Applied Machine Learning Project

Pages 1–10
2022

Hiv 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, Logistic regression, Boosting, Bagging ,AdaBoost, Ensemble, MLP, Random forest, Decision tree, Python, Balanced accuracy, Confusion matrix

ABSTRACT

The following project is a machine learning project written in Python. This project's dataset describes the formula of chemical molecules of $41,127$ different drugs. I should mention that the chemical properties of each of them were available to us with a 200-dimensional normalized vector. Using understanding machine learning from theory to algorithms [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

I'm so sorrowful that I have to start my report with shocking statistics. My motivation for choosing this project was that this disease has led to the death of many victims. This project is definitely too small to have the slightest impact. But I think the essential message of this project is the statistics ahead, which motivated me to work on this project. I hope the definitive cure will be discovered as soon as possible.

- 1.5 million people became newly infected with HIV in 2020.
- $680,000$ people died from AIDS-related illnesses in 2020.
- 79.3 million people have become infected with HIV since the start of the epidemic. [1]
- 36.3 million people have died from AIDS-related illnesses since the start of the epidemic.[2]
- Studies from England and South Africa have found that the risk of dying from COVID-19 among people with HIV was double that of the general population. [2]

1 INTRODUCTION

My data contains two files named `hiv_global_cdf_rdkit` and `hiv`. 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]

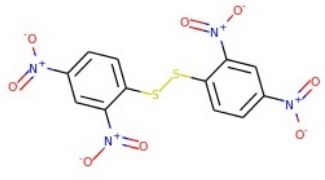

Figure 1: The chemical formula of the eighth data

The label file is relatively straightforward and unambiguous, but the feature file contains a significant amount of empty information. Of the 41,127 data, 39,629 are wholly unambiguous. I should note that data labels are highly imbalanced. I will discuss each of these points in detail in the following parts. But I must now point out that this data has two main challenges.

1. The relatively high number of data will lead to a large volume of calculations
2. Data imbalance may be an area of expertise that I am certainly not specialized in.

2 DESCRIPTION OF RAW DATA

2.1 Labels distribution

As I mentioned before, we are dealing with highly imbalanced data. For further explanation, I must say that out of 41,127 data; only 1443 data are labeled 1. This means that only 3.508% of the data are labeled 1, and 96.491% are labeled zero. For a better understanding, Figure 2 shows this imbalanced distribution.

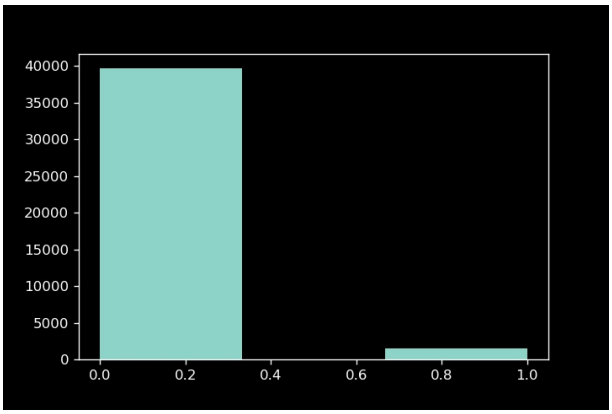

Figure 2: Labels distribution

2.2 Missing data

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.[4]

But a significant challenge is missing data. Is this data damaged as same as the same original distribution? No! Out of 1498 missing data, 172 of data have label 1, which is approximately 11.5% of the total missing data. In other words, if we want to discard all the damaged data, the balance of the remaining information is not the same as reality. If the damaged data is discarded, only 3.207% of the remaining data will be labeled as 1. This means that more than 11.9% of the tagged one information is somehow damaged. To solve this problem, I separated the data with label one then replaced the missing feature with the average of that column. [3] you can see more detail in Listing 1

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
#Importing essential packages
initial_Features=pd.read_csv('hiv_global_cdf_rdkit.csv')
initial_dataset=pd.read_csv('hiv.csv')
#Reading data
ones_array=[]
only_ones_inf=initial_Features
for i in range(0,41127):
    if initial_dataset['HIV_active'][i]>0:
        ones_array.append(i)
    else:
        only_ones_inf=only_ones_inf.drop(i)
#Separate data labeled 1
only_ones_inf_filled_mean=only_ones_inf.fillna(only_ones_inf.mean())
#Fill in the blank data with the average of the same property (only for data that has a label.)
only_zeros_inf=initial_Features
for i in ones_array:
    only_zeros_inf=only_zeros_inf.drop(i)
#Separate data labeled 0
secondary_Features=pd.concat([only_ones_inf_filled_mean,only_zeros_inf]).sort_index().dropna()
#Define new data (without deleting data that is labeled one)
secondary_dataset=pd.read_csv('hiv.csv')
tmp=pd.concat([only_ones_inf_filled_mean,only_zeros_inf]).sort_index().isna().any(axis=1)
np.size(tmp)
for i in np.arange(0,np.size(tmp)):
    if tmp[i]==True:
        secondary_dataset=secondary_dataset.drop(i,axis=0)
#Syncing secondary labels according to secondary data
```

Listing 1: Deal with missing data [5] [6]

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 192 algebraic dimensions.[Refer to Listing 2] 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.

