Cancer Prediction using Binary Classification

Omar Kashmar

M.Tech Aerospace Engineering

IIT Bombay

213011004

Self-Project

Abstract:

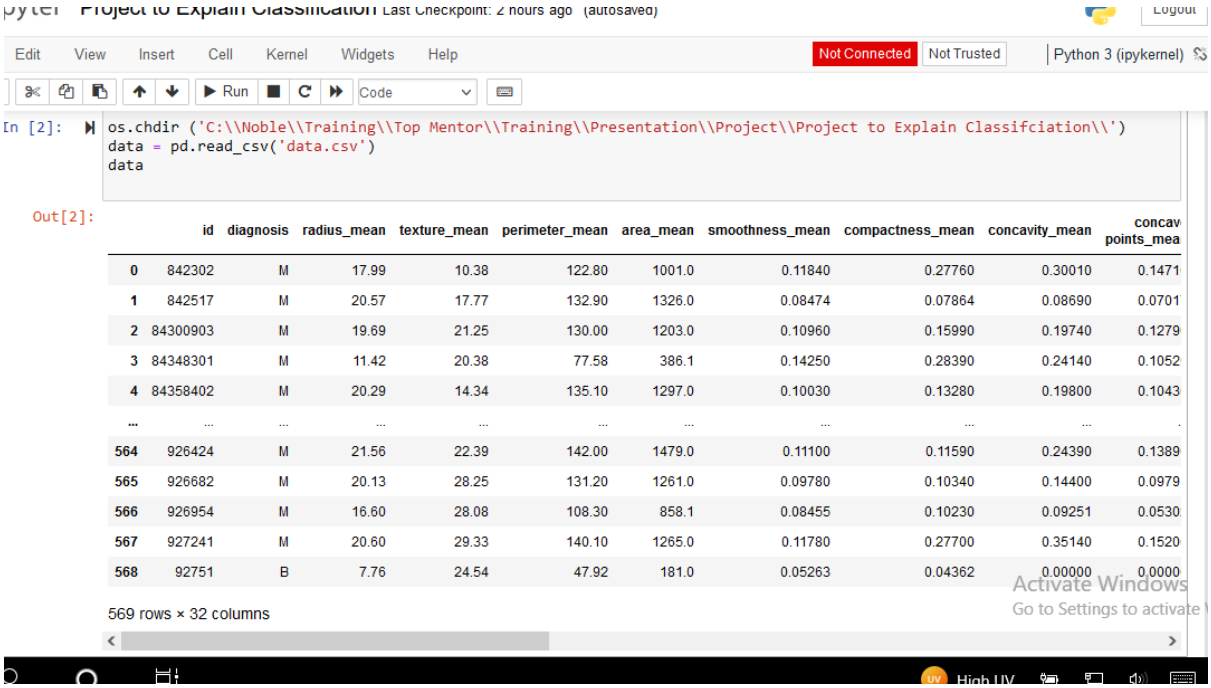
This report presents the results of a project focused on predicting the occurrence of cancer using binary classification techniques. The primary objective was to develop accurate models that could classify patients into two classes: those with cancer and those without cancer. The study employed various classification algorithms, including Support Vector Machines (SVM), Random Forest, and XGBoost, to achieve this goal. The report discusses the dataset, methodology, model selection, evaluation metrics, and the results obtained.

Introduction

Cancer is a complex and devastating disease that requires early detection and diagnosis for effective treatment. Predictive modeling using machine learning techniques offers the potential to assist medical professionals in identifying potential cases of cancer, enabling earlier interventions and improving patient outcomes. This project aims to explore the applicability of binary classification algorithms to predict the presence of cancer based on relevant medical features.

Dataset

The dataset used for this project consists of medical data collected from patients, including various features such as area_mean, perimeter_mean, concavity_worst status, and more. The target variable indicates whether a patient has cancer (class 1) or does not have cancer (class 0).



Methodology

1. **Data Preprocessing:** The dataset underwent preprocessing steps, including handling missing values, encoding categorical variables, and scaling numeric features.
2. **Data Splitting:** The dataset was split into training and testing sets to ensure unbiased evaluation of model performance.
3. **Resampling:** Given the imbalanced nature of the dataset, resampling techniques, specifically oversampling, were applied to mitigate class imbalance and improve model performance.
4. **Model Selection:** Three classification algorithms were chosen for this project: Support Vector Machines (SVM), Random Forest, and XGBoost. Hyperparameter tuning was performed using grid search and cross-validation.

