Pima Indians Diabetes Database

Context

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Content

The datasets consists of several medical predictor variables and one target variable, Outcome. Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

Pipelines In SkLearn

In [91]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
```

In [92]:
```
# Suppressing Warnings
import warnings
warnings.filterwarnings('ignore')
```

In [93]:
```
# Importing Pandas and NumPy
import pandas as pd, numpy as np
```

In [94]:
```
# Importing all datasets
diabetes = pd.read_csv("C:/Users/HP/Desktop/Upgrad Case Study/Pima_Diabetes/diabetes.csv")
diabetes.head(4)
```

Out[94]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0

In [95]:
```
diabetes.dtypes
```

Out[95]:
```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

In [96]:
```
diabetes.shape
```

Out[96]:
```
(768, 9)
```

In [97]:
```
diabetes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
```

```
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [98]:
```
diabetes.columns
```

Out[98]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

Rescaling the Features

We will use MinMax scaling.

In [99]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

In [100…
```
# Apply scaler() to all the columns except the 'yes-no' and 'dummy' variables
num_vars = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI", "DiabetesPedigreeFunction", "Age", "Outc

diabetes[num_vars] = scaler.fit_transform(diabetes[num_vars])

diabetes.head()
```

Out[100…

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	0.352941	0.743719	0.590164	0.353535	0.000000	0.500745	0.234415	0.483333	1.0
1	0.058824	0.427136	0.540984	0.292929	0.000000	0.396423	0.116567	0.166667	0.0
2	0.470588	0.919598	0.524590	0.000000	0.000000	0.347243	0.253629	0.183333	1.0
3	0.058824	0.447236	0.540984	0.232323	0.111111	0.418778	0.038002	0.000000	0.0
4	0.000000	0.688442	0.327869	0.353535	0.198582	0.642325	0.943638	0.200000	1.0

Checking For `Any Null Values`

In [101…
```
diabetes.isnull().sum()
```

Out[101…
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

- **Here, No missing value imputation is required.**

Checking for Outliers

In [102…
```
# Checking for outliers in the continuous variables
num_diabetes = diabetes[["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI", "DiabetesPedigreeFunction",
```

In [103…
```
# Checking outliers at 25%, 50%, 75%, 90%, 95% and 99%
num_diabetes.describe(percentiles=[.25, .5, .75, .90, .95, .99])
```

Out[103…

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	0.226180	0.607510	0.566438	0.207439	0.094326	0.476790	0.168179	0.204015	0.348958
std	0.198210	0.160666	0.158654	0.161134	0.136222	0.117499	0.141473	0.196004	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.058824	0.497487	0.508197	0.000000	0.000000	0.406855	0.070773	0.050000	0.000000
50%	0.176471	0.587940	0.590164	0.232323	0.036052	0.476900	0.125747	0.133333	0.000000
75%	0.352941	0.704774	0.655738	0.323232	0.150414	0.545455	0.234095	0.333333	1.000000
90%	0.529412	0.839196	0.721311	0.404040	0.248227	0.618480	0.341845	0.500000	1.000000
95%	0.588235	0.909548	0.737705	0.444444	0.346336	0.661624	0.450406	0.616667	1.000000
99%	0.764706	0.984925	0.868852	0.518485	0.614539	0.756468	0.691857	0.766667	1.000000

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

In [104…

```
Q1 = num_diabetes.quantile(0.25)
Q3 = num_diabetes.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
Pregnancies                 0.294118
Glucose                     0.207286
BloodPressure               0.147541
SkinThickness               0.323232
Insulin                     0.150414
BMI                         0.138599
DiabetesPedigreeFunction    0.163322
Age                         0.283333
Outcome                     1.000000
dtype: float64
```

Quantile-based Flooring and Capping

In [105…

```
print(diabetes.quantile(0.05))
print(diabetes.quantile(0.95))
```

```
Pregnancies                 0.000000
Glucose                     0.396985
BloodPressure               0.317213
SkinThickness               0.000000
Insulin                     0.000000
BMI                         0.324888
DiabetesPedigreeFunction    0.026623
Age                         0.000000
Outcome                     0.000000
Name: 0.05, dtype: float64
Pregnancies                 0.588235
Glucose                     0.909548
BloodPressure               0.737705
SkinThickness               0.444444
Insulin                     0.346336
BMI                         0.661624
DiabetesPedigreeFunction    0.450406
Age                         0.616667
```

```
Outcome                     1.000000
Name: 0.95, dtype: float64
```

In [106…]
```
diabetes = diabetes[~((diabetes < (Q1 - 1.5 * IQR)) | (diabetes > (Q3 + 1.5 * IQR))).any(axis=1)]
print(diabetes.shape)
```

```
(639, 9)
```

Test-Train Split

In [107…]
```
from sklearn.model_selection import train_test_split
```

In [108…]
```
diabetes.head(3)
```

Out[108…]

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	0.352941	0.743719	0.590164	0.353535	0.0	0.500745	0.234415	0.483333	1.0
1	0.058824	0.427136	0.540984	0.292929	0.0	0.396423	0.116567	0.166667	0.0
2	0.470588	0.919598	0.524590	0.000000	0.0	0.347243	0.253629	0.183333	1.0

In [109…]
```
# Putting feature variable to X
X = diabetes.drop(['Outcome'], axis=1)
X.head()
```

Out[109…]

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.352941	0.743719	0.590164	0.353535	0.000000	0.500745	0.234415	0.483333
1	0.058824	0.427136	0.540984	0.292929	0.000000	0.396423	0.116567	0.166667
2	0.470588	0.919598	0.524590	0.000000	0.000000	0.347243	0.253629	0.183333
3	0.058824	0.447236	0.540984	0.232323	0.111111	0.418778	0.038002	0.000000
5	0.294118	0.582915	0.606557	0.000000	0.000000	0.381520	0.052519	0.150000

In [110…]
```
# Putting response variable to y
```