```
secondary_Features=secondary_Features.loc[:,secondary_Features.apply(pd.Series.nunique) != 1]
```

Listing 2: Remove ineffective features

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 [4] 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. [5] I also intended to allocate 10% of the data volume to the test set.

```
from sklearn.model_selection import train_test_split
X_training_data, X_test, y_training_data, y_test=train_test_split(secondary_Features,secondary_dataset, stratify=secondary_dataset['HIV_active'],test_size=0.10,random_state=1234)
```

Listing 3: using train test split [4]

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 [7]

I also looked at the data and found that PCA on my data, with 77 principal components, can maintain 95.0935% of the data variance. So, I made this dimensional reduction for some algorithms, as shown in Listing 4

```
from sklearn.decomposition import PCA
pca = PCA(n_components=77)
principalComponents = pca.fit_transform(
    X_training_data)
X_training_data_PCA = pd.DataFrame(data =
    principalComponents)
```

Listing 4: PCA [4]

5 ALGORITHMS

In this section, I will run various algorithms on the data. I will give a specific explanation about each algorithm. Then you will see the result of running the algorithm. I have to say that I ran the algorithms using gridsearchcv and chose the best hyperparameters. [6]

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.

$$\textbf{Balanced accuracy} = \frac{1}{2}\left(\frac{\mathrm{TP}}{\mathrm{P}} + \frac{\mathrm{TN}}{\mathrm{N}}\right) \quad [7]$$

Also, in the algorithms results, you will encounter a matrix called confusion matrix, which is defined as in Figure 3.

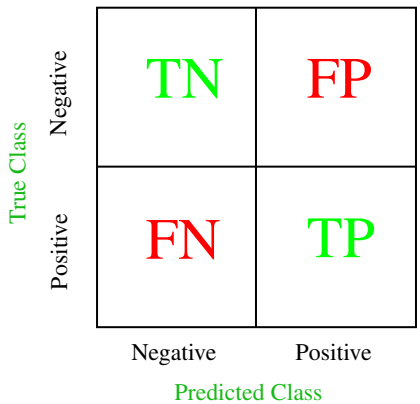

Figure 3: confusion matrix[8][8]

5.1 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 [9]. [1] You can see the execution of KNN on 77-dimensional data (after PCA application) in Listing 5.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix,
    plot_confusion_matrix
knn = KNeighborsClassifier()
param_grid={'n_neighbors':np.arange(1,100,2),'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.to_numpy().astype('float'
    ),np.transpose(y_training_data_PCA.to_numpy().
    astype('int'))[0])
print("best mean cv score= ",grid.best_score_)
print("best parameters= ",grid.best_params_)
y_pred_knn=grid.best_estimator_.predict(X_test_pca)
print(confusion_matrix(y_test['HIV_active'],np.
    transpose(np.matrix(y_pred_knn))))
X_test_pca=pd.DataFrame(data=pca.transform(X_test))
grid.score(X_test_pca,y_test['HIV_active'])
```

Listing 5: GridSearchCV-KNN-PCA [4]

```
best mean cv score=  0.7096760964856432
best parameters=  {'metric': 'manhattan', 'n_neighbors': 1}
[[3757   80]
 [  74   70]]
0.7326307445052559
```

Listing 6: GridSearchCV-KNN-PCA results

You can see the results of the algorithm accuracy with different meters and hyperparameters in Figure 4.