5. **Model Evaluation:** The models were evaluated using various metrics, including accuracy, precision, recall, F1-score, and confusion matrices. Stratified cross-validation was employed to ensure robust evaluation.

Results:

The heat map that combine the relation between the variables:

Normalization

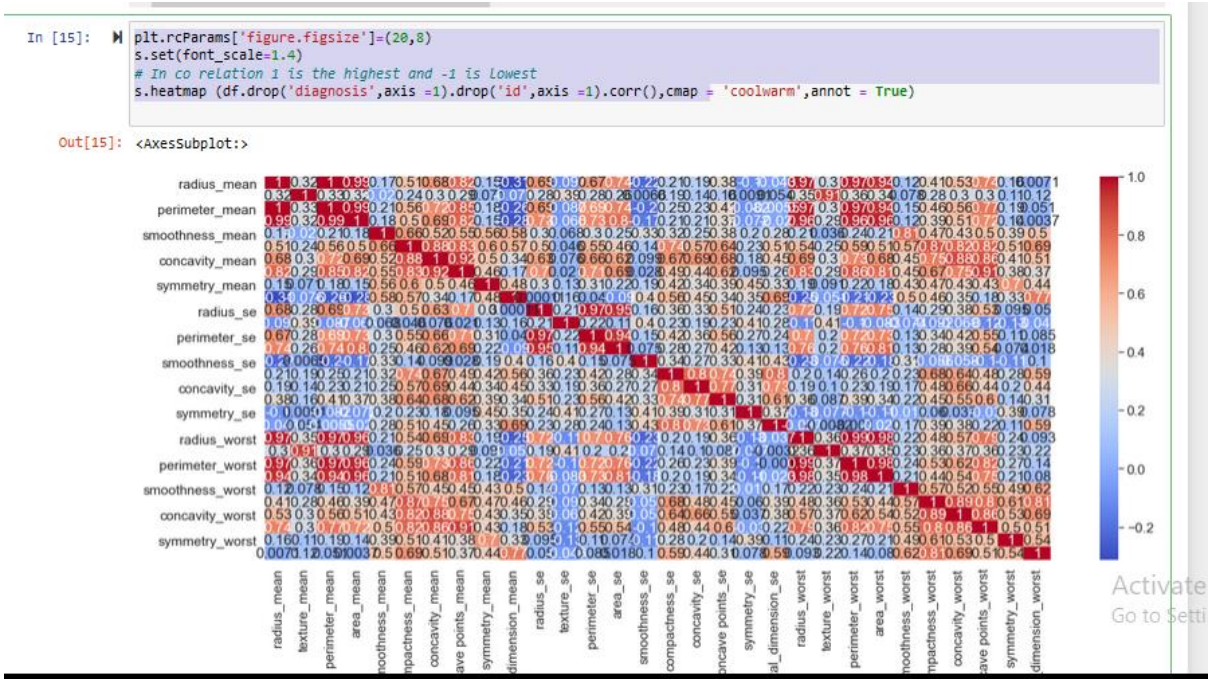
df_norm = (x- x.mean()) / (x.max()- x.min())

df_norm= pd.concat ([df_norm,y], axis =1)

