SCIKIT-LEARN PYTHON

Intelligent Systems

Alejandro Silva Rodríguez
Grado en ingeniería de la Salud

Index

Introduction

In this practice, we are going to develop some classifiers in order to understand the way they work. We are going to use the scikit-learn package of python, one of the most used tools in data science nowadays. For divulgation purposes, I am going to describe step by step every data transformation, algorithm implementation, performance evaluation and why I take the decisions.

Dataset

First of all, I am going to present my dataset election. I choose CDC Diabetes Health Indicators from the UCI Repository of Machine Learning Databases. It contains healthcare statistics and lifestyle survey information about 253680 persons. We can find Categorical and Integer Types between the 21 features. The target attribute is Diabetes binary and tells whether a person is diabetic (prediabetic included) or not.

Description of the attributes:

Variable Name	Role	Type	Description
ID	ID	Integer	Patient ID
Diabetes_binary	Target	Binary	0 = no diabetes 1 = prediabetes or diabetes
HighBP	Feature	Binary	0 = no high BP 1 = high BP
HighChol	Feature	Binary	0 = no high cholesterol 1 = high cholesterol
CholCheck	Feature	Binary	0 = no cholesterol check in 5 years 1 = yes cholesterol check in 5 years
BMI	Feature	Integer	Body Mass Index
Smoker	Feature	Binary	Have you smoked at least 100 cigarettes in your entire life? [Note: 5 packs = 100 cigarettes] 0 = no 1 = yes
Stroke	Feature	Binary	(Ever told) you had a stroke. 0 = no 1 = yes
HeartDiseaseorAttack	Feature	Binary	coronary heart disease (CHD) or myocardial infarction (MI) 0 = no 1 = yes
PhysActivity	Feature	Binary	physical activity in past 30 days - not including job 0 = no 1 = yes
Fruits	Feature	Binary	Consume Fruit 1 or more times per day 0 = no 1 = yes
Veggies	Feature	Binary	Consume Vegetables 1 or more times per day 0 = no 1 = yes
HvyAlcoholConsump	Feature	Binary	Heavy drinkers (adult men having more than 14 drinks per week and adult women having more than 7 drinks per week) 0 = no 1 = yes
AnyHealthcare	Feature	Binary	Have any kind of health care coverage, including health insurance, prepaid plans such as HMO, etc. 0 = no 1 = yes
NoDocbcCost	Feature	Binary	Was there a time in the past 12 months when you needed to see a doctor but could not because of cost? 0 = no 1 = yes
GenHlth	Feature	Integer	Would you say that in general your health is: scale 1-5 1 = excellent 2 = very good 3 = good 4 = fair 5 = poor
MentHlth	Feature	Integer	Now thinking about your mental health, which includes stress, depression, and problems with emotions, for how many days during the past 30 days was your mental health not good? scale 1-30 days
PhysHlth	Feature	Integer	Now thinking about your physical health, which includes physical illness and injury, for how many days during the past 30 days was your physical health not good? scale 1-30 days
DiffWalk	Feature	Binary	Do you have serious difficulty walking or climbing stairs? 0 = no 1 = yes
Sex	Feature	Binary	0 = female 1 = male

I found the dataset on UCI repository, but it was originally uploaded on Kaggle.com:

Centers for Disease Control and Prevention (2017). Behavioral Risk Factor Surveillance System. Behavioral Risk Factor Surveillance System (Diabetes Health Indicators Dataset (kaggle.com))

In order to perform a better classification, I am going to normalize and balance the data. There are more nondiabetic samples than diabetic, and this may cause an incorrect classification. Anyway, I am going to present the results without balancing the dataset later to see the difference.

Classification

Classification is a datamining field which aim is to categorize samples into predefined classes, categories or groups based on their attributes. It is a part of supervised learning, so the samples have x_n attributes and a goal (tag) class y.

The goal of classification is to build a model that accurately predicts the class labels of new instances based on their features. There are two main types of classification: binary classification and multi-class classification. Binary classification involves classifying instances into two classes, such as "positive" or "negative" while multi-class classification involves classifying instances into more than two classes. In this case we are going to classify binary data.

Naïve Bayes

Introduction

Naïve Bayes classifier is a probabilistic classifier based on Baye's Theorem with strong independence assumptions between the input x_i and the given class y.

$$P(y \mid \mathbf{x}) = \frac{P(y)P(\mathbf{x} \mid y)}{\sum_{v \in V} P(v)P(\mathbf{x} \mid v)} \qquad P(\mathbf{x} \mid y) = \prod_{d=1}^{D} P(x_d \mid y)$$

It is used the m-estimate to estimate the posterior probabilities $P(x_d|y)$. Where n is the number of training examples of class y, p is a prior probability, and m is a constant which expresses our confidence in p, measured in number of samples.

$$P(x_d \mid y) \approx \frac{n' + mp}{n + m}$$

Implementation

In scikit-learn package, we can find the naïve bayes classificator implementation as GaussianNB

Firstly, we import all the packages.

```
###DATASET Y MANIPULATION
from ucimlrepo import fetch_ucirepo
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
###NAIVE BAYES
from sklearn.naive_bayes import GaussianNB
###METRICS
from sklearn import metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix
import matplotlib.pyplot as plt
```

Secondly, we import the dataset from the UCI repository with the help of the ucimrepo package.

```
# fetch dataset
diabetes = fetch_ucirepo(id=891)

X = diabetes.data.features
y = diabetes.data.targets
```

In order to avoid irrelevant attributes for our predictive model, we are going to see a correlations table ordered.

```
### CORRELACION
data = pd.concat([X, y], axis=1)  # Combina 'X' y 'y' en un solo DataFrame

# Calcula las correlaciones entre todas las características y 'y'
correlations = data.corrwith(data['Diabetes_binary'])

# Convierte las correlaciones a valores absolutos y ordénalas de manera
descendente
correlations = correlations.abs().sort_values(ascending=False)

# Muestra las características más correlacionadas con 'y'
print(correlations)
```

the output:

Diabetes_binary	1.000000
GenHlth	0.405729
HighBP	0.375482
BMI	0.291842
HighChol	0.284405
DiffWalk	0.276154
Age	0.268378
Income	0.225007
PhysHlth	0.216938
HeartDiseaseorAttack	0.210804

Education	0.170981
PhysActivity	0.158905
Stroke	0.128851
CholCheck	0.111140
MentHlth	0.091841
HvyAlcoholConsump	0.091521
Smoker	0.083811
Veggies	0.080183
Fruits	0.060454
Sex	0.045762
NoDocbcCost	0.040271
AnyHealthcare	0.025165