```
y = diabetes['Outcome']
y.head()
```

Out[110…
```
0    1.0
1    0.0
2    1.0
3    0.0
5    0.0
Name: Outcome, dtype: float64
```

In [111…
```
# Splitting the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, test_size=0.3, random_state=100)
```

In [112…
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(447, 8)
(192, 8)
(447,)
(192,)
```

Pipeline Creation

In [113…
```
## Pipelines Creation
## 1. Data Preprocessing by using Standard Scaler
## 2. Reduce Dimension using PCA
## 3. Apply  Classifier
```

In [114…
```
pipeline_lr=Pipeline([('scalar1',StandardScaler()),
                      ('pca1',PCA(n_components=2)),
                      ('lr_classifier',LogisticRegression(random_state=0))])
```

In [115…
```
pipeline_dt=Pipeline([('scalar2',StandardScaler()),
                      ('pca2',PCA(n_components=2)),
                      ('dt_classifier',DecisionTreeClassifier())])
```

In [116…
```
pipeline_randomforest=Pipeline([('scalar3',StandardScaler()),
                    ('pca3',PCA(n_components=2)),
                    ('rf_classifier',RandomForestClassifier())])
```

In [117…
```
pipeline_xgbclassifier=Pipeline([('scalar3',StandardScaler()),
                    ('pca3',PCA(n_components=2)),
                    ('rf_classifier',XGBClassifier())])
```

In [118…
```
## LEts make the list of pipelines
pipelines = [pipeline_lr, pipeline_dt, pipeline_randomforest, pipeline_xgbclassifier]
```

In [119…
```
best_accuracy=0.0
best_classifier=0
best_pipeline=""
```

In [120…
```
# Dictionary of pipelines and classifier types for ease of reference
pipe_dict = {0: 'Logistic Regression', 1: 'Decision Tree', 2: 'RandomForest', 3: 'XGBClassifier'}

# Fit the pipelines
for pipe in pipelines:
        pipe.fit(X_train, y_train)
```

```
[16:29:16] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, t
he default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_
metric if you'd like to restore the old behavior.
```

In [121…
```
for i,model in enumerate(pipelines):
    print("{} Test Accuracy: {}".format(pipe_dict[i],model.score(X_test,y_test)))
```

```
Logistic Regression Test Accuracy: 0.7135416666666666
Decision Tree Test Accuracy: 0.7552083333333334
RandomForest Test Accuracy: 0.703125
XGBClassifier Test Accuracy: 0.7083333333333334
```

In [122…
```
for i,model in enumerate(pipelines):
    if model.score(X_test,y_test)>best_accuracy:
        best_accuracy=model.score(X_test,y_test)
        best_pipeline=model
```

```
        best_classifier=i
print('Classifier with best accuracy:{}'.format(pipe_dict[best_classifier]))
```

```
Classifier with best accuracy:Decision Tree
```

In [123…

```
from sklearn.tree import DecisionTreeClassifier
dt_classifier=DecisionTreeClassifier(random_state=0).fit(X_train,y_train)
prediction=dt_classifier.predict(X_test)
```

In [124…

```
y_test.value_counts()
```

Out[124…

```
0.0    129
1.0     63
Name: Outcome, dtype: int64
```

In [125…

```
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
print(confusion_matrix(y_test,prediction))
print(accuracy_score(y_test,prediction))
print(classification_report(y_test,prediction))
```

```
[[109  20]
 [ 32  31]]
0.7291666666666666
              precision    recall  f1-score   support

         0.0       0.77      0.84      0.81       129
         1.0       0.61      0.49      0.54        63

    accuracy                           0.73       192
   macro avg       0.69      0.67      0.68       192
weighted avg       0.72      0.73      0.72       192
```

In [126…

```
### Hyper Parameter Tuning
```

In [127…

```
### Manual Hyperparameter Tuning
model=DecisionTreeClassifier(criterion='entropy',
                             max_features='sqrt',min_samples_leaf=10,random_state=100).fit(X_train,y_train)
predictions=model.predict(X_test)
print(confusion_matrix(y_test,predictions))
```

```
print(accuracy_score(y_test,predictions))
print(classification_report(y_test,predictions))
```

```
[[115  14]
 [ 35  28]]
0.7447916666666666
              precision    recall  f1-score   support

         0.0       0.77      0.89      0.82       129
         1.0       0.67      0.44      0.53        63

    accuracy                           0.74       192
   macro avg       0.72      0.67      0.68       192
weighted avg       0.73      0.74      0.73       192
```

Randomized Search Cv

In [128…

```
import numpy as np
from sklearn.model_selection import RandomizedSearchCV
# Number of features to consider at every split
max_features = ['auto', 'sqrt','log2']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 1000,10)]
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10,14]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4,6,8]
# Create the random grid
random_grid = {'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
              'criterion':['entropy','gini']}
print(random_grid)
```

```
{'max_features': ['auto', 'sqrt', 'log2'], 'max_depth': [10, 120, 230, 340, 450, 560, 670, 780, 890, 1000], 'min_samples_split':
[2, 5, 10, 14], 'min_samples_leaf': [1, 2, 4, 6, 8], 'criterion': ['entropy', 'gini']}
```