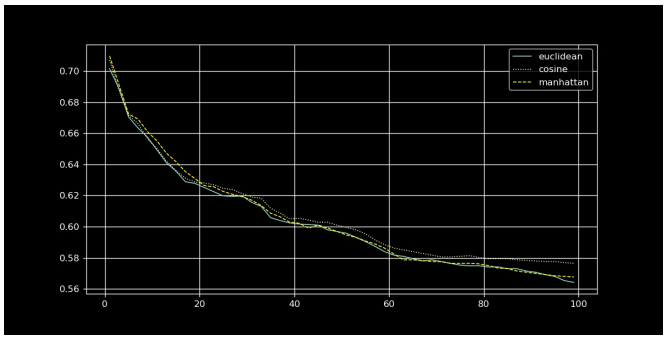

Figure 4: KNN accuracy

I implemented the KNN algorithm on the data without reducing the PCA dimensions, but it did not obtain the desired result, so I omitted the implementation of the algorithm on the data without PCA here. Also, in this project, I felt that PCA on some algorithms like KNN improves the result. While in the case of some algorithms, such as the decision tree algorithm, it can destroy the results. The reason for this is almost apparent given the nature of these two algorithms. Anyway, this is just a conjecture, and it does not mean that this is correct.

5.2 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. [9]

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. For example, I invite you to compare the results of Listings (7,8) and (9,10).

```
from sklearn.tree import DecisionTreeClassifier,
    export_graphviz
DecisionTree = DecisionTreeClassifier()
param_grid={'criterion':['gini','entropy'],'max_depth'
    :[18,20]}
grid=GridSearchCV(DecisionTree,param_grid=param_grid,
    cv=10,scoring='balanced_accuracy',
    return_train_score=False)
grid.fit(X_training_data_PCA.to_numpy().astype('float'
    ),np.transpose(y_training_data_PCA.to_numpy().
    astype('int'))[0])
print("best mean cv score= ",grid.best_score_)
print("best parameters= ",grid.best_params_)
y_pred_dt=grid.best_estimator_.predict(X_test_pca)
print(confusion_matrix(y_test['HIV_active'],np.
    transpose(np.matrix(y_pred_dt))))
print(grid.score(X_test_pca,y_test['HIV_active']))
```

Listing 7: GridSearchCV-Decision Tree-PCA [4]

```
best mean cv score=  0.6131812593469491
best parameters=  {'criterion': 'entropy', 'max_depth': 20}
[[3725  112]
 [ 100   44]]
0.6381830423073582
```

Listing 8: GridSearchCV-Decision Tree-PCA results

```
from sklearn.tree import DecisionTreeClassifier,
    export_graphviz
DecisionTree = DecisionTreeClassifier()
param_grid={'criterion':['gini','entropy'],'max_depth'
    :[19,18]}
grid=GridSearchCV(DecisionTree,param_grid=param_grid,
    cv=5,scoring='balanced_accuracy',
    return_train_score=False)
grid.fit(X_training_data.to_numpy().astype('float'),
    y_training_data['HIV_active'])
print("best mean cv score= ",grid.best_score_)
print("best parameters= ",grid.best_params_)
y_pred_dt2=grid.best_estimator_.predict(X_test)
print(confusion_matrix(y_test['HIV_active'],np.
    transpose(np.matrix(y_pred_dt2))))
print(grid.score(X_test,y_test['HIV_active']))
```

Listing 9: GridSearchCV-Decision Tree [4]

```
best mean cv score=  0.6578310426299441
best parameters=  {'criterion': 'entropy', 'max_depth': 19}
[[3793   44]
 [  92   52]]
0.6748219094778907
```

Listing 10: GridSearchCV-Decision Tree results

5.3 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. [10] 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 11 and 12.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
RFC = RandomForestClassifier()
param_grid={'n_estimators':[200,300,1000],'criterion'
    :['gini','entropy'],'max_depth':[5,6,8,10,18,25],
    '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.to_numpy().astype('float'),
    y_training_data['HIV_active'])
print("best mean cv score= ",grid.best_score_)
print("best parameters= ",grid.best_params_)
y_pred_RFC=grid.best_estimator_.predict(X_test)
print(confusion_matrix(y_test['HIV_active'],np.
    transpose(np.matrix(y_pred_RFC))))
print(grid.score(X_test,y_test['HIV_active']))
```

Listing 11: GridSearchCV-Random Forest [4]

```
best mean cv score=  0.6348288515425576
best parameters=  {'criterion': 'gini', 'max_depth': 25, '
    max_features': 'auto', 'n_estimators': 200}
[[3828    9]
 [ 106   38]]
0.6307716532012857
```

Listing 12: GridSearchCV-Random Forest results

5.4 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. [10] In the following, I have implemented this beautiful algorithm on my data.