df_norm

Print Co relation

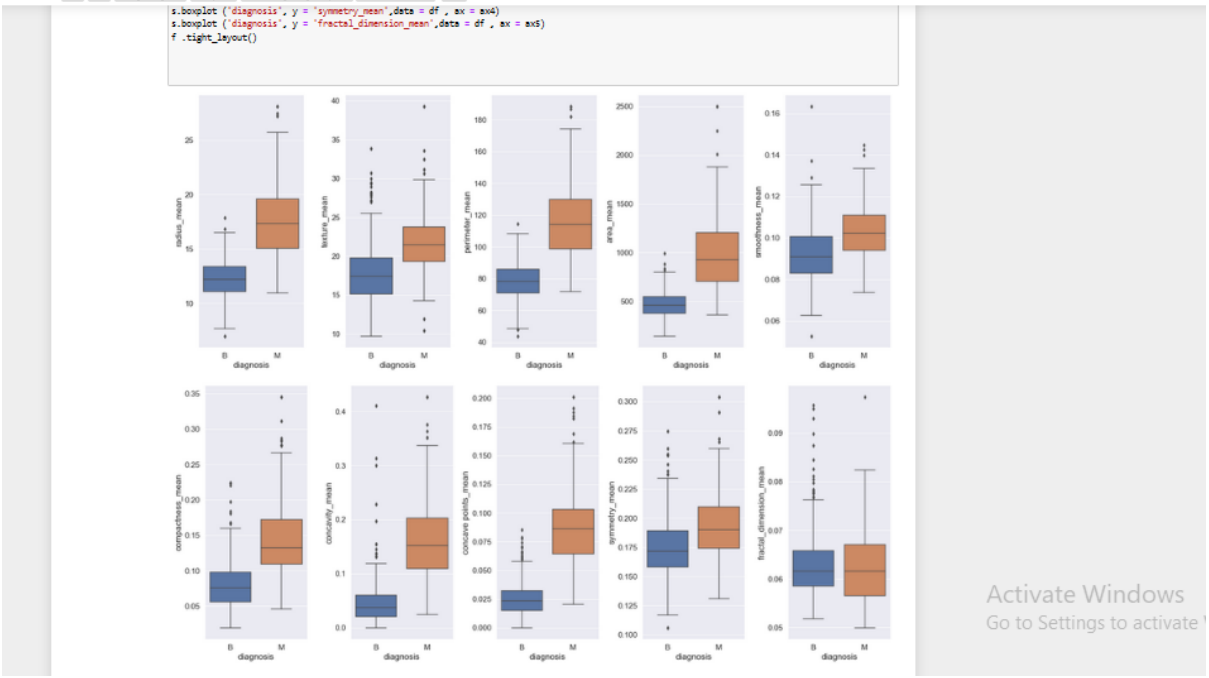
df.drop('diagnosis',axis =1).drop('id',axis =1).corr()



Create Box Plot

#box plot to check the outliers. Not going to remove outliers since this data is important.

```
# Observation, when 'diagnosis' is "B", the values are lower
plt.rcParams['figure.figsize']=(20,8)
f, (ax1,ax2,ax3,ax4,ax5) = plt.subplots (1,5)
s.boxplot ('diagnosis', y = 'radius_mean',data = df , ax = ax1)
s.boxplot ('diagnosis', y = 'texture_mean',data = df , ax = ax2)
s.boxplot ('diagnosis', y = 'perimeter_mean',data = df , ax = ax3)
s.boxplot ('diagnosis', y = 'area_mean',data = df , ax = ax4)
s.boxplot ('diagnosis', y = 'smoothness_mean',data = df , ax = ax5)
f .tight_layout()

f, (ax1,ax2,ax3,ax4,ax5) = plt.subplots (1,5)
s.boxplot ('diagnosis', y = 'compactness_mean',data = df , ax = ax1)
s.boxplot ('diagnosis', y = 'concavity_mean',data = df , ax = ax2)
s.boxplot ('diagnosis', y = 'concave points_mean',data = df , ax = ax3)
s.boxplot ('diagnosis', y = 'symmetry_mean',data = df , ax = ax4)
s.boxplot ('diagnosis', y = 'fractal_dimension_mean',data = df , ax = ax5)
f .tight_layout()
```



Create SVM Model

#cv = cross validation

```
param = {
        'C': [0.1,1,100,1000],
        'gamma':[0.0001,0.001, 0.005, 0.1,1, 3,5,10, 100]
      }
```

FitModel (x_norm,y_norm,'SVC',SVC(), param, cv =10)

Create Random Forest

param = { 'n_estimators': [100,500,1000,2000] }

FitModel (x_norm,y_norm,'Random Forest',RandomForestClassifier(), param, cv =10)

Create Random Forest Normal Way

np.random.seed(10)

```
x_train,x_test, y_train,y_test = train_test_split (x_norm,y_norm,test_size = 0.2)
forest = RandomForestClassifier (n_estimators = 500)
fit = forest.fit (x_train, y_train)
accuracy = fit.score(x_test,y_test)
predict = fit.predict(x_test)
cmatrix = confusion_matrix (y_test, predict)
print ('Classification Report:',classification_report(y_test,predict))
print ('Accuracy Score', (accuracy_score(y_test,predict)))
print ('Accuracy of Random Forest ', (accuracy))
print ('Confusion Matrix :\n',cmatrix)
```