In [129…

```
rf=RandomForestClassifier()
rf_randomcv=RandomizedSearchCV(estimator=rf,param_distributions=random_grid,n_iter=100,cv=3,verbose=2,
                               random_state=100,n_jobs=-1)
### fit the randomized model
rf_randomcv.fit(X_train,y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

Out[129…
```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(), n_iter=100,
                   n_jobs=-1,
                   param_distributions={'criterion': ['entropy', 'gini'],
                                        'max_depth': [10, 120, 230, 340, 450,
                                                      560, 670, 780, 890,
                                                      1000],
                                        'max_features': ['auto', 'sqrt',
                                                         'log2'],
                                        'min_samples_leaf': [1, 2, 4, 6, 8],
                                        'min_samples_split': [2, 5, 10, 14]},
                   random_state=100, verbose=2)
```

In [130…
```
rf_randomcv.best_params_
```

Out[130…
```
{'min_samples_split': 10,
 'min_samples_leaf': 1,
 'max_features': 'sqrt',
 'max_depth': 670,
 'criterion': 'entropy'}
```

In [131…
```
rf_randomcv
```

Out[131…
```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(), n_iter=100,
                   n_jobs=-1,
                   param_distributions={'criterion': ['entropy', 'gini'],
                                        'max_depth': [10, 120, 230, 340, 450,
                                                      560, 670, 780, 890,
                                                      1000],
                                        'max_features': ['auto', 'sqrt',
                                                         'log2'],
                                        'min_samples_leaf': [1, 2, 4, 6, 8],
                                        'min_samples_split': [2, 5, 10, 14]},
                   random_state=100, verbose=2)
```

In [132…
```
best_random_grid=rf_randomcv.best_estimator_
```

In [133…
```
from sklearn.metrics import accuracy_score
y_pred=best_random_grid.predict(X_test)
print(confusion_matrix(y_test,y_pred))
```

```
print("Accuracy Score {}".format(accuracy_score(y_test,y_pred)))
print("Classification report: {}".format(classification_report(y_test,y_pred)))
```

```
[[120   9]
 [ 34  29]]
Accuracy Score 0.7760416666666666
Classification report:               precision    recall  f1-score   support

         0.0       0.78      0.93      0.85       129
         1.0       0.76      0.46      0.57        63

    accuracy                           0.78       192
   macro avg       0.77      0.70      0.71       192
weighted avg       0.77      0.78      0.76       192
```

GridSearch CV

In [134…

```
rf_randomcv.best_params_
```

Out[134…

```
{'min_samples_split': 10,
 'min_samples_leaf': 1,
 'max_features': 'sqrt',
 'max_depth': 670,
 'criterion': 'entropy'}
```

In [135…

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'criterion': [rf_randomcv.best_params_['criterion']],
    'max_depth': [rf_randomcv.best_params_['max_depth']],
    'max_features': [rf_randomcv.best_params_['max_features']],
    'min_samples_leaf': [rf_randomcv.best_params_['min_samples_leaf'],
                         rf_randomcv.best_params_['min_samples_leaf']+2,
                         rf_randomcv.best_params_['min_samples_leaf'] + 4],
    'min_samples_split': [rf_randomcv.best_params_['min_samples_split'] - 2,
                          rf_randomcv.best_params_['min_samples_split'] - 1,
                          rf_randomcv.best_params_['min_samples_split'],
                          rf_randomcv.best_params_['min_samples_split'] +1,
                          rf_randomcv.best_params_['min_samples_split'] + 2]
}

print(param_grid)
```

```
{'criterion': ['entropy'], 'max_depth': [670], 'max_features': ['sqrt'], 'min_samples_leaf': [1, 3, 5], 'min_samples_split': [8,
9, 10, 11, 12]}
```

In [136…
```
#### Fit the grid_search to the data
dt=DecisionTreeClassifier()
grid_search=GridSearchCV(estimator=dt,param_grid=param_grid,cv=10,n_jobs=-1,verbose=2)
grid_search.fit(X_train,y_train)
```

```
Fitting 10 folds for each of 15 candidates, totalling 150 fits
```

Out[136…
```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'criterion': ['entropy'], 'max_depth': [670],
                         'max_features': ['sqrt'],
                         'min_samples_leaf': [1, 3, 5],
                         'min_samples_split': [8, 9, 10, 11, 12]},
             verbose=2)
```

In [137…
```
grid_search.best_estimator_
```

Out[137…
```
DecisionTreeClassifier(criterion='entropy', max_depth=670, max_features='sqrt',
                       min_samples_leaf=5, min_samples_split=8)
```

In [138…
```
best_grid=grid_search.best_estimator_
```

In [139…
```
best_grid
```

Out[139…
```
DecisionTreeClassifier(criterion='entropy', max_depth=670, max_features='sqrt',
                       min_samples_leaf=5, min_samples_split=8)
```

In [140…
```
y_pred=best_grid.predict(X_test)
print(confusion_matrix(y_test,y_pred))
print("Accuracy Score {}".format(accuracy_score(y_test,y_pred)))
print("Classification report: {}".format(classification_report(y_test,y_pred)))
```

```
[[102  27]
 [ 33  30]]
Accuracy Score 0.6875
Classification report:               precision    recall  f1-score   support

         0.0       0.76      0.79      0.77       129
         1.0       0.53      0.48      0.50        63
```

```
              accuracy                           0.69       192
             macro avg       0.64      0.63      0.64       192
          weighted avg       0.68      0.69      0.68       192
```

Automated Hyperparameter Tuning

Automated Hyperparameter Tuning can be done by using techniques such as

- **Bayesian Optimization**
- **Gradient Descent**
- **Evolutionary Algorithms**

Bayesian Optimization

Bayesian optimization uses probability to find the minimum of a function. The final aim is to find the input value to a function which can gives us the lowest possible output value.It usually performs better than random,grid and manual search providing better performance in the testing phase and reduced optimization time. In Hyperopt, Bayesian Optimization can be implemented giving 3 three main parameters to the function fmin.