```
from xgboost import XGBClassifier
import warnings
warnings.simplefilter('ignore')
param_grid={'n_estimators':[700,2000],'max_depth'
    :[4,6],'eval_metric':['mlogloss']}
grid=GridSearchCV(XGBClassifier(),param_grid=
    param_grid,cv=10,scoring='balanced_accuracy',
    return_train_score=False)
```

```
grid.fit(X_training_data.to_numpy().astype('float'),(
    np.transpose((y_training_data['HIV_active']).
    to_numpy().astype('int'))))
print("best mean cv score= ",grid.best_score_)
print("best parameters= ",grid.best_params_)
y_pred_xgbc=grid.best_estimator_.predict(X_test)
print(confusion_matrix(y_test['HIV_active'],np.
    transpose(np.matrix(y_pred_xgbc))))
print(grid.score(X_test,y_test['HIV_active']))
```

Listing 13: GridSearchCV-XGBoost [4]

```
best mean cv score=  0.6785054406535646
best parameters=  {'eval_metric': 'mlogloss', 'max_depth': 4,
    'n_estimators': 2000}
[[3817   20]
 [  90   54]]
0.684893797237425
```

Listing 14: GridSearchCV-XGBoost results

5.5 C-Support Vector Classification

In machine learning, support-vector machines (SVMs, also support-vector networks[11]) 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
param_grid={'kernel':['poly'],'degree':[5,6,7,8]}
grid=GridSearchCV(SVC(),param_grid=param_grid,cv=10,
    scoring='balanced_accuracy',return_train_score=
    False)
grid.fit(X_training_data.to_numpy().astype('float'),(
    np.transpose((y_training_data['HIV_active']).
    to_numpy().astype('int'))))
print("best mean cv score= ",grid.best_score_)
print("best parameters= ",grid.best_params_)
y_pred_svm=grid.best_estimator_.predict(X_test)
print(confusion_matrix(y_test['HIV_active'],np.
    transpose(np.matrix(y_pred_svm))))
print(grid.score(X_test,y_test['HIV_active']))
```

Listing 15: GridSearchCV-SVC [4]

```
best mean cv score=  0.6878128245770883
best parameters=  {'degree': 7, 'kernel': 'poly'}
[[3769   68]
 [  79   65]]
0.7168333550516897
```

Listing 16: GridSearchCV-SVC results

5.6 Logistic Regression

Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables. [12]

In logistic regression we learn a family of functions h from $\mathbb{R}^d$ to the interval $[0, 1]$. However, logistic regression is used for classification tasks: We can interpret $h(x)$ as the *probability* that the label of **x** is 1. The hypothesis class associated with logistic regression is the composition of a sigmoid function $\phi_{\text{sig}} : \mathbb{R} \to [0, 1]$ over the class of linear functions L_d . In particular, the sigmoid function used in logistic regression is the *logistic function*, defined as

$$\phi_{\text{sig}}(z) = \frac{1}{1 + \exp(-z)}$$

The results of the Logistic Regression algorithm on my data were not very good. That's why I examined both the primary data and the data generated by the PCA algorithm with this algorithm. The result was better when I did not use PCA. But compared to other algorithms, it was not very desirable. I would like to draw your attention to Listings 17,18,19 and 20.

```
from sklearn.linear_model import LogisticRegression
param_grid={'penalty':['l1','l2','none','elasticnet'],
    'solver':['newton-cg','lbfgs','liblinear','sag','
    saga']}
grid=GridSearchCV(LogisticRegression(),param_grid=
    param_grid,cv=5,scoring='balanced_accuracy',
    return_train_score=False)
grid.fit(X_training_data_PCA.to_numpy().astype('float'
    ),(np.transpose((y_training_data_PCA['HIV_active'
    ]).to_numpy().astype('int'))))
print("best mean cv score= ",grid.best_score_)
print("best parameters= ",grid.best_params_)
y_pred_lr_p=grid.best_estimator_.predict(X_test_pca)
print(confusion_matrix(y_test['HIV_active'],np.
    transpose(np.matrix(y_pred_lr_p))))
print(grid.score(X_test_pca,y_test['HIV_active']))
```

Listing 17: GridSearchCV-Logistic Regression-PCA [4]

```
best mean cv score=  0.5343843746116311
best parameters=  {'penalty': 'none', 'solver': 'newton-cg'}
[[3835    2]
 [ 130   14]]
0.5483504908348537
```

