11] In machine learning, a hyperparameter is a parameter whose value is used to control the learning process. By contrast, the values of other parameters (typically node weights) are derived via training.

[12] Multi Layer Perceptron

ACKNOWLEDGEMENTS

I would like to thank my applied machine learning professor (Dr. Hajiabolhasan) and all teacher assistants who gave me a golden opportunity to work on this project. I wish these loved ones good health and success.

REFERENCES

[1] Shai Shalev-Shwartz and Shai Ben-David. Understanding Machine Learning - From Theory to Algorithms. Cambridge University Press, 2014.

[2] Li Di and Edward H. Kerns. Chapter 17 - toxicity. In Li Di and Edward H. Kerns, editors, Drug-Like Properties (Second Edition), pages 251–262. Academic Press, Boston, second edition edition, 2016.

[3] hulab.rxnfinder.org. Convert smiles to image, 2019.

[4] J. D. Hunter. Matplotlib: A 2d graphics environment. Computing in Science & Engineering, 9(3):90–95, 2007.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825–2830, 2011.

[6] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. Nature, 585(7825):357–362, September 2020.

[7] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, Proceedings of the 9th Python in Science Conference, pages 56 – 61, 2010.

[8] Jr Mohammed J. Zaki, Wagner Meira. Data Mining and Machine Learning-Fundamental Concepts and Algorithms. Cambridge University Press, 2020.

[9] Joydwip Mohajon. Confusion matrix for your multi-class machine learning model, 2020.

[10] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus F. M. Ng, B. Liu, Philip S. Yu, Zhi-Hua Zhou, Michael S. Steinbach, David J. Hand, and Dan Steinberg. Top 10 algorithms in data mining. Knowledge and Information Systems, 14:1–37, 2007.

[11] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[12] Corinna Cortes and Vladimir Naumovich Vapnik. Support-vector networks. Machine Learning, 20:273–297, 2004.

[13] Wikipedia contributors. Adaboost Wikipedia, the free encyclopedia, 2022. [Online; accessed 13-Feb-2022].

[14] Wikipedia contributors. Bootstrap Wikipedia, the free encyclopedia, 2021. [Online; accessed 13-Feb-2022].

[15] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The Elements of Statistical Learning. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[16] Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. American Journal of Psychology, 76:705, 1963.

[17] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In David E. Rumelhart and James L. Mcclelland, editors, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations, pages 318–362. MIT Press, Cambridge, MA, 1986.

[18] G. Cybenko. Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals, and Systems (MCSS), 2(4):303–314, December 1989.

[19] rapaio. How to get an aggregate confusion matrix from n different classifications, 2014. [Online; accessed 10-Jun-2014].

[20] Jorge Cadima Ian T Jolliffe. Principal component analysis: a review and recent developments. Philosophical transactions. Series A, Mathematical, physical, and engineering sciences, (7825):374, April 2016.