- **Objective Function = defines the loss function to minimize.**
- **Domain Space = defines the range of input values to test (in Bayesian Optimization this space creates a probability distribution for each of the used Hyperparameters).**
- **Optimization Algorithm = defines the search algorithm to use to select the best input values to use in each new iteration.**

In [152…

```
from hyperopt import hp,fmin,tpe,STATUS_OK,Trials
```

In [159…

```
space = {'criterion': hp.choice('criterion', ['entropy', 'gini']),
        'max_depth': hp.quniform('max_depth', 10, 1200, 10),
        'max_features': hp.choice('max_features', ['auto', 'sqrt','log2', None]),
        'min_samples_leaf': hp.uniform('min_samples_leaf', 0, 0.5),
        'min_samples_split' : hp.uniform ('min_samples_split', 0, 1)
    }
```

In [160…

```
space
```

Out[160…
```
{'criterion': <hyperopt.pyll.base.Apply at 0x1e0fe8310d0>,
 'max_depth': <hyperopt.pyll.base.Apply at 0x1e0fe811ee0>,
 'max_features': <hyperopt.pyll.base.Apply at 0x1e0fe8113d0>,
 'min_samples_leaf': <hyperopt.pyll.base.Apply at 0x1e0fe811250>,
 'min_samples_split': <hyperopt.pyll.base.Apply at 0x1e0fe811fa0>}
```

In [161…
```
def objective(space):
    model = DecisionTreeClassifier(max_depth = space['max_depth'],
                                   max_features = space['max_features'],
                                   min_samples_leaf = space['min_samples_leaf'],
                                   min_samples_split = space['min_samples_split']
                                   )

    accuracy = cross_val_score(model, X_train, y_train, cv = 5).mean()

    # We aim to maximize accuracy, therefore we return it as a negative value
    return {'loss': -accuracy, 'status': STATUS_OK }
```

In [162…
```
from sklearn.model_selection import cross_val_score
trials = Trials()
best = fmin(fn= objective,
            space= space,
            algo= tpe.suggest,
            max_evals = 80,
            trials= trials)
best
```

```
100%|██████████| 80/80 [00:03<00:00, 20.73trial/s, best loss: -0.7738077403245942]
```

Out[162…
```
{'criterion': 0,
 'max_depth': 430.0,
 'max_features': 3,
 'min_samples_leaf': 0.19622205767178413,
 'min_samples_split': 0.36714887797691664}
```

In [163…
```
crit = {0: 'entropy', 1: 'gini'}
feat = {0: 'auto', 1: 'sqrt', 2: 'log2', 3: None}


print(crit[best['criterion']])
print(feat[best['max_features']])
```

```
entropy
```

```
None
```

In [164…
```
best['min_samples_leaf']
```

Out[164…
```
0.19622205767178413
```

In [165…
```
trainedforest = DecisionTreeClassifier(criterion = crit[best['criterion']], max_depth = best['max_depth'],
                                       max_features = feat[best['max_features']],
                                       min_samples_leaf = best['min_samples_leaf'],
                                       min_samples_split = best['min_samples_split']).fit(X_train,y_train)
predictionforest = trainedforest.predict(X_test)
print(confusion_matrix(y_test,predictionforest))
print(accuracy_score(y_test,predictionforest))
print(classification_report(y_test,predictionforest))
acc5 = accuracy_score(y_test,predictionforest)
```

```
[[119  10]
 [ 39  24]]
0.7447916666666666
              precision    recall  f1-score   support

         0.0       0.75      0.92      0.83       129
         1.0       0.71      0.38      0.49        63

    accuracy                           0.74       192
   macro avg       0.73      0.65      0.66       192
weighted avg       0.74      0.74      0.72       192
```

Genetic Algorithms

Genetic Algorithms tries to apply natural selection mechanisms to Machine Learning contexts.

Let's immagine we create a population of N Machine Learning models with some predifined Hyperparameters. We can then calculate the accuracy of each model and decide to keep just half of the models (the ones that performs best). We can now generate some offsprings having similar Hyperparameters to the ones of the best models so that go get again a population of N models. At this point we can again caltulate the accuracy of each model and repeate the cycle for a defined number of generations. In this way, just the best models will survive at the end of the process.