Listing 18: GridSearchCV-Logistic Regression-PCA results

```
from sklearn.linear_model import LogisticRegression
param_grid={'penalty':['l1','l2','none','elasticnet'],
    'solver':['newton-cg','lbfgs','liblinear','sag','
    saga']}
grid=GridSearchCV(LogisticRegression(),param_grid=
    param_grid,cv=5,scoring='balanced_accuracy',
    return_train_score=False)
grid.fit(X_training_data.to_numpy().astype('float'),(
    np.transpose((y_training_data['HIV_active']).
    to_numpy().astype('int'))))
print("best mean cv score= ",grid.best_score_)
print("best parameters= ",grid.best_params_)
y_pred_lr=grid.best_estimator_.predict(X_test)
print(confusion_matrix(y_test['HIV_active'],np.
    transpose(np.matrix(y_pred_lr))))
print(grid.score(X_test,y_test['HIV_active']))
```

Listing 19: GridSearchCV-Logistic Regression [4]

```
best mean cv score=  0.5862637041676992
best parameters=  {'penalty': 'none', 'solver': 'lbfgs'}
[[3830    7]
 [ 115   29]]
0.5997822734775432
```

Listing 20: GridSearchCV-Logistic Regression results

5.7 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
param_grid={'n_estimators':[500,700,1000,8000],'
    algorithm':['SAMME.R'],'learning_rate':[1]}
grid=GridSearchCV(AdaBoostClassifier(),param_grid=
    param_grid,cv=5,scoring='balanced_accuracy',
    return_train_score=False)
grid.fit(X_training_data_PCA.to_numpy().astype('float'
    ),(np.transpose((y_training_data_PCA['HIV_active'
    ]).to_numpy().astype('int'))))
print("best mean cv score= ",grid.best_score_)
print("best parameters= ",grid.best_params_)
y_pred_adaboost_pca=grid.best_estimator_.predict(
    X_test_pca)
print(confusion_matrix(y_test['HIV_active'],np.
    transpose(np.matrix(y_pred_adaboost_pca))))
print(grid.score(X_test_pca,y_test['HIV_active']))
```

Listing 21: GridSearchCV-AdaBoost-PCA [4]

```
best mean cv score=  0.5711347496883823
best parameters=  {'algorithm': 'SAMME.R', 'learning_rate': 1,
     'n_estimators': 1000}
[[3811   26]
 [ 123   21]]
0.5695286030753193
```

Listing 22: GridSearchCV-AdaBoost-PCA results

```
from sklearn.ensemble import AdaBoostClassifier
param_grid={'n_estimators':[500,700,1000,8000],'
    algorithm':['SAMME', 'SAMME.R'],'learning_rate'
    :[1]}
grid=GridSearchCV(AdaBoostClassifier(),param_grid=
    param_grid,cv=5,scoring='balanced_accuracy',
    return_train_score=False)
grid.fit(X_training_data.to_numpy().astype('float'),(
    np.transpose((y_training_data['HIV_active']).
    to_numpy().astype('int'))))
print("best mean cv score= ",grid.best_score_)
print("best parameters= ",grid.best_params_)
y_pred_adaboost=grid.best_estimator_.predict(X_test)
print(confusion_matrix(y_test['HIV_active'],np.
    transpose(np.matrix(y_pred_adaboost))))
print(grid.score(X_test,y_test['HIV_active']))
```

Listing 23: GridSearchCV-AdaBoost [4]

```
best mean cv score=  0.6508824113415643
best parameters=  {'algorithm': 'SAMME.R', 'learning_rate': 1,
     'n_estimators': 8000}
[[3813   24]
 [  97   47]]
0.6600670011293546
```

Listing 24: GridSearchCV-AdaBoost results

5.8 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 [11] . 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
param_grid={'n_estimators':[10,100,500,5000],'
    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.to_numpy().astype('float'),(
    np.transpose((y_training_data['HIV_active']).
    to_numpy().astype('int'))))
print("best mean cv score= ",grid.best_score_)
print("best parameters= ",grid.best_params_)
y_pred_bag=grid.best_estimator_.predict(X_test)
print(confusion_matrix(y_test['HIV_active'],np.
    transpose(np.matrix(y_pred_bag))))
print(grid.score(X_test,y_test['HIV_active']))
```

Listing 25: GridSearchCV-Bagging [4]

```
best mean cv score=  0.638850300575269
best parameters=  {'max_features': 0.8, 'max_samples': 0.5, '
    n_estimators': 500}
[[3827   10]
 [ 101   43]]
0.648002454174268
```

Listing 26: GridSearchCV-Bagging results

5.9 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]