XG Boost

```
param = { 'n_estimators': [100,500,1000,2000]  }
FitModel (x_norm,y_norm,'XGBoost', XGBClassifier(),param, cv = 10)
```

Final Results:

- ***Random Forest***

```
In [39]: X_res = pd.DataFrame(X_res)
         Y_res = pd.DataFrame(Y_res)
         X_res.columns = x_norm.columns
         param = { 'n_estimators': [100,500,1000,2000]  }
         FitModel (X_res [feat_to_keep], Y_res ,'Random Forest',RandomForestClassifier(), param, cv =10)

         Fitting 10 folds for each of 4 candidates, totalling 40 fits
         [1 1 0 1 0 1 0 0 1 0 1 1 0 1 0 1 1 1 1 1 1 0 0 0 0 1 0 1 1 0 0 0 1 1 1 1 1 1
          1 0 0 0 0 1 0 1 1 1 1 0 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0
          1 1 1 0 0 0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 0 1 1 0 0 0 0 0 0 0 0 1
          0 1 0 0 1 0 1 1 1 1 0 0 1 1 0 0 0 0 0 0 1 0 1 1 0 0 1 0 0 0 0 0 0 0]
         Best Params : {'n_estimators': 1000}
         Classification Report:               precision    recall  f1-score   support

                    0       0.98      0.96      0.97        68
                    1       0.96      0.99      0.97        75

             accuracy                           0.97       143
            macro avg       0.97      0.97      0.97       143
         weighted avg       0.97      0.97      0.97       143

         Accuracy Score 0.972027972027972
         Confusion Matrix :
          [[65  3]
          [ 1 74]]

In [40]: param = {
                    'C': [0.1,1,100,1000],
```

Accuracy for Random forest: from the table *:0.97*

- SVC

```
 [ 1 74]]
```

```
[40]: param = {
            'C': [0.1,1,100,1000],
            'gamma':[0.0001,0.001, 0.005, 0.1,1, 3,5,10, 100]
         }
FitModel (X_res [feat_to_keep], Y_res,'SVC',SVC(), param, cv =5)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
[1 1 0 1 0 1 0 1 1 0 1 1 0 1 0 1 1 1 1 1 1 0 0 0 0 1 0 1 1 0 0 0 1 1 1 1 1 1
 1 0 0 0 0 1 0 1 1 1 1 0 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1 1 0 1 1 1 0 0
 1 1 1 0 0 0 1 0 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 1
 0 1 0 0 1 0 1 1 1 0 0 1 1 0 0 0 0 0 0 1 0 1 1 0 0 1 0 0 0 0 1 0]
Best Params : {'C': 100, 'gamma': 10}
Classification Report:               precision    recall  f1-score   support

           0       1.00      0.94      0.97        68
           1       0.95      1.00      0.97        75

    accuracy                           0.97       143
   macro avg       0.97      0.97      0.97       143
weighted avg       0.97      0.97      0.97       143

Accuracy Score 0.972027972027972
Confusion Matrix :
 [[64  4]
 [ 0 75]]
```