In [166…
```
import numpy as np
```

```
from sklearn.model_selection import RandomizedSearchCV

# Number of features to consider at every split
max_features = ['auto', 'sqrt','log2']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 1000,10)]
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10,14]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4,6,8]
# Create the random grid
param = {
                'max_features': max_features,
                'max_depth': max_depth,
                'min_samples_split': min_samples_split,
                'min_samples_leaf': min_samples_leaf,
               'criterion':['entropy','gini']}
print(param)
```

```
{'max_features': ['auto', 'sqrt', 'log2'], 'max_depth': [10, 120, 230, 340, 450, 560, 670, 780, 890, 1000], 'min_samples_split':
[2, 5, 10, 14], 'min_samples_leaf': [1, 2, 4, 6, 8], 'criterion': ['entropy', 'gini']}
```

In [167…

```
from tpot import TPOTClassifier


tpot_classifier = TPOTClassifier(generations= 5, population_size= 24, offspring_size= 12,
                                 verbosity= 2, early_stop= 12,
                                 config_dict={'sklearn.ensemble.RandomForestClassifier': param},
                                 cv = 4, scoring = 'accuracy')
tpot_classifier.fit(X_train,y_train)
```

```
Generation 1 - Current best internal CV score: 0.7874839124839125

Generation 2 - Current best internal CV score: 0.7874839124839125

Generation 3 - Current best internal CV score: 0.7874839124839125

Generation 4 - Current best internal CV score: 0.7874839124839125

Generation 5 - Current best internal CV score: 0.7874839124839125

Best pipeline: RandomForestClassifier(input_matrix, criterion=entropy, max_depth=340, max_features=log2, min_samples_leaf=8, min_s
amples_split=14)
```

Out[167…
```
TPOTClassifier(config_dict={'sklearn.ensemble.RandomForestClassifier': {'criterion': ['entropy',
                                                                                      'gini'],
                                                                        'max_depth': [10,
                                                                                      120,
                                                                                      230,
                                                                                      340,
                                                                                      450,
                                                                                      560,
                                                                                      670,
                                                                                      780,
                                                                                      890,
                                                                                      1000],
                                                                        'max_features': ['auto',
                                                                                         'sqrt',
                                                                                         'log2'],
                                                                        'min_samples_leaf': [1,
                                                                                             2,
                                                                                             4,
                                                                                             6,
                                                                                             8],
                                                                        'min_samples_split': [2,
                                                                                              5,
                                                                                              10,
                                                                                              14]}},
               cv=4, early_stop=12, generations=5, offspring_size=12,
               population_size=24, scoring='accuracy', verbosity=2)
```

In [168…
```
accuracy = tpot_classifier.score(X_test, y_test)
print(accuracy)
```

```
0.765625
```

Optimize hyperparameters of the model using Optuna

The hyperparameters of the above algorithm are `n_estimators` and `max_depth` for which we can try different values to see if the model accuracy can be improved. The `objective` function is modified to accept a trial object. This trial has several methods for sampling hyperparameters. We create a study to run the hyperparameter optimization and finally read the best hyperparameters.

In [173…
```
import optuna
import sklearn.svm
def objective(trial):

    classifier = trial.suggest_categorical('classifier', ['DecisionTree', 'SVC'])
```

```
    if classifier == 'DecisionTree':
        n_estimators = trial.suggest_int('n_estimators', 200, 2000,10)
        max_depth = int(trial.suggest_float('max_depth', 10, 100, log=True))

        clf = sklearn.tree.DecisionTreeClassifier(
            max_depth=max_depth)
    else:
        c = trial.suggest_float('svc_c', 1e-10, 1e10, log=True)

        clf = sklearn.svm.SVC(C=c, gamma='auto')

    return sklearn.model_selection.cross_val_score(
        clf,X_train,y_train, n_jobs=-1, cv=3).mean()
```