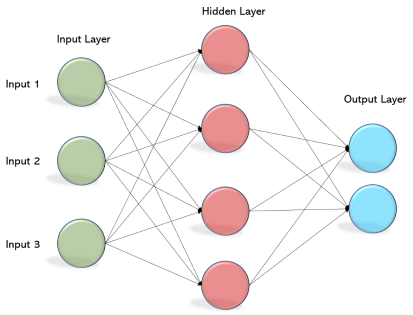

Figure 5: MLP neural network graph

```
from sklearn.neural_network import MLPClassifier
```

```
param_grid={'activation':['identity', 'tanh', 'relu'],
    'hidden_layer_sizes':[80,100,150,255],'
    learning_rate': ['constant','adaptive'],'alpha':
    [0.0001, 0.05]}
grid=GridSearchCV(MLPClassifier(),param_grid=
    param_grid,cv=5,scoring='balanced_accuracy',
    return_train_score=False)
grid.fit(X_training_data.to_numpy().astype('float'),(
    np.transpose((y_training_data['HIV_active']).
    to_numpy().astype('int'))))
print("best mean cv score= ",grid.best_score_)
print("best parameters= ",grid.best_params_)
y_pred_MLP=grid.best_estimator_.predict(X_test)
print(confusion_matrix(y_test['HIV_active'],np.
    transpose(np.matrix(y_pred_MLP))))
print(grid.score(X_test,y_test['HIV_active']))
```

Listing 27: GridSearchCV-MLP [4]

```
best mean cv score=  0.7064590605580141
best parameters=  {'activation': 'tanh', 'alpha': 0.0001, '
    hidden_layer_sizes': 255, 'learning_rate': 'constant'}
[[3812   25]
 [  84   60]]
0.7050755798801147
```

Listing 28: GridSearchCV-MLP results

```
param_grid={'activation':['identity', 'tanh', 'relu'],
    'hidden_layer_sizes':[100,150,255],'learning_rate'
    : ['constant','adaptive'],'alpha': [0.0001, 0.05]}
grid=GridSearchCV(MLPClassifier(),param_grid=
    param_grid,cv=5,scoring='balanced_accuracy',
    return_train_score=False)
grid.fit(X_training_data_PCA.to_numpy().astype('float'
    ),(np.transpose((y_training_data_PCA['HIV_active'
    ]).to_numpy().astype('int'))))
print("best mean cv score= ",grid.best_score_)
print("best parameters= ",grid.best_params_)
y_pred_MLP_pca=grid.best_estimator_.predict(X_test_pca
    )
print(confusion_matrix(y_test['HIV_active'],np.
    transpose(np.matrix(y_pred_MLP_pca))))
print(grid.score(X_test_pca,y_test['HIV_active']))
```

Listing 29: GridSearchCV-MLP [4]

```
 [  83   61]]
0.7052900486491183
```

Listing 30: GridSearchCV-MLP-PCA best mean cv score

6 EVALUATION

6.1 Use other preprocessing methods

I even tried other feature selection methods, using `SelectKBest` and `chi2` from the `sklearn.feature_selection` class. The results were better than I expected but generally not comparable to the written results. For example, I will briefly write the results of several algorithms.

In this test, it should be noted that 50 of the best features were selected. Also, after implementing the PCA algorithm, the algorithm found 15 optimal directions to maintain the maximum variance of the data. So, I held more than 93.4% of the data variance thanks to the PCA algorithm.

1. KNN-PCA
   - cv score: 66.92346027308284%
   - Test set score: 66.98382244103356%
2. Decision Tree-PCA
   - cv score: 60.45851339643267%
   - Test set score: 62.33432281108215%
3. Decision Tree
   - cv score: 64.87972726669244%
   - Test set score: 64.55047140972423%
4. XGBoost
   - cv score: 65.21894713707077%
   - Test set score: 65.76370788686167%

These results were enough to make me abandon this method for predicting labels.

6.2 Select the final model

In this section, I want to compare the accuracy of different algorithms 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 want to examine the final accuracy of the algorithm on the test set and CV score.

To select the final algorithm, it is obvious that we must select the best accuracy of the algorithm on the test set. Note Figure 7. As you can see, the KNN-PCA algorithm was the most accurate. The second place belongs to MLP.