Accuracy for **SVC** : from the table *:0.97.*

- `XGBoost:`

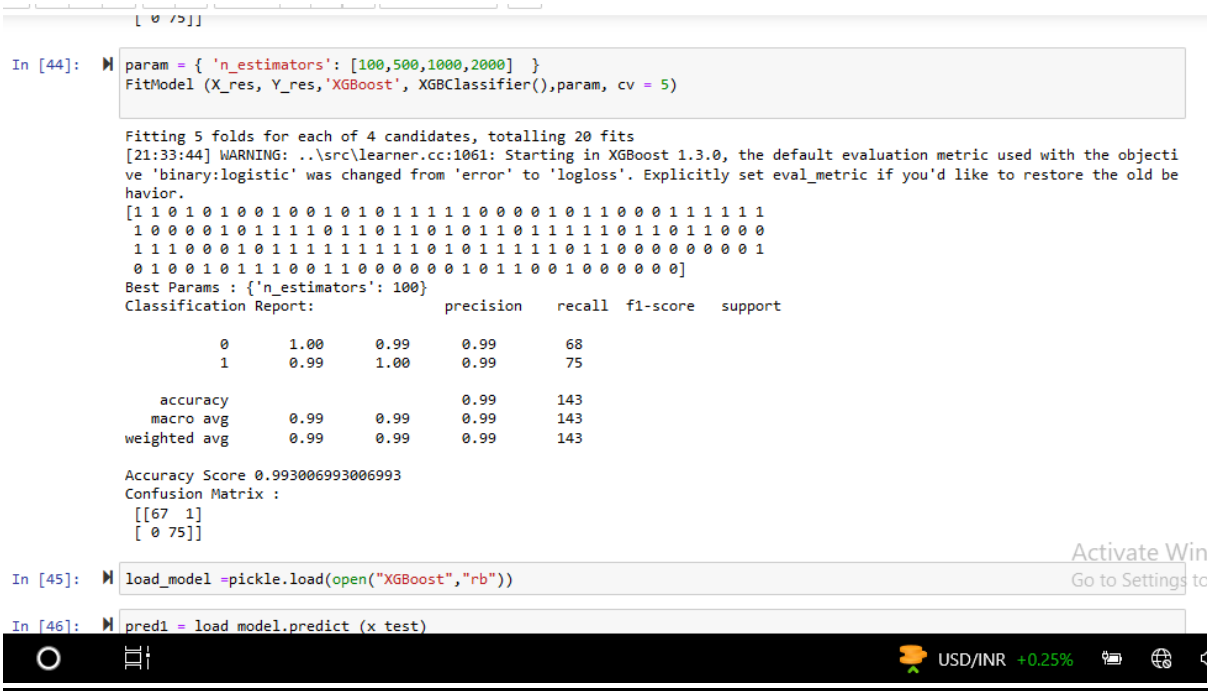
```
[41]: param = { 'n_estimators': [100,500,1000,2000]  }
FitModel (X_res [feat_to_keep], Y_res,'XGBoost', XGBClassifier(),param, cv = 5)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
[21:33:00] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used with the objecti
ve 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old be
havior.
[1 1 0 1 0 1 0 0 1 0 0 1 0 1 0 1 1 1 1 1 1 0 0 0 0 1 0 1 1 0 0 0 1 1 1 1 1 1
 1 0 0 0 0 1 0 1 1 1 1 0 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0
 1 1 1 0 0 0 1 0 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 1
 0 1 0 0 1 0 1 1 1 0 0 1 1 0 0 0 0 0 0 1 0 1 1 0 0 1 0 0 0 0 0 0]
Best Params : {'n_estimators': 100}
Classification Report:               precision    recall  f1-score   support

           0       0.99      0.97      0.98        68
           1       0.97      0.99      0.98        75

    accuracy                           0.98       143
   macro avg       0.98      0.98      0.98       143
weighted avg       0.98      0.98      0.98       143

Accuracy Score 0.9790209790209791
Confusion Matrix :
 [[66  2]
 [ 1 74]]
```

```
[42]: param = { 'n_estimators': [100,500,1000,2000]  }
FitModel (X_res, Y_res ,'Random Forest',RandomForestClassifier(), param, cv =10)
```

```
Fitting 10 folds for each of 4 candidates, totalling 40 fits
```


Accuracy for ***XGBoost***: from the table *:0.98.*



Executed Hyperparameter Tuning using **GridsearchCV** resulting in a maximum of **99% =0.99 accuracy** by the XGBoost

Conclusion

The results demonstrate that the chosen classification algorithms, namely **Support Vector Machines, Random Forest, and XGBoost**, all achieved high accuracy and strong performance in **predicting cancer cases**. These models hold promise in aiding medical professionals in early cancer detection, which can significantly impact patient outcomes and treatment strategies. Further research could explore the integration of additional features and **optimization** of model **hyperparameters**.

Import the nesserly librarryes:

```
%matplotlib inline
import numpy as np
import pandas as pd
import seaborn as s
# seaborn and pandas for unbalanced data exploration and visualization
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn import svm
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, LabelEncoder
#optimum parameter choosing
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from xgboost import XGBClassifier
import pickle
import os
import warnings
warnings.filterwarnings('ignore')
```