In [174…

```
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)

trial = study.best_trial

print('Accuracy: {}'.format(trial.value))
print("Best hyperparameters: {}".format(trial.params))
```

```
[I 2021-10-11 17:31:55,369] A new study created in memory with name: no-name-a960356a-222c-4e0b-9c32-056d90a89f73
[I 2021-10-11 17:31:55,464] Trial 0 finished with value: 0.7114093959731543 and parameters: {'classifier': 'DecisionTree', 'n_esti
mators': 1790, 'max_depth': 13.389165448326867}. Best is trial 0 with value: 0.7114093959731543.
[I 2021-10-11 17:31:55,498] Trial 1 finished with value: 0.7897091722595079 and parameters: {'classifier': 'SVC', 'svc_c': 3.20293
61679282657}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:33:35,687] Trial 2 finished with value: 0.6823266219239373 and parameters: {'classifier': 'SVC', 'svc_c': 3442384
4.66282742}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:33:35,757] Trial 3 finished with value: 0.70917225950783 and parameters: {'classifier': 'DecisionTree', 'n_estima
tors': 300, 'max_depth': 11.864654884529658}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:33:35,779] Trial 4 finished with value: 0.7046979865771812 and parameters: {'classifier': 'DecisionTree', 'n_esti
mators': 340, 'max_depth': 10.06714979937242}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:33:38,246] Trial 5 finished with value: 0.7516778523489934 and parameters: {'classifier': 'SVC', 'svc_c': 323486.
18641983654}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:33:38,281] Trial 6 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 1.48584
98534900923e-06}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:33:38,306] Trial 7 finished with value: 0.7069351230425056 and parameters: {'classifier': 'DecisionTree', 'n_esti
mators': 1560, 'max_depth': 22.225619840182937}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:33:38,344] Trial 8 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 1.10350
00223983762e-07}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:33:38,377] Trial 9 finished with value: 0.7807606263982102 and parameters: {'classifier': 'SVC', 'svc_c': 152.464
01753065604}. Best is trial 1 with value: 0.7897091722595079.
```

```
[I 2021-10-11 17:33:38,416] Trial 10 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 0.0457
2420022946995}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:33:38,454] Trial 11 finished with value: 0.7718120805369129 and parameters: {'classifier': 'SVC', 'svc_c': 57.650
97882555224}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:33:38,534] Trial 12 finished with value: 0.7651006711409396 and parameters: {'classifier': 'SVC', 'svc_c': 2.1453
440497981813}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:33:38,632] Trial 13 finished with value: 0.7718120805369129 and parameters: {'classifier': 'SVC', 'svc_c': 5923.2
01279684798}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:33:38,669] Trial 14 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 0.0007
009750075004121}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:37:17,830] Trial 15 finished with value: 0.7337807606263982 and parameters: {'classifier': 'SVC', 'svc_c': 164205
7255.258291}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:37:17,891] Trial 16 finished with value: 0.7695749440715884 and parameters: {'classifier': 'SVC', 'svc_c': 425.61
439114116877}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:37:17,935] Trial 17 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 0.0026
619889114165365}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:37:26,499] Trial 18 finished with value: 0.7293064876957495 and parameters: {'classifier': 'SVC', 'svc_c': 153227
3.4336294588}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:37:26,537] Trial 19 finished with value: 0.7874720357941835 and parameters: {'classifier': 'SVC', 'svc_c': 2.5181
99688115628}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:37:26,578] Trial 20 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 5.7786
87545218239e-10}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:37:26,622] Trial 21 finished with value: 0.7583892617449663 and parameters: {'classifier': 'SVC', 'svc_c': 1.9373
444621884457}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:37:26,797] Trial 22 finished with value: 0.7762863534675616 and parameters: {'classifier': 'SVC', 'svc_c': 10021.
678786998353}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:37:26,839] Trial 23 finished with value: 0.7718120805369129 and parameters: {'classifier': 'SVC', 'svc_c': 15.127
21460164932}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:37:26,890] Trial 24 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 0.0117
95970765812794}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:37:26,928] Trial 25 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 3.6891
8077312476e-05}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:37:26,965] Trial 26 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 0.2209
9219392837322}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:37:26,999] Trial 27 finished with value: 0.6845637583892618 and parameters: {'classifier': 'DecisionTree', 'n_est
imators': 1000, 'max_depth': 87.84559642225105}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:37:27,046] Trial 28 finished with value: 0.7718120805369127 and parameters: {'classifier': 'SVC', 'svc_c': 629.58
34536466872}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:37:27,081] Trial 29 finished with value: 0.7024608501118568 and parameters: {'classifier': 'DecisionTree', 'n_est
imators': 950, 'max_depth': 92.70365024417966}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:37:27,528] Trial 30 finished with value: 0.7695749440715884 and parameters: {'classifier': 'SVC', 'svc_c': 51084.
0210704227}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:37:27,610] Trial 31 finished with value: 0.7651006711409396 and parameters: {'classifier': 'SVC', 'svc_c': 4019.0
52930099931}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:37:27,651] Trial 32 finished with value: 0.7718120805369129 and parameters: {'classifier': 'SVC', 'svc_c': 44.405
396260306105}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:38:02,354] Trial 33 finished with value: 0.7024608501118568 and parameters: {'classifier': 'SVC', 'svc_c': 358337
```

```
9.8088415135}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:38:02,537] Trial 34 finished with value: 0.7740492170022372 and parameters: {'classifier': 'SVC', 'svc_c': 21744.
84119772389}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:38:02,575] Trial 35 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 0.8448
432315174956}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:38:02,613] Trial 36 finished with value: 0.7046979865771812 and parameters: {'classifier': 'DecisionTree', 'n_est
imators': 1380, 'max_depth': 43.84263251018664}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:39:53,684] Trial 37 finished with value: 0.7002237136465325 and parameters: {'classifier': 'SVC', 'svc_c': 140750
838.87687635}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:39:53,724] Trial 38 finished with value: 0.767337807606264 and parameters: {'classifier': 'SVC', 'svc_c': 314.549
7062724049}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:39:53,759] Trial 39 finished with value: 0.7024608501118568 and parameters: {'classifier': 'DecisionTree', 'n_est
imators': 750, 'max_depth': 45.7104177311094}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:39:54,509] Trial 40 finished with value: 0.767337807606264 and parameters: {'classifier': 'SVC', 'svc_c': 103185.
7072979429}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:39:54,565] Trial 41 finished with value: 0.7651006711409396 and parameters: {'classifier': 'SVC', 'svc_c': 2275.9
199454753625}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:39:54,783] Trial 42 finished with value: 0.7740492170022372 and parameters: {'classifier': 'SVC', 'svc_c': 23705.
075736253555}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:01,718] Trial 43 finished with value: 0.7315436241610738 and parameters: {'classifier': 'SVC', 'svc_c': 136299
8.2871137443}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:01,756] Trial 44 finished with value: 0.7718120805369129 and parameters: {'classifier': 'SVC', 'svc_c': 14.856
500760625554}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:01,798] Trial 45 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 0.1787
9307563945257}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:01,838] Trial 46 finished with value: 0.7762863534675616 and parameters: {'classifier': 'SVC', 'svc_c': 68.617
49594662392}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:01,881] Trial 47 finished with value: 0.785234899328859 and parameters: {'classifier': 'SVC', 'svc_c': 8.55845
2913579394}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:01,925] Trial 48 finished with value: 0.7829977628635346 and parameters: {'classifier': 'SVC', 'svc_c': 6.0845