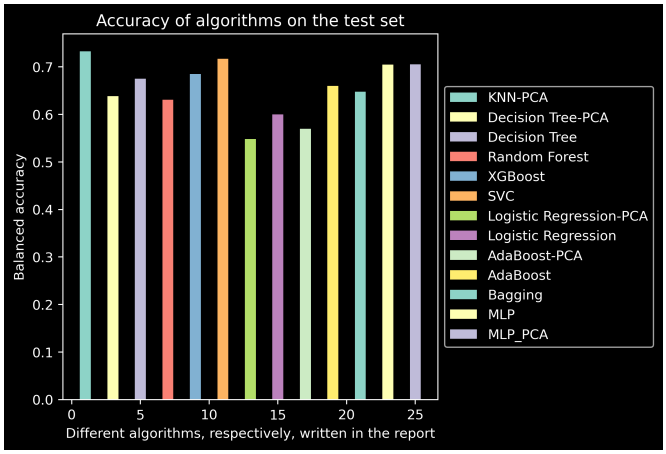

Figure 6: Accuracy of algorithms on the test set[19]

We have to keep in mind that the criteria for selecting hyperparameters were cv scores, so it would be better to show you a similar chart based on cv scores. Please pay attention to the Figure 7.

Looking at Figure 7, you may not notice the difference at first glance with Figure 6. For this reason, I have prepared Figure 8 for you, and all the bars in Figure 7 are displayed in a lighter gray color. All bars in Figure 6 are displayed in the initial format.

Since the CV score is our main criterion for choosing the best algorithm, I sort the previous charts based on the CV score.

I am delighted that the best algorithm I chose based on the CV score also performed better in the test set than the other algorithms. As you can

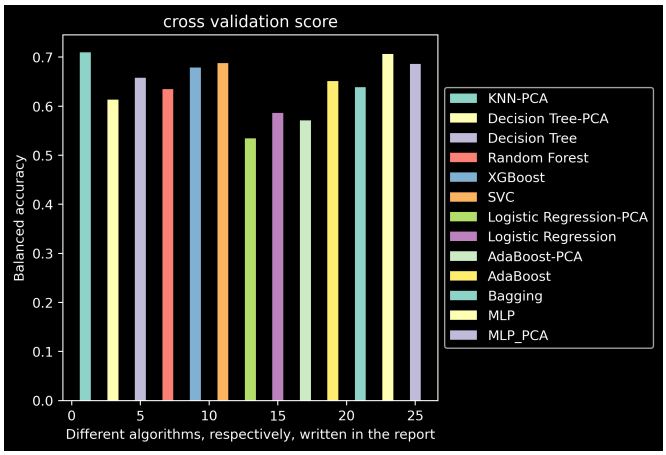

Figure 7: cross validation score[19]

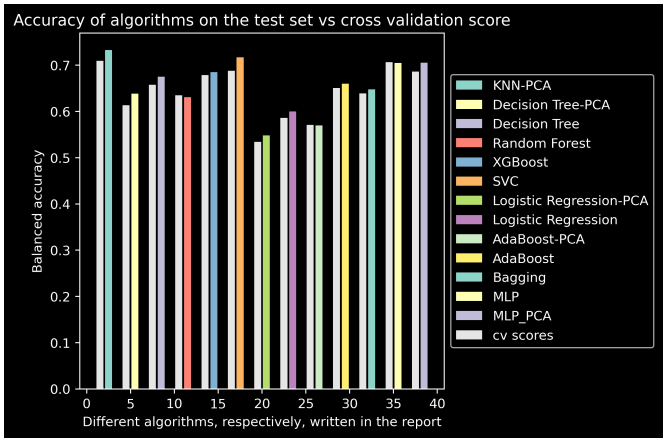

Figure 8: Accuracy of algorithms on the test set vs cross validation score[19]

see, Figure 9 shows the accuracy of the algorithms on the test set (with the Balanced accuracy meter). Of course, the algorithms are sorted based on their CV scores.

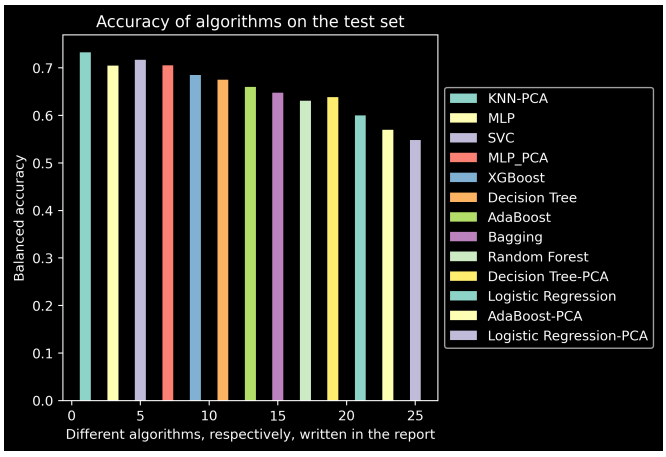

Figure 9: Accuracy of algorithms on the test set (sorted)[19]

Figure 10, which in my opinion does not need to be introduced, is our main criterion for choosing the best model.