I am going to normalize the data and delete the uninformative attributes to avoid the "curse of dimensionality".

```
scaler = StandardScaler()
#scaler = MinMaxScaler()
X_n = scaler.fit_transform(X)
X = pd.DataFrame(X_n, columns=X.columns)
X =
X.drop(columns=['AnyHealthcare','NoDocbcCost','Sex','Fruits','Veggies'])
######## TO BALANCE THE CLASSES #######

y0 = y[y == 0]
y1 = y[y == 1]
y0_sampled = y0.sample(frac=0.2, random_state=10)
y_combined = pd.concat([y0_sampled, y1]).dropna()
y=pd.DataFrame(y_combined)
X = X.loc[(~y_combined.isna()).index].dropna()
```

I am going to set a param grid with the hyperparameters that I want to optimize in validation.

```
param_grid = {
    'priors': [None, [0.25, 0.75], [0.4, 0.6], [0.3, 0.7]],
    'var_smoothing': [1e-9, 1e-8, 1e-7, 5e-8],
}
```

Now, it is time to do the cross-validation. I am going to use the stratified 10-fold to be more statistically correct. Later, I am going to divide training into training and validation to adjust the hyperparameters.

```
#KFOLD
rd = 10
particiones = 10
skf = StratifiedKFold(n_splits=particiones,shuffle=True,random_state=rd)

#This is for doing the performance evaluation later.
i=1
```

```
ACCM=[]
PRM=[]
FALLM=[]
RCM=[]
F1M=[]
AUCM=[]
for train, test in skf.split(X,y):

   X_train,  X_test,  y_train,  y_test  =  X.iloc[train],  X.iloc[test],
y.iloc[train], y.iloc[test]
  ###DIVISION OF TRAINING INTO VALIDATION (20%) AND TRAINING (80%)
   X_train,  X_val,  y_train,  y_val  =  train_test_split(X_train,  y_train,
test_size=0.2, random_state=rd)
  ### HYPERPARAMETERS VALIDATION
  y_val=np.ravel(y_val)
  acc=0
  for prior in param_grid['priors']:
      for varsm in param_grid['var_smoothing']:

          gnb1=GaussianNB(priors=prior,var_smoothing=varsm)
          gnb1.fit(X_train, y_train)
          y_val_prob = gnb1.predict_proba(X_val)
          auc1 = metrics.roc_auc_score(y_val, y_val_prob[:,1])

          if auc1>acc:
              gnb=gnb1

  ### We get the gnb with better AUC
  y_train=np.ravel(y_train)
  gnb.fit(X_train, y_train)
  y_pred = gnb.predict(X_test)
```

It is time to measure the performance evaluation of each classifier.

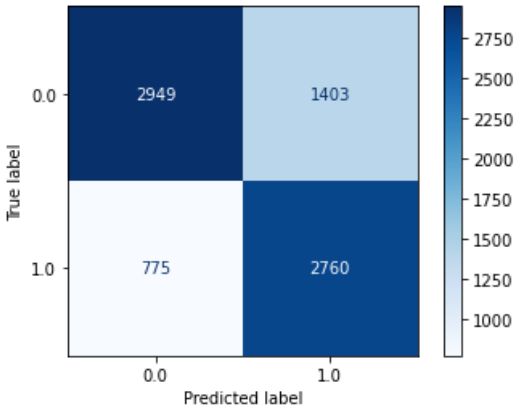
```
  ### PERFORMANCE EVALUATION
  conf_matrix = confusion_matrix(y_test, y_pred)
  TN = conf_matrix[0, 0]  # True Negatives
  FP = conf_matrix[0, 1]  # False Positives
  FN = conf_matrix[1, 0]  # False Negatives
  TP = conf_matrix[1, 1]  # True Positives
  #Accuracy
  acc_score = accuracy_score(y_test, y_pred)
  #precision
  precision = TP / (TP + FP)
  #Fallout
  fallout = FP / (FP + TN)
  #Recall
  recall = recall_score(y_test, y_pred)
  #f1
```

```
  f1 = f1_score(y_test, y_pred)
  #Matriz de confusion
        metrics.ConfusionMatrixDisplay.from_estimator(gnb,        X_test,
y_test,cmap=plt.cm.Blues)
  #pintamos la curva
  y_prob = gnb.predict_proba(X_test)
  auc = metrics.roc_auc_score(y_test, y_prob[:,1])
  metrics.RocCurveDisplay.from_estimator(gnb, X_test, y_test)

  i+=1
  ### to compute the mean of the evaluators.
  ACCM.append(acc_score)
  PRM.append(precision)
  FALLM.append(fallout)
  RCM.append(recall)
  F1M.append(f1)
  AUCM.append(auc)

#### Mean of performance evaluations.
print('')
print(f"The mean ACCURACY is {round(np.mean(np.array(ACCM)),4)} with a
standard deviation of {round(np.std(np.array(ACCM)),4)}")
print(f"The mean PRECISION is {round(np.mean(np.array(PRM)),4)}  with a
standard deviation of {round(np.std(np.array(PRM)),4)}")
print(f"The mean FALLOUT is {round(np.mean(np.array(FALLM)),4)}  with a
standard deviation of {round(np.std(np.array(FALLM)),4)}")
print(f"The mean RECALL is {round(np.mean(np.array(RCM)),4)}  with a
standard deviation of {round(np.std(np.array(RCM)),4)}")
print(f"The mean F1 is {round(np.mean(np.array(F1M)),4)}  with a standard
deviation of {round(np.std(np.array(F1M)),4)}")
print(f"The mean AUC is {round(np.mean(np.array(AUCM)),4)}  with a
standard deviation of {round(np.std(np.array(AUCM)),4)}")
```

output:
The mean ACCURACY is 0.7208 with a standard deviation of 0.0049
The mean PRECISION is 0.6598 with a standard deviation of 0.0053
The mean FALLOUT is 0.3257 with a standard deviation of 0.0068
The mean RECALL is 0.778 with a standard deviation of 0.0048
The mean F1 is 0.7141 with a standard deviation of 0.0045
The mean AUC is 0.7853 with a standard deviation of 0.0057

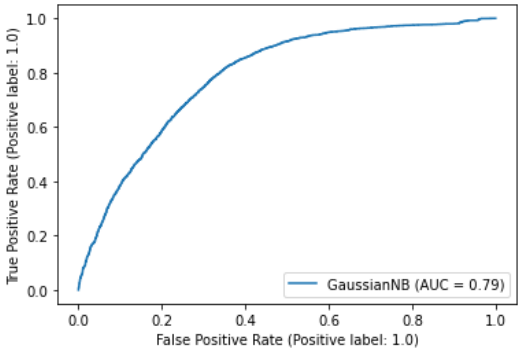
Since the models predicted the samples with similar performance, I am going to show some graphics form one model.

Confusion matrix:



There is not a big unbalance between False Positive and False Negative.