55776153658}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:01,975] Trial 49 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 0.0277
4003761873225}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:02,015] Trial 50 finished with value: 0.7807606263982102 and parameters: {'classifier': 'SVC', 'svc_c': 4.0630
97044733171}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:02,055] Trial 51 finished with value: 0.785234899328859 and parameters: {'classifier': 'SVC', 'svc_c': 2.75154
4389997308}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:02,102] Trial 52 finished with value: 0.7740492170022372 and parameters: {'classifier': 'SVC', 'svc_c': 4.5352
91288517453}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:02,142] Trial 53 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 0.0003
991635908366108}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:02,181] Trial 54 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 0.5108
145677812798}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:02,223] Trial 55 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 0.0584
81657953813784}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:02,266] Trial 56 finished with value: 0.7740492170022372 and parameters: {'classifier': 'SVC', 'svc_c': 4.8943
34019790844}. Best is trial 1 with value: 0.7897091722595079.
```

```
[I 2021-10-11 17:40:02,309] Trial 57 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 0.0106
0185712471628}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:02,349] Trial 58 finished with value: 0.7762863534675616 and parameters: {'classifier': 'SVC', 'svc_c': 255.16
243565325289}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:02,389] Trial 59 finished with value: 0.7807606263982102 and parameters: {'classifier': 'SVC', 'svc_c': 11.112
512055430507}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:02,428] Trial 60 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 0.0023
75059776597546}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:02,465] Trial 61 finished with value: 0.7718120805369129 and parameters: {'classifier': 'SVC', 'svc_c': 60.218
07499925354}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:02,509] Trial 62 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 0.6551
604425582954}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:02,561] Trial 63 finished with value: 0.7829977628635346 and parameters: {'classifier': 'SVC', 'svc_c': 9.1305
7637736985}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:02,613] Trial 64 finished with value: 0.7695749440715884 and parameters: {'classifier': 'SVC', 'svc_c': 1563.9
51805668613}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:02,655] Trial 65 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 0.1541
3144629759346}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:02,692] Trial 66 finished with value: 0.7718120805369129 and parameters: {'classifier': 'SVC', 'svc_c': 15.326
688279341186}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:02,727] Trial 67 finished with value: 0.7046979865771812 and parameters: {'classifier': 'DecisionTree', 'n_est
imators': 2000, 'max_depth': 22.92675486592533}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:02,767] Trial 68 finished with value: 0.7181208053691276 and parameters: {'classifier': 'SVC', 'svc_c': 1.2268
212052779115}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:02,804] Trial 69 finished with value: 0.785234899328859 and parameters: {'classifier': 'SVC', 'svc_c': 2.74098
8194313001}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:02,843] Trial 70 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 0.0795
7849251211468}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:02,890] Trial 71 finished with value: 0.7807606263982104 and parameters: {'classifier': 'SVC', 'svc_c': 116.29
819433776079}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:02,935] Trial 72 finished with value: 0.7718120805369129 and parameters: {'classifier': 'SVC', 'svc_c': 51.730
42933758526}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:02,984] Trial 73 finished with value: 0.7718120805369129 and parameters: {'classifier': 'SVC', 'svc_c': 508.93
75031136691}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:03,030] Trial 74 finished with value: 0.7472035794183446 and parameters: {'classifier': 'SVC', 'svc_c': 1.6988
256471592782}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:03,072] Trial 75 finished with value: 0.7807606263982104 and parameters: {'classifier': 'SVC', 'svc_c': 118.95
839093425694}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:03,109] Trial 76 finished with value: 0.7718120805369129 and parameters: {'classifier': 'SVC', 'svc_c': 12.128
25772482499}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:03,149] Trial 77 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 0.4052
9999284964896}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:03,190] Trial 78 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 0.0123
94201597317215}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:03,234] Trial 79 finished with value: 0.7695749440715884 and parameters: {'classifier': 'SVC', 'svc_c': 937.13
24029971748}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:03,263] Trial 80 finished with value: 0.7024608501118568 and parameters: {'classifier': 'DecisionTree', 'n_est
```

```
imators': 600, 'max_depth': 58.0608480569627}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:03,300] Trial 81 finished with value: 0.7829977628635346 and parameters: {'classifier': 'SVC', 'svc_c': 170.89
492999763922}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:03,339] Trial 82 finished with value: 0.7695749440715884 and parameters: {'classifier': 'SVC', 'svc_c': 19.378
410869437275}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:03,382] Trial 83 finished with value: 0.7874720357941835 and parameters: {'classifier': 'SVC', 'svc_c': 2.8239
276859380555}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:03,421] Trial 84 finished with value: 0.7874720357941835 and parameters: {'classifier': 'SVC', 'svc_c': 2.8222
01463372105}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:03,464] Trial 85 finished with value: 0.756152125279642 and parameters: {'classifier': 'SVC', 'svc_c': 1.92506
96876184268}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:03,505] Trial 86 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 0.2308
635450740855}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:03,547] Trial 87 finished with value: 0.7807606263982102 and parameters: {'classifier': 'SVC', 'svc_c': 4.0539
76887800732}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:03,589] Trial 88 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 0.0325
13480576063354}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:03,625] Trial 89 finished with value: 0.7695749440715884 and parameters: {'classifier': 'SVC', 'svc_c': 33.733
40748991421}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:03,665] Trial 90 finished with value: 0.7829977628635346 and parameters: {'classifier': 'SVC', 'svc_c': 6.4103
71303455362}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:03,704] Trial 91 finished with value: 0.7807606263982102 and parameters: {'classifier': 'SVC', 'svc_c': 5.4403
94875610009}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:03,744] Trial 92 finished with value: 0.7404921700223714 and parameters: {'classifier': 'SVC', 'svc_c': 1.5055
012356103585}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:03,781] Trial 93 finished with value: 0.7762863534675616 and parameters: {'classifier': 'SVC', 'svc_c': 238.30
682590171824}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:03,823] Trial 94 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 0.5040
950241857691}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:03,862] Trial 95 finished with value: 0.6935123042505594 and parameters: {'classifier': 'SVC', 'svc_c': 0.0895
4694781804302}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:03,905] Trial 96 finished with value: 0.7829977628635346 and parameters: {'classifier': 'SVC', 'svc_c': 7.4697
7521202966}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:04,002] Trial 97 finished with value: 0.7628635346756152 and parameters: {'classifier': 'SVC', 'svc_c': 4327.0
23679946644}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:04,045] Trial 98 finished with value: 0.7695749440715884 and parameters: {'classifier': 'SVC', 'svc_c': 36.062
70915789271}. Best is trial 1 with value: 0.7897091722595079.
[I 2021-10-11 17:40:04,089] Trial 99 finished with value: 0.7002237136465324 and parameters: {'classifier': 'SVC', 'svc_c': 0.9100
782849104319}. Best is trial 1 with value: 0.7897091722595079.
Accuracy: 0.7897091722595079
Best hyperparameters: {'classifier': 'SVC', 'svc_c': 3.2029361679282657}
```