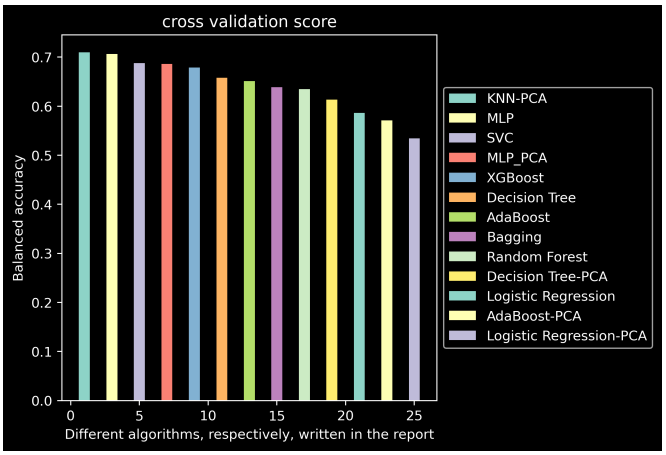

Figure 10: cross validation score (sorted)[19]

In Figure 11, you can also see the comparison of Figures 9 and 10 in one frame.

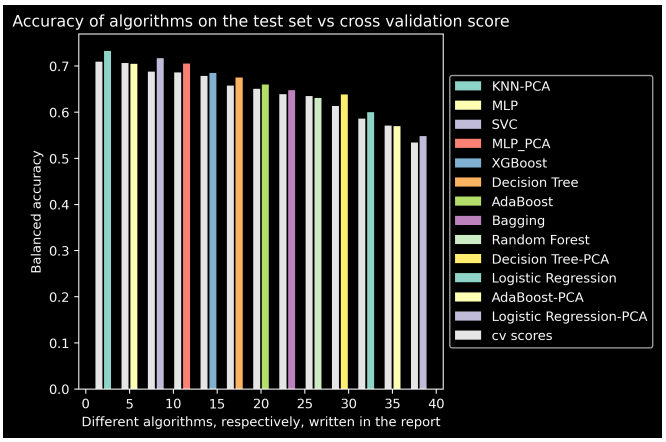

Figure 11: Accuracy of algorithms on the test set vs cross validation score (sorted)[19]

6.3 The final accuracy of the chosen algorithm

So far, we have found that the KNN-PCA algorithm has performed best. To determine the absolute accuracy, we need to look at the confusion matrix of this algorithm. As you can see in Listing 6, the confusion matrix is as follows.

$$\text{confusion matrix} = \begin{bmatrix} 3757 & 80 \\ 74 & 70 \end{bmatrix}$$

Since our main measure for this project was Balanced accuracy, the final accuracy is announced with the same meter, but for information on other meters, you can also see Figure 12.

$$\text{Balanced accuracy} = 73.26307445052559\%$$

Because maybe the people reading this report have worked with different meters. I rewrote the accuracy of the final algorithm based on some famous measures in Figure 12. The formula for each measure is also written in front of the measure name. I hope you enjoyed reading this report.

Measure	Value	Derivations
Sensitivity	0.9807	TPR = TP / (TP + FN)
Specificity	0.4667	SPC = TN / (FP + TN)
Precision	0.9792	PPV = TP / (TP + FP)
Negative Predictive Value	0.4861	NPV = TN / (TN + FN)
False Positive Rate	0.5333	FPR = FP / (FP + TN)
False Discovery Rate	0.0208	FDR = FP / (FP + TP)
False Negative Rate	0.0193	FNR = FN / (FN + TP)
Accuracy	0.9613	ACC = (TP + TN) / (P + N)
F1 Score	0.9799	F1 = 2TP / (2TP + FP + FN)
Matthews Correlation Coefficient	0.4562	TP*TN - FP*FN / sqrt((TP+FP)*(TP+FN)*(TN+FP)*(TN+FN))

Figure 12: Other measurements of the accuracy of the selected algorithm

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

NOTES

[1]This statistic is related to the year 2020.

[2]This statistic is related to the year 2020.

[3]This way, data labeled **zero** will not affect missing features labeled **one**.

[4]**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]

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

[6]**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.[4]

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

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

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

[10]Independent and identically distributed

[11]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.

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] unaids.org. Global hiv aids statistics, 2022.

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

[4] 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.

[5] 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.

[6] 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.

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

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

[9] 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.

[10] 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.

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

[12] statisticssolutions.com. What is logistic regression?, 2022.

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

[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.