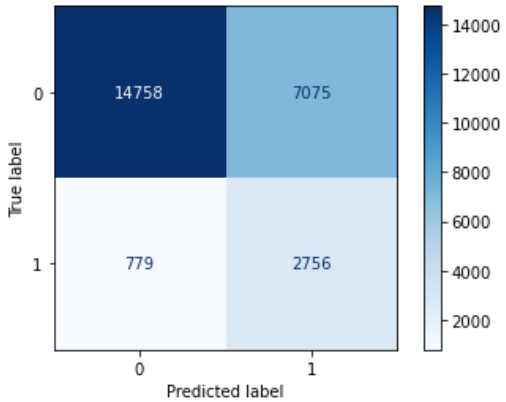
ROC Curve:



The performance was quite good.

If we do not do the initial balance between classes, we will get:

The mean ACCURACY is 0.6888 with a standard deviation of 0.0018
The mean PRECISION is 0.2787 with a standard deviation of 0.0013
The mean FALLOUT is 0.3254 with a standard deviation of 0.0026
The mean RECALL is 0.7767 with a standard deviation of 0.0058
The mean F1 is 0.4102 with a standard deviation of 0.0019
The mean AUC is 0.7848 with a standard deviation of 0.0026
At first sight, results are similar, but with the confusion matrix we can see what is wrong.



We can see that it is totally unbalance. There are too much False Positives. The precision and F1 are worse than before.

Nearest Neighbors

Introduction

The nearest neighbor algorithm is one of the "Lazy Learning" algorithms since it doesn't have a formal training phase. Instead, it follows an "instance-based learning" approach, where the model serves as a simple storage for training data instances.

The premise for classifying a new instance is based on analyzing the class of "similar" instances. The key here lies in defining what we mean by similarity between instances. To achieve this, we need to establish what is called a distance function, which assigns an output value between instances based on how similar they are. Since instances are represented as numerical variables, the most common of these distance functions is the Euclidean distance but there are a lot of ways to measure the distance:

-Minkowski metric

Euclidean distance (p=2) $d(x,y) = \sqrt[p]{\sum_i |x_i - y_i|^p}$

Manhattan distance (p=1) $d(x,y) = \sum |x_i - y_i|^2$

-Euclidean squared distance $d(x,y) = \max_i |x_i - y_i|$

-Chebyshev distance $d(x,y) = 1 - r_{x,y}$

-Pearson correlation distance $r_{x,y} = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2}\sqrt{\sum_i (y_i - \bar{y})^2}}$

In the nearest neighbor algorithm, to determine the output class of a new instance, you must calculate the distance value for all the available instances in the training set. Then, the most frequent class among the k nearest neighbors to the new instance is assigned as the output class.

Implementation

To avoid redundancy, in this implementation we are not explaining the initial data transformation. I have decided not to drop any column since the performance is better with all columns in this case.

We are going to implement k-nearest neighbors using the KNeighborsClassifier from scikit-learn and we are going to do it analogically to the previous classifier.

```
###DATASET MANIPULATION
from ucimlrepo import fetch_ucirepo
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
### KNN
from sklearn.neighbors import KNeighborsClassifier
###METRICS
from sklearn import metrics
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix
import matplotlib.pyplot as plt
```

Since I was not able to connect to the ucimrepo website (the server was down), I had to download the dataset and use pd.read_csv() method.

```
  File ~\AppData\Roaming\Python\Python311\site-
packages\ucimlrepo\fetch.py:69 in fetch_ucirepo
    raise ConnectionError('Error connecting to server')

ConnectionError: Error connecting to server
```

```
# fetch dataset
diabetes                                                                =
pd.read_csv("C://Users//alexs//OneDrive//Documentos//UNI//TERCERO//Sistem
as                                                    inteligentes//Scikit-
learn//Dataset//diabetes_binary_health_indicators_BRFSS2015.csv")

X = diabetes
y = diabetes["Diabetes_binary"]
X.drop(["Diabetes_binary"], axis = 1, inplace=True)
### NORMALIZATION
scaler = StandardScaler()
#scaler = MinMaxScaler()
X_n = scaler.fit_transform(X)
X = pd.DataFrame(X_n, columns=X.columns)

######## TARJETS BALANCE #######

y0 = y[y == 0]
y1 = y[y == 1]
y0_sampled = y0.sample(frac=0.2, random_state=10)
y_combined = pd.concat([y0_sampled, y1]).dropna()
y=pd.DataFrame(y_combined)
X = X.loc[(~y_combined.isna()).index].dropna()
```

In this case, the params that we are going to validate are the number of neighbors and the way the distance is measured. When p is equal to 1, we use Manhattan distance. When p is equal to 2 we use Euclidean distance.

```
### HYPER PARAMETERS
param_grid = {
    'n_neighbors': [3,301,1001],
    'weights': ['distance'],
    'p': [1, 2]
}
#KFOLD
rd = 10
particiones = 10
skf = StratifiedKFold(n_splits=particiones,shuffle=True,random_state=rd)
i=1
```

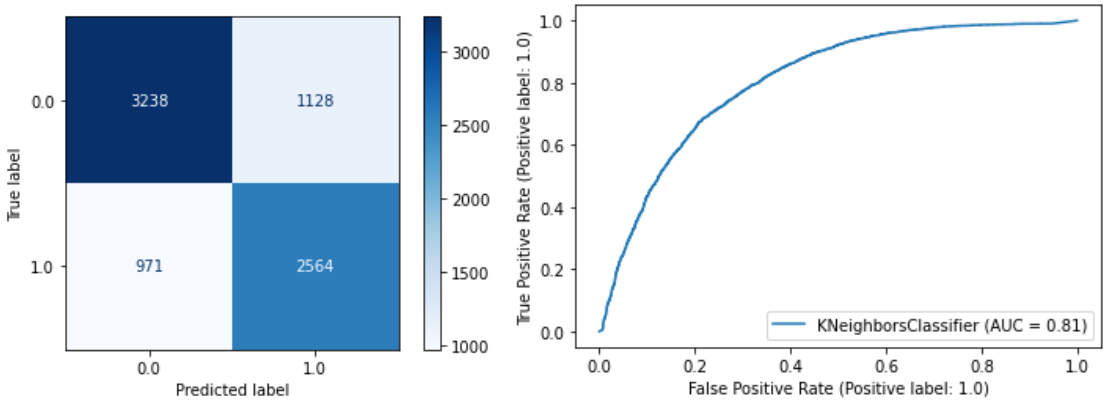
```
ACCM=[]
PRM=[]
FALLM=[]
RCM=[]
F1M=[]
AUCM=[]
for train, test in skf.split(X,y):

   X_train,  X_test,  y_train,  y_test  =  X.iloc[train],  X.iloc[test],
y.iloc[train], y.iloc[test]
  ###DIVISION OF TRAINING INTO VALIDACION AND TRAINING
   X_train,  X_val,  y_train,  y_val  =  train_test_split(X_train,  y_train,
test_size=0.4, random_state=rd)
  ### HYPERPARAMETERS VALIDATION
  y_val=np.ravel(y_val)
  auc=0
  for n_neighbors_i in param_grid['n_neighbors']:
        for weights_i in param_grid['weights']:
            for p_i in param_grid['p']:
                knn1 = KNeighborsClassifier(n_neighbors= n_neighbors_i,
                                            weights= weights_i,
                                            p=p_i,
                                            leaf_size= 1,)

                knn1.fit(X_train, y_train)
                y_val_prob = knn1.predict_proba(X_val)
                auc1 = metrics.roc_auc_score(y_val, y_val_prob[:,1])

                if auc1>auc:
                   auc=auc1
                   knn=knn1
  ### WE GET THE KNN WITH THE BEST AUC

  y_train=np.ravel(y_train)
  knn.fit(X_train, y_train)
  y_pred = knn.predict(X_test)
  knn.get_parameters()

  ### PERFORMANCE EVALUATION
  conf_matrix = confusion_matrix(y_test, y_pred)
  TN = conf_matrix[0, 0]  # True Negatives
  FP = conf_matrix[0, 1]  # False Positives
  FN = conf_matrix[1, 0]  # False Negatives
  TP = conf_matrix[1, 1]  # True Positives
  #Accuracy
  acc_score = accuracy_score(y_test, y_pred)
  #precision
  precision = TP / (TP + FP)
  #Fallout
```

```
  fallout = FP / (FP + TN)
  #Recall
  recall = recall_score(y_test, y_pred)
  #f1
  f1 = f1_score(y_test, y_pred)
  #Confusion matrix
  metrics.ConfusionMatrixDisplay.from_estimator(knn, X_test,
y_test,cmap=plt.cm.Blues)
  y_prob = knn.predict_proba(X_test)
  auc = metrics.roc_auc_score(y_test, y_prob[:,1])
  metrics.RocCurveDisplay.from_estimator(knn, X_test, y_test)
  i+=1

  ACCM.append(acc_score)
  PRM.append(precision)
  FALLM.append(fallout)
  RCM.append(recall)
  F1M.append(f1)
  AUCM.append(auc)
#### MESUREMENTS MEANS
print('')
print(f"The mean ACCURACY is {round(np.mean(np.array(ACCM)),4)} with a
standard deviation of {round(np.std(np.array(ACCM)),4)}")
print(f"The mean PRECISION is {round(np.mean(np.array(PRM)),4)}  with a
standard deviation of {round(np.std(np.array(PRM)),4)}")
print(f"The mean FALLOUT is {round(np.mean(np.array(FALLM)),4)}  with a
standard deviation of {round(np.std(np.array(FALLM)),4)}")
print(f"The mean RECALL is {round(np.mean(np.array(RCM)),4)}  with a
standard deviation of {round(np.std(np.array(RCM)),4)}")
print(f"The mean F1 is {round(np.mean(np.array(F1M)),4)}  with a standard
deviation of {round(np.std(np.array(F1M)),4)}")
print(f"The mean AUC is {round(np.mean(np.array(AUCM)),4)}  with a
standard deviation of {round(np.std(np.array(AUCM)),4)}")
```

Output:

The mean ACCURACY is 0.7375 with a standard deviation of 0.0046
The mean PRECISION is 0.698 with a standard deviation of 0.0044
The mean FALLOUT is 0.255 with a standard deviation of 0.0066
The mean RECALL is 0.7282 with a standard deviation of 0.0128
The mean F1 is 0.7127 with a standard deviation of 0.0068
The mean AUC is 0.8117 with a standard deviation of 0.0041

Confusion Matrix and ROC Curve:



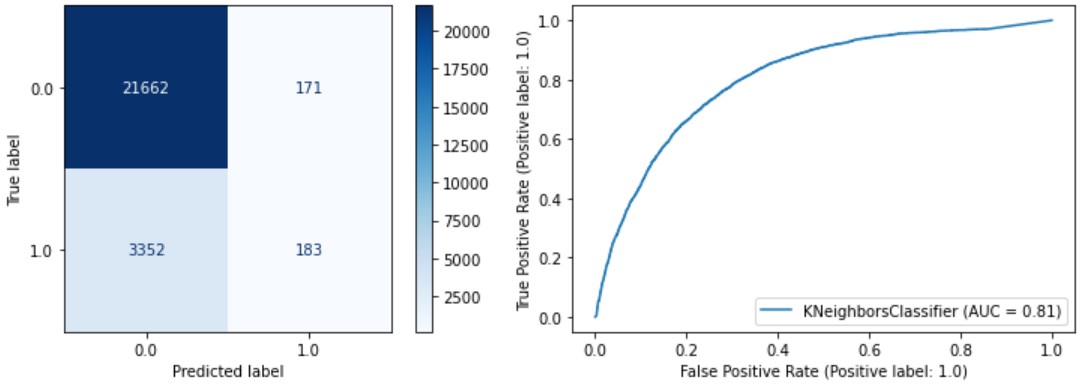
Best model parameters:
{'algorithm': 'auto', 'leaf_size': 1, 'metric': 'minkowski', 'metric_params': None, 'n_jobs': None, 'n_neighbors': 301, 'p': 1, 'weights': 'distance'}

In general, knn is very stable and there is no big changes when you change the hyperparamenters. However, The best number of neighbors is between 301 and 501. I have selected a odd number of neighbors to avoid draws.

To illustrate the importance of data balance, here is an example k-nearest neighbor with an unbalanced dataset:
Output:

El valor medio de ACCURRACY para el dt es 0.8613 con una desviación típica de 0.0005
El valor medio de PRECISION para el dt es 0.5215 con una desviación típica de 0.0186
El valor medio de FALLOUT para el dt es 0.0076 con una desviación típica de 0.0006
El valor medio de RECALL para el dt es 0.0514 con una desviación típica de 0.0029
El valor medio de F1 para el dt es 0.0935 con una desviación típica de 0.0049
El valor medio de AUC para el dt es 0.8055 con una desviación típica de 0.0039

Confusion Matrix and ROC Curve:



Like we saw in the previous model, the performance with unbalanced data is very bad.

Decision Tree

Introduction

A decision tree is a function that receive a vector of attributes X and returns a decision. Starting from the root, the vector must pass through several nodes and arrive to a leaf node.

In each internal node of the tree there is a test that decides the next node of the tree depending on the attribute a_i. Between each node, there is a branch labeled with possibles values of the attribute a_i. When the node is a leaf node, then, it returns the target value.

One of the most important parameters is 'criterion'. It determines how the tree is divided, and there are two ways:

-Entropy: It gives the number of bits needed to codify the goal values of the examples.

$$H(Examples) = -\sum_{v \in GoalAttribute} \frac{|Examples(v)|}{|Examples|} \log_2 \frac{|Examples(v)|}{|Examples|}$$

-Information Gain: the difference in entropy as Examples is split on an attribute A.

$$IG(Examples, A) = H(Examples) - \sum_{t \in T} \frac{|t|}{|Examples|} H(t)$$

The others hyper parameters like 'max_depth' or 'max_features' are important to determine the degree of fitting of the tree. If we have a long tree, it may be overfitted, and consequently, it can't predict samples that were not used to train the model.

Implementation

```
###DATASET MANIPULATION
from ucimlrepo import fetch_ucirepo
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
### Decision Tree
from sklearn import tree
###METRICS
from sklearn import metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix
import matplotlib.pyplot as plt
```

```
# fetch dataset
diabetes = fetch_ucirepo(id=891)

X = diabetes.data.features
y = diabetes.data.targets
### NORMALIZATION
scaler = StandardScaler()
#scaler = MinMaxScaler()
X_n = scaler.fit_transform(X)
X = pd.DataFrame(X_n, columns=X.columns)
X                                                                    =
X.drop(columns=['AnyHealthcare','NoDocbcCost','Sex','Fruits','Veggies'])
######## TARJETS BALANCE #######

y0 = y[y == 0]
y1 = y[y == 1]
y0_sampled = y0.sample(frac=0.2, random_state=10)
y_combined = pd.concat([y0_sampled, y1]).dropna()
y=pd.DataFrame(y_combined)
X = X.loc[(~y_combined.isna()).index].dropna()

### HYPER PARAMETERS
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [7, 10, 15,30,50],  # Añadido 10 y 15
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 3],  # Añadido 3
    'max_features': [None, 'sqrt', 'log2', 0.5],  # Añadido 0.5
    'max_leaf_nodes': [None, 5, 10, 15],  # Añadido 15
    'min_impurity_decrease': [0.0, 0.1, 0.2]
}
#KFOLD
rd = 10
particiones = 10
skf = StratifiedKFold(n_splits=particiones,shuffle=True,random_state=rd)

i=1
ACCM=[]
PRM=[]
FALLM=[]
RCM=[]
F1M=[]
AUCM=[]
for train, test in skf.split(X,y):

   X_train,  X_test,  y_train,  y_test  =  X.iloc[train],  X.iloc[test],
y.iloc[train], y.iloc[test]
  ###DIVISION OF TRAINING INTO VALIDACION AND TRAINING
```

```
   X_train,  X_val,  y_train,  y_val  =  train_test_split(X_train,  y_train,
test_size=0.2, random_state=rd)
  ### HYPER PARAMETERS VALIDATION
  y_val=np.ravel(y_val)
  auc=0
  for criterion_i in param_grid['criterion']:
      for max_depth_i in param_grid['max_depth']:
          for min_samples_split_i in param_grid['min_samples_split']:
              for min_samples_leaf_i in param_grid['min_samples_leaf']:
                  for max_features_i in param_grid['max_features']:
                     for max_leaf_nodes_i in param_grid['max_leaf_nodes']:
                          for min_impurity_decrease_i in
param_grid['min_impurity_decrease']:


                              dt1 =
tree.DecisionTreeClassifier(criterion= criterion_i,
                          max_depth = max_depth_i,
                          max_features = max_features_i,
                          max_leaf_nodes = max_leaf_nodes_i,
                          min_impurity_decrease = min_impurity_decrease_i,
                          min_samples_leaf = min_samples_leaf_i,
                          min_samples_split = min_samples_split_i)

                              dt1.fit(X_train, y_train)
                              y_val_prob = dt1.predict_proba(X_val)
                              auc1 = metrics.roc_auc_score(y_val,
y_val_prob[:,1])
                              if auc1>auc:
                                  auc=auc1
                                  dt=dt1

  ### WE GET THE TREE WITH MORE AUC
  y_train=np.ravel(y_train)
  dt.fit(X_train, y_train)
  y_pred = dt.predict(X_test)
  dt.get_params()

  ### PERFORMANCE EVALUATION
  conf_matrix = confusion_matrix(y_test, y_pred)
  TN = conf_matrix[0, 0]  # True Negatives
  FP = conf_matrix[0, 1]  # False Positives
  FN = conf_matrix[1, 0]  # False Negatives
  TP = conf_matrix[1, 1]  # True Positives
  #Accuracy
  acc_score = accuracy_score(y_test, y_pred)
  #precision
  precision = TP / (TP + FP)
  #Fallout
```

```
  fallout = FP / (FP + TN)
  #Recall
  recall = recall_score(y_test, y_pred)
  #f1
  f1 = f1_score(y_test, y_pred)
  #Confusion Matrix
  metrics.ConfusionMatrixDisplay.from_estimator(dt, X_test,
y_test,cmap=plt.cm.Blues)

  y_prob = dt.predict_proba(X_test)
  auc = metrics.roc_auc_score(y_test, y_prob[:,1])
  metrics.RocCurveDisplay.from_estimator(dt, X_test, y_test)
  i+=1

  ACCM.append(acc_score)
  PRM.append(precision)
  FALLM.append(fallout)
  RCM.append(recall)
  F1M.append(f1)
  AUCM.append(auc)


#### MESUREMENTS MEANS
print('')
print(f"The mean ACCURACY is {round(np.mean(np.array(ACCM)),4)} with a
standard deviation of {round(np.std(np.array(ACCM)),4)}")
print(f"The mean PRECISION is {round(np.mean(np.array(PRM)),4)}  with a
standard deviation of {round(np.std(np.array(PRM)),4)}")
print(f"The mean FALLOUT is {round(np.mean(np.array(FALLM)),4)}  with a
standard deviation of {round(np.std(np.array(FALLM)),4)}")
print(f"The mean RECALL is {round(np.mean(np.array(RCM)),4)}  with a
standard deviation of {round(np.std(np.array(RCM)),4)}")
print(f"The mean F1 is {round(np.mean(np.array(F1M)),4)}  with a standard
deviation of {round(np.std(np.array(F1M)),4)}")
print(f"The mean AUC is {round(np.mean(np.array(AUCM)),4)}  with a
standard deviation of {round(np.std(np.array(AUCM)),4)}")
```

Best model parameters:

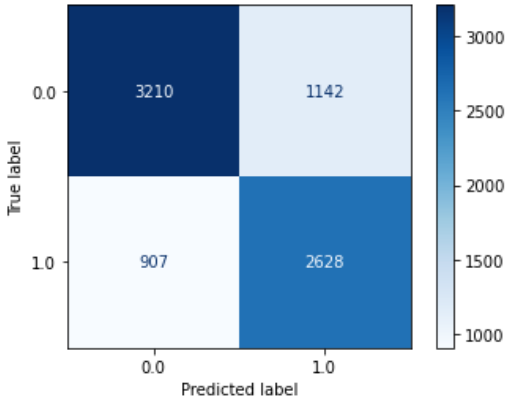
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': 7,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,

'random_state': None,
'splitter': 'best'}

Output:

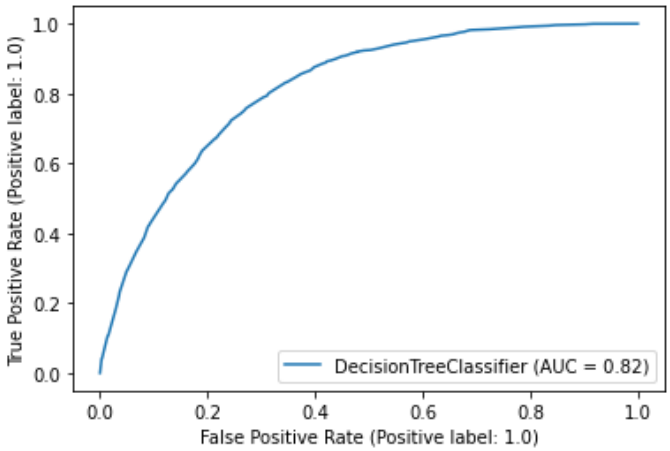
The mean ACCURACY is 0.7362 with a standard deviation of 0.0063
The mean PRECISION is 0.6951 with a standard deviation of 0.0098
The mean FALLOUT is 0.2616 with a standard deviation of 0.0153
The mean RECALL is 0.7334 with a standard deviation of 0.0166
The mean F1 is 0.7135 with a standard deviation of 0.0076
The mean AUC is 0.8131 with a standard deviation of 0.0061
Since the models predicted the samples with similar performance, I am going to show some graphics form one model.

Confusion matrix:



The confusion matrix shows that there are no strange behaviors and the model has a good performance.

ROC Curve:



It has a similar performance to k-nearest neighbor.

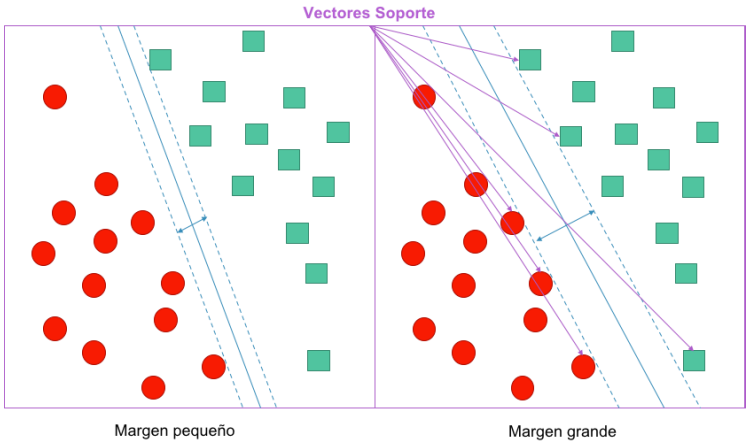
Support Vector Machine

Introduction

The goal of an SVM is to find a hyperplane that separates instances of two classes. A hyperplane is exactly the same type of discriminant function used in other linear classifiers, such as logistic regression. The equation of the hyperplane is essentially a dot product of a vector of input variables x with a vector of weights or importance w $\| x - w \|$:

$y=(x,w)= w_0+w1 \cdot x1+...+wn \cdot xn$

The difference between SVM and other methods of linear separation is that, among all possible hyperplanes that divide instances into two parts, the one achieving the maximum margin is chosen. This margin is calculated as the maximum distance between boundary instances, as depicted in the following figure:



The name of this learning technique is precisely determined by these instances on which the decision boundary "leans." They are the points closest to the hyperplane and directly influence its orientation to achieve the mentioned maximum margin.

There are different mechanisms or mathematical approaches to finding the optimal orientation of the hyperplane, but they mainly depend on the distance of misclassified instances (on the other side of the "linear boundary"). For this reason, the most important parameter of an SVM is the Cost, denoted as `C`:

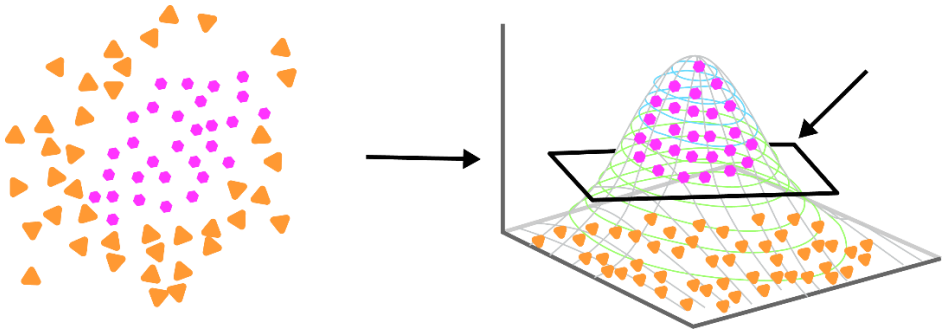
- A low value would accept making a certain number of classification errors, slightly lowering the prediction quality on the training set but seeking better generalization on the test set.

- A high value allows better fitting the model to the training data but implies a higher risk of overfitting.

Despite the above, a simple hyperplane is not the right solution when the classes represented in the problem are not linearly separable or when there is a lot of noise in the data. However, SVMs have been noted to achieve great predictive results even in complex problems.

The employed solution is known as the kernel trick, and it essentially involves mapping the data into a more complex space with non-linear variables and using the linear SVM classifier in this

new space. For example, by adding a new dimension or variable, we can find a suitable separation of the data, as seen in the figure mentioned above.



The two most common examples of such kernel functions are the following:

- Polynomial Function: $K(x,w)=\langle x,w\rangle^d$.This transforms the linear sum of weight w_i and variable x_i products into a higher-degree polynomial (2, 3, etc.). Thus, instead of a hyperplane or "straight line," we have a more complex data division.

- Radial Base Function (RBF): $K(x,w)= e^{-\frac{\|x-w\|^2}{2\cdot\sigma}}$.With the use of RBF functions, non-linear discriminant functions are represented as "circular" areas in this case.

Implementation

```
###DATASET MANIPULATION
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
### SVM
from sklearn import svm
###METRICS
from sklearn import metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix
import matplotlib.pyplot as plt


# fetch dataset
diabetes                                                                =
pd.read_csv("C://Users//alexs//OneDrive//Documentos//UNI//TERCERO//Sistem
as                                                      inteligentes//Scikit-
learn//Dataset//diabetes_binary_health_indicators_BRFSS2015.csv")

X = diabetes
y = diabetes["Diabetes_binary"]
X.drop(["Diabetes_binary"], axis = 1, inplace=True)
### NORMALIZATION
#scaler = StandardScaler()
scaler = MinMaxScaler()
X_n = scaler.fit_transform(X)
```

```
X = pd.DataFrame(X_n, columns=X.columns)
X                                                                    =
X.drop(columns=['AnyHealthcare','NoDocbcCost','Sex','Fruits','Veggies'])
######## TARJETS BALANCE #######

y0 = y[y == 0]
y1 = y[y == 1]
y0_sampled = y0.sample(frac=0.2, random_state=10)
y_combined = pd.concat([y0_sampled, y1]).dropna()
y=pd.DataFrame(y_combined)
X = X.loc[(~y_combined.isna()).index].dropna()
### HYPER PARAMETERS
param_grid = {
    'C': [1,10,20],
    'kernel': ['poly', 'rbf', 'linear']
}
```

I am not going to use the Sigmoid Kernel because it has a bad performance on this problem.

```
#KFOLD
rd = 10
particiones = 10
skf = StratifiedKFold(n_splits=particiones,shuffle=True,random_state=rd)

i=1
ACCM=[]
PRM=[]
FALLM=[]
RCM=[]
F1M=[]
AUCM=[]
for train, test in skf.split(X,y):

   X_train,  X_test,  y_train,  y_test  =  X.iloc[train],  X.iloc[test],
y.iloc[train], y.iloc[test]
  ###DIVISION OF TRAINING INTO VALIDATION (20%) AND TRAINING (80%)
   X_train,  X_val,  y_train,  y_val  =  train_test_split(X_train,  y_train,
test_size=0.2, random_state=rd)
  ###HYPERPARAMETERS VALIDATION
  y_val=np.ravel(y_val)
  auc=0
  for C_i in param_grid['C']:
      for kernel_i in param_grid['kernel']:
```

The library SVM implements a Classifier SVC model and a regression SVR model. In this problem, we need the classifier model.

```
        svm1 = svm.SVC(C=C_i,kernel=kernel_i,probability=True)
        svm1.fit(X_train, y_train)
        y_val_prob = svm1.predict_proba(X_val)
```

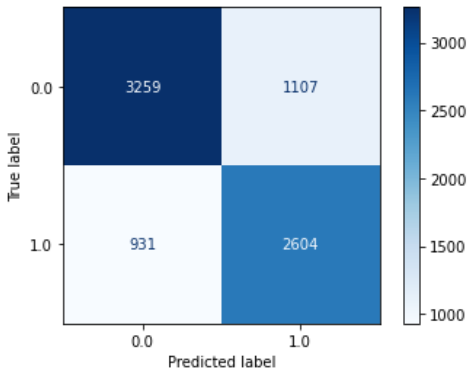
```
        auc1 = metrics.roc_auc_score(y_val, y_val_prob[:,1])

        if auc1>auc:
            auc=auc1
            SVM=svm1
  ### WE GET THE SVC WITH BETTER AUC
  y_train=np.ravel(y_train)
  SVM.fit(X_train, y_train)
  y_pred = SVM.predict(X_test)
  SVM.get_params()

  ### PERFORMANCE EVALUATION
  conf_matrix = confusion_matrix(y_test, y_pred)
  TN = conf_matrix[0, 0]  # True Negatives
  FP = conf_matrix[0, 1]  # False Positives
  FN = conf_matrix[1, 0]  # False Negatives
  TP = conf_matrix[1, 1]  # True Positives
  #Accuracy
  acc_score = accuracy_score(y_test, y_pred)
  #precision
  precision = TP / (TP + FP)
  #Fallout
  fallout = FP / (FP + TN)
  #Recall
  recall = recall_score(y_test, y_pred)
  #f1
  f1 = f1_score(y_test, y_pred)
  #Confusion matrix
        metrics.ConfusionMatrixDisplay.from_estimator(SVM,        X_test,
y_test,cmap=plt.cm.Blues)
  #ROC Curve
  y_prob = SVM.predict_proba(X_test)
  auc = metrics.roc_auc_score(y_test, y_prob[:,1])
  metrics.RocCurveDisplay.from_estimator(SVM, X_test, y_test)
  i+=1

  ACCM.append(acc_score)
  PRM.append(precision)
  FALLM.append(fallout)
  RCM.append(recall)
  F1M.append(f1)
  AUCM.append(auc)
#### MEASUREMENTS MEANS
print('')
print(f"The mean ACCURACY is {round(np.mean(np.array(ACCM)),4)} with a
standard deviation of {round(np.std(np.array(ACCM)),4)}")
print(f"The mean PRECISION is {round(np.mean(np.array(PRM)),4)}  with a
standard deviation of {round(np.std(np.array(PRM)),4)}")
```

```
print(f"The mean FALLOUT is {round(np.mean(np.array(FALLM)),4)}  with a
standard deviation of {round(np.std(np.array(FALLM)),4)}")
print(f"The mean RECALL is {round(np.mean(np.array(RCM)),4)}  with a
standard deviation of {round(np.std(np.array(RCM)),4)}")
print(f"The mean F1 is {round(np.mean(np.array(F1M)),4)}  with a standard
deviation of {round(np.std(np.array(F1M)),4)}")
print(f"The mean AUC is {round(np.mean(np.array(AUCM)),4)}  with a
standard deviation of {round(np.std(np.array(AUCM)),4)}")
```

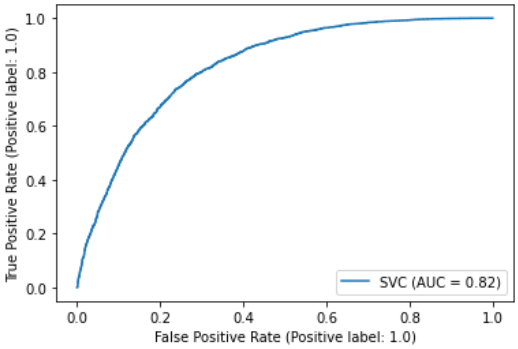
Best model:{'C': 10,
'break_ties': False,
'cache_size': 200,
'class_weight': None,
'coef0': 0.0,
'decision_function_shape': 'ovr',
'degree': 3,
'gamma': 'scale',
'kernel': 'linear',
'max_iter': -1,
'probability': True,
'random_state': None,
'shrinking': True,
'tol': 0.001,
'verbose': False}

In most of the cases, the best kernel is 'linear'.
Output:

The mean ACCURACY is 0.7444 with a standard deviation of 0.0048
The mean PRECISION is 0.7041 with a standard deviation of 0.0046
The mean FALLOUT is 0.2515 with a standard deviation of 0.0053
The mean RECALL is 0.7393 with a standard deviation of 0.0102
The mean F1 is 0.7213 with a standard deviation of 0.0063
The mean AUC is 0.8226 with a standard deviation of 0.0037
Confusion Matrix:



ROC Curve:



The performance is a bit better than the previous models.

Performance Comparation

In the previous chapters, we measured the performance evaluation of each model, and I did some comparations. In the next table, there is a comparation between each model and the best value of each measurement remarked.

Model	Accuracy	Precision	Fallout	Recall	F1	AUC
Naive Bayes	0.7208	0.6598	0.3257	0.778	0.7141	0.7853
KNN	0.7375	0.698	0.255	0.7282	0.7127	0.8117
Decision Tree	0.7362	0.6951	0.2616	0.7334	0.7135	0.8131
SVM	0.7444	0.7041	0.2515	0.7393	0.7213	0.8226

As we can see, there is not a big difference in the performance of the models. However, Naïve Bayes seems to be the worst model since it has the worst fallout and precision by far. The best model is SVM since it has the best performance in almost all the measurements.

Additionally, I am going to use a tool from pycaret to illustrate how different models performance on this problem and compare it to the previous models. I will use google collab to avoid downloading all the packages on my machine.

First, we install pycaret and dependences:

```
! pip install fastapi
! pip install python-multipart
! pip install uvicorn
! pip install -q git+https://github.com/pycaret/pycaret.git
```

Import the used libraries and read the dataset (I using a previously balanced dataset for this task).

```
import pandas as pd
import numpy as np
from pycaret.classification import *
```

```
df = pd.read_csv("/content/diabetes_binary_5050split_health_indicators_BRFSS2015.csv")
```

Let's create the models:

```
setup_classification = setup(data=df, target='Diabetes_binary')
best_models = compare_models(n_select=5)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lightgbm	Light Gradient Boosting Machine	0.7543	0.8312	0.8009	0.7327	0.7653	0.5087	0.5109	21.060
gbc	Gradient Boosting Classifier	0.7539	0.8319	0.7960	0.7343	0.7639	0.5079	0.5097	31.980
ada	Ada Boost Classifier	0.7518	0.8292	0.7751	0.7406	0.7574	0.5036	0.5042	10.660
lr	Logistic Regression	0.7485	0.8257	0.7684	0.7391	0.7534	0.4970	0.4975	13.000
xgboost	Extreme Gradient Boosting	0.7476	0.8239	0.7911	0.7278	0.7581	0.4952	0.4972	0.5580
ridge	Ridge Classifier	0.7475	0.0000	0.7763	0.7340	0.7545	0.4949	0.4958	0.0520
lda	Linear Discriminant Analysis	0.7475	0.8249	0.7763	0.7340	0.7545	0.4949	0.4958	0.1370
rf	Random Forest Classifier	0.7399	0.8120	0.7806	0.7219	0.7501	0.4798	0.4814	33.400
qda	Quadratic Discriminant Analysis	0.7308	0.7845	0.7859	0.7080	0.7449	0.4616	0.4645	0.0700
et	Extra Trees Classifier	0.7278	0.7933	0.7673	0.7112	0.7381	0.4556	0.4571	36.970
svm	SVM - Linear Kernel	0.7208	0.8121	0.8411	0.6893	0.7479	0.4416	0.4721	0.9700
nb	Naive Bayes	0.7206	0.7879	0.7094	0.7256	0.7174	0.4411	0.4412	0.0550
knn	K Neighbors Classifier	0.7041	0.7611	0.7263	0.6954	0.7105	0.4082	0.4086	18.640
dt	Decision Tree Classifier	0.6583	0.6582	0.6506	0.6608	0.6556	0.3165	0.3166	0.1760
dummy	Dummy Classifier	0.5000	0.5000	0.2000	0.1000	0.1333	0.0000	0.0000	0.0440

Like we saw in the previous comparation, the performance do not vary a lot between the models. The performance measurements are roughly the same as in the previous models. The only difference is that the Decision Tree Classifier had a worse performance than Naïve Bayes Classifier

We get the best model:

```
best_tune = tune_model(best_models[0])
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8660	0.8300	0.1742	0.5619	0.2660	0.2141	0.2593
1	0.8654	0.8295	0.1601	0.5593	0.2489	0.1993	0.2472
2	0.8660	0.8241	0.1653	0.5657	0.2559	0.2058	0.2537

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
3	0.8682	0.8313	0.1851	0.5849	0.2812	0.2296	0.2764
4	0.8679	0.8318	0.1802	0.5853	0.2756	0.2247	0.2726
5	0.8659	0.8290	0.1628	0.5660	0.2529	0.2033	0.2518
6	0.8659	0.8279	0.1738	0.5599	0.2653	0.2133	0.2582
7	0.8648	0.8227	0.1710	0.5472	0.2605	0.2080	0.2513
8	0.8641	0.8286	0.1677	0.5390	0.2559	0.2032	0.2457
9	0.8652	0.8310	0.1734	0.5521	0.2639	0.2114	0.2550
Mean	0.8659	0.8286	0.1714	0.5621	0.2626	0.2113	0.2571
Std	0.0012	0.0029	0.0073	0.0140	0.0096	0.0092	0.0096

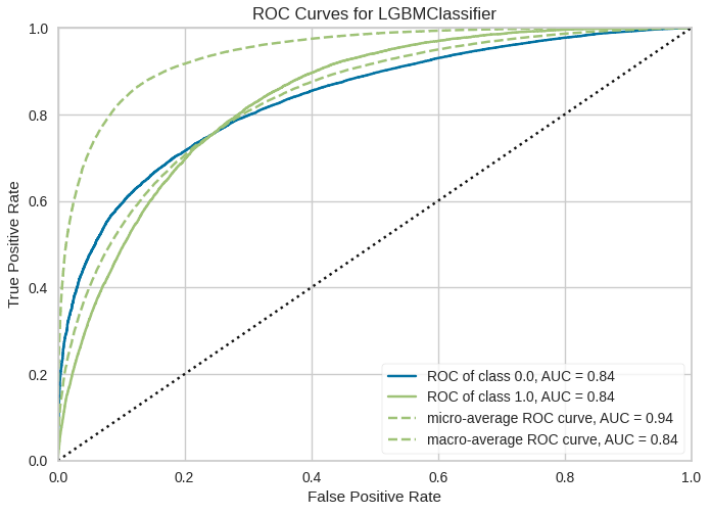
We can see the performance on the 10 folds and the mean of the performance evaluators.

To finalize, let's see the best model and the ROC Curve of it:

```
final_model = finalize_model(best_tune)
final model
```

```
LGBMClassifier(bagging_fraction=0.9,                bagging_freq=2,
feature_fraction=0.9,
learning_rate=0.05,
min_child_samples=51,
min_split_gain=0.6,
n_estimators=300,
n_jobs=-1,
num_leaves=20,
random_state=1353,
reg_alpha=1e-07,
reg_lambda=0.3)
```

```
plot_model(final model)
```



This tool allows us to obtain a great model and compare all types of models. However, you have to be careful and check if it works correctly.

Conclusions

In this assignment, we have used several models in order to classify the clinical and survey data of our problem. On top of that, I had to overcome many issues like the unbalance of the data or the fall of the UCI Machine Learning Repository servers. We have noticed that all models have a similar performance in the problem if they are correctly optimized, but not all the models are equally easy to optimize. The model that has a better performance independently of the hyperparameters is k-nearest neighbor, because it is very consistent with at least 20 neighbors. Another important point is the time complexity. Support Vector Machine has a very big time complexity in comparation to the other models. It may make the difference at the time of choosing the model. This is why there are some methods like Naïve Bayes that are used a lot for doing a first approach to the problem since it is computationally light.

The reason why there are many different classificators is because none of them is perfect, we have to use the correct method to achieve our purpose.