In [175…

```
trial
```

Out[175…

```
FrozenTrial(number=1, values=[0.7897091722595079], datetime_start=datetime.datetime(2021, 10, 11, 17, 31, 55, 465534), datetime_co
mplete=datetime.datetime(2021, 10, 11, 17, 31, 55, 497449), params={'classifier': 'SVC', 'svc_c': 3.2029361679282657}, distributio
```

```
ns={'classifier': CategoricalDistribution(choices=('DecisionTree', 'SVC')), 'svc_c': LogUniformDistribution(high=10000000000.0, lo
w=1e-10)}, user_attrs={}, system_attrs={}, intermediate_values={}, trial_id=1, state=TrialState.COMPLETE, value=None)
```

In [176…]
```
study.best_params
```

Out[176…]
```
{'classifier': 'SVC', 'svc_c': 3.2029361679282657}
```

In [177…]
```
dt=DecisionTreeClassifier(max_depth=30)
dt.fit(X_train,y_train)
```

Out[177…]
```
DecisionTreeClassifier(max_depth=30)
```

In [180…]
```
y_pred=dt.predict(X_test)
print(confusion_matrix(y_test,y_pred))
print(accuracy_score(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
[[111  18]
 [ 37  26]]
0.7135416666666666
              precision    recall  f1-score   support

         0.0       0.75      0.86      0.80       129
         1.0       0.59      0.41      0.49        63

    accuracy                           0.71       192
   macro avg       0.67      0.64      0.64       192
weighted avg       0.70      0.71      0.70       192
```

In [181…]
```
y_pred[:100]
```

Out[181…]
```
array([0., 0., 0., 0., 1., 1., 0., 1., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 1.,
       0., 1., 0., 1., 0., 0., 0., 1., 0., 1., 0., 1., 0., 0., 0., 0., 0.,
       0., 0., 0., 1., 1., 0., 0., 0., 0., 1., 0., 1., 0., 0., 0., 0., 0.,
       0., 0., 1., 1., 1., 0., 1., 0., 0., 1., 0., 0., 1., 0., 0., 0., 1.,
       0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [182…]
```
# Converting y_pred to a dataframe which is an array
```

```
y_pred_1 = pd.DataFrame(y_pred)
```

In [183…

```
# Let's see the head
y_pred_1.head()
```

Out[183…

	0
0	0.0
1	0.0
2	0.0
3	0.0
4	1.0

In [184…

```
# Converting y_test to dataframe
y_test_df = pd.DataFrame(y_test)
```

In [185…

```
# Putting CustID to index
y_test_df['ID'] = y_test_df.index
```

In [186…

```
# Removing index for both dataframes to append them side by side
y_pred_1.reset_index(drop=True, inplace=True)
y_test_df.reset_index(drop=True, inplace=True)
```

In [187…

```
# Appending y_test_df and y_pred_1
y_pred_final = pd.concat([y_test_df, y_pred_1],axis=1)
```

In [188…

```
y_pred_final.head()
```

Out[188…

	Outcome	ID	0
0	0.0	183	0.0

	Outcome	ID	0
1	0.0	481	0.0
2	1.0	93	0.0
3	1.0	510	0.0
4	0.0	311	1.0

In [189…
```
y_pred_final = y_pred_final.rename(columns={0:"Diabetes_Probability"})
```

In [190…
```
y_pred_final.head(4)
```

Out[190…

	Outcome	ID	Diabetes_Probability
0	0.0	183	0.0
1	0.0	481	0.0
2	1.0	93	0.0
3	1.0	510	0.0

In [191…
```
y_pred_final.Diabetes_Probability.value_counts()
```

Out[191…
```
0.0    148
1.0     44
Name: Diabetes_Probability, dtype: int64
```

In [ ]: