DATA MINING ASSIGNMENT

BREAST CANCER PREDICTION

INSPIRED BY LIFE

Manipal University
Jaipur

By-

Name: Jyoti Tuli

Reg No.: 179302068

Background of The Study

Data Mining is rapidly growing to occupy all the industries of the world today. This is just the beginning. The medical industries collect huge amounts of data containing hidden information useful for making effective decisions by providing appropriate results using data mining techniques.

Data mining knowledge is used to give a user-oriented approach to new and hidden patterns in the data which can be used by the medical experts for predicting breast cancer which can improve the entire research and prevention process.

CONTENTS

Introduction

In this project, multiple modern machine learning and pattern recognition methods have been used in order to classify or predict the risk of breast cancer based on the data. The methods discussed in this project consist of a number of classification methods (i.e. Naïve Bayes, K-NN, Decision Trees, Logistic Regression and SVM).

Libraries Used

Import libraries

```
import numpy as np                  #large collection of high-level mathematical functions
import pandas as pd                 #Used for data manipulation and analysis
import matplotlib.pyplot as plt     #Collection of command style functions
import seaborn as sns               #Data visualization library based on matplotlib
```

```
#sklearn- software machine learning library
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix , classification_report , accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
```

Loading Dataset

Reading Dataset

```
df = pd.read_csv("dataset.csv")
```

```
#Display dataset
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	te
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	

5 rows × 33 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

Data Preprocessing

Data Preprocessing

```
df.drop('id',axis=1,inplace=True)
df.drop('Unnamed: 32',axis=1,inplace=True)
# size of the dataframe
len(df)
```

```
569
```

```
df.diagnosis.unique()
```

```
array(['M', 'B'], dtype=object)
```

```
df['diagnosis'] = df['diagnosis'].map({'M':1,'B':0})
df.head()
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean
0	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419
1	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812
2	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069
3	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597
4	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809

5 rows × 31 columns

```
df.describe()
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.
mean	0.372583	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.
std	0.483918	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.
min	0.000000	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.
25%	0.000000	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.
50%	0.000000	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.
75%	1.000000	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.
max	1.000000	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.

8 rows × 31 columns

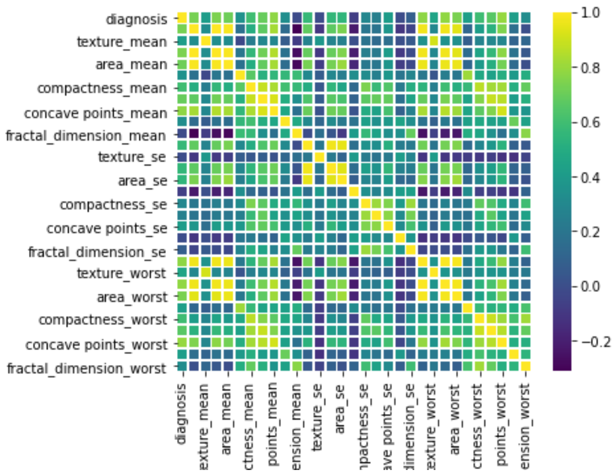
Data Visualization

Heatmap

Heatmap

```
corr = df.corr()
# plot the heatmap
fig = plt.figure(figsize=(6,5))
sns.heatmap(corr,linewidths=.75,cmap= 'viridis')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2d41226ebe0>
```

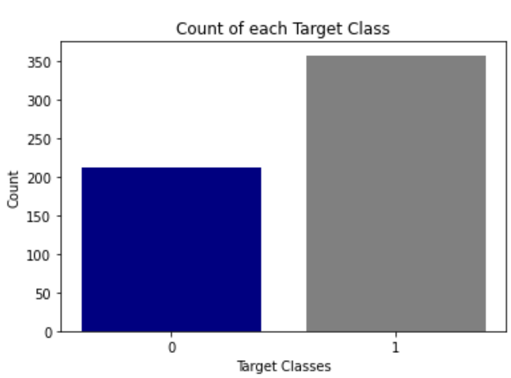


Bar graph

Bar Graph

```
plt.rcParams['figure.figsize'] = 6,4
plt.bar(df['diagnosis'].unique(), df['diagnosis'].value_counts(), color = ['grey', 'navy'])
plt.xticks([0, 1])
plt.xlabel('Target Classes')
plt.ylabel('Count')
plt.title('Count of each Target Class')
```

```
Text(0.5, 1.0, 'Count of each Target Class')
```



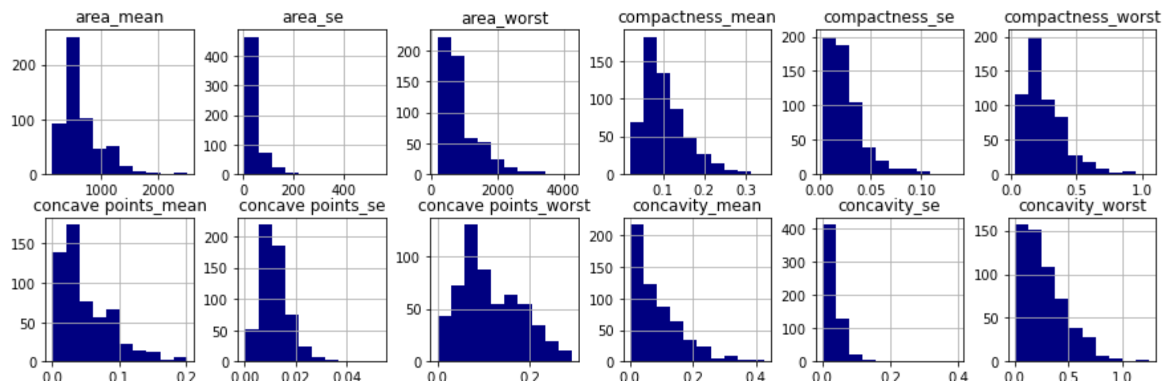
Histogram

Histogram

```
fig = plt.figure(figsize = (15,15))
ax = fig.gca()

df.hist(ax=ax,color='navy')
plt.show()
```

```
<ipython-input-53-553dfa30f3e4>:4: UserWarning: To output multiple subplots, the figure containing the passed axes is being cle
ared
  df.hist(ax=ax,color='navy')
```



<u>Data Splitting</u>

Spliting of Data

```
y=df['diagnosis']
x=df.drop('diagnosis',axis=1)
```

```
train_size=0.80
test_size=0.20
seed=5

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=train_size,test_size=test_size,random_state=seed)
```

Naïve Bayes Algorithm

Naive Bayes Algorithm

```
#Create a Gaussian Classifier
nb = GaussianNB()

# Train the model using the training sets
nb.fit(x_train, y_train)

#Predict Output
nb_pred = nb.predict(x_test)
```

```
#Predicting the score
nb_score = accuracy_score(y_test, nb_pred)
nb_score
```

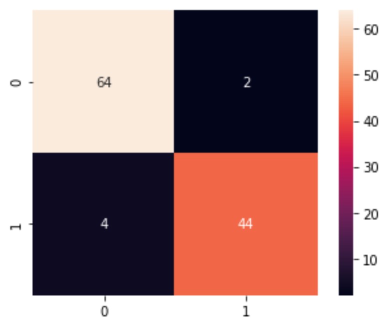
```
0.9473684210526315
```

```
cf_matrix=confusion_matrix(y_test , nb_pred)
sns.heatmap(cf_matrix, annot=True,square=True)
print(classification_report(y_test , nb_pred))
```

```
              precision    recall  f1-score   support

           0       0.94      0.97      0.96        66
           1       0.96      0.92      0.94        48

    accuracy                           0.95       114
   macro avg       0.95      0.94      0.95       114
weighted avg       0.95      0.95      0.95       114
```



Support Vector Machine Algorithm

Support Vector Algorithm

```
#Create a SVM Classifier
svm_class = svm.SVC()

# Train the model using the training sets
svm_class.fit(x_train, y_train)

#Predict Output
svm_pred = svm_class.predict(x_test)
```

```
#Predicting the score
svm_score = accuracy_score(y_test, svm_pred)
svm_score
```

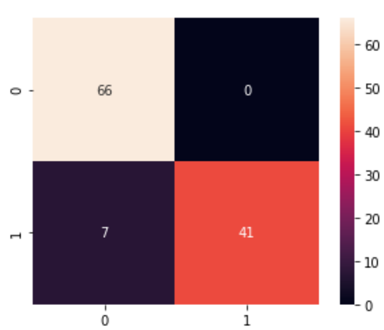
```
0.9385964912280702
```

```
cf_matrix=confusion_matrix(y_test , svm_pred)
sns.heatmap(cf_matrix, annot=True,square=True)
print(classification_report(y_test , svm_pred))
```

```
              precision    recall  f1-score   support

           0       0.90      1.00      0.95        66
           1       1.00      0.85      0.92        48

    accuracy                           0.94       114
   macro avg       0.95      0.93      0.94       114
weighted avg       0.94      0.94      0.94       114
```



K Nearest Neighbours

K-Nearest Neighbours

```
#Create a KNN Classifier
model = KNeighborsClassifier(n_neighbors=7)

# Train the model using the training sets
model.fit(x_train,y_train)

#Predict Output
y_predicted= model.predict(x_test)
```

```
knn_score = accuracy_score(y_test, y_predicted)
knn_score
```

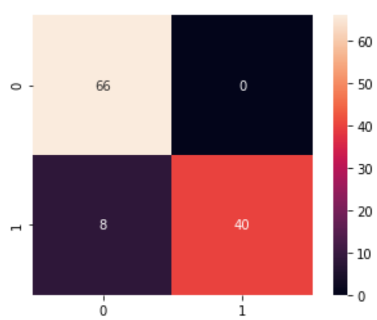
```
0.9298245614035088
```

```
cf_matrix=confusion_matrix(y_test , y_predicted)
sns.heatmap(cf_matrix, annot=True,square=True)
print(classification_report(y_test , y_predicted))
```

```
              precision    recall  f1-score   support

           0       0.89      1.00      0.94        66
           1       1.00      0.83      0.91        48

    accuracy                           0.93       114
   macro avg       0.95      0.92      0.93       114
weighted avg       0.94      0.93      0.93       114
```



Decision Tree

Decision Tree

```
#Create a KNN Classifier
model = DecisionTreeClassifier()
# Train the model using the training sets
model.fit(x_train,y_train)

#Predict Output
y_predicted= model.predict(x_test)
```

```
dt_score = accuracy_score(y_test, y_predicted)
dt_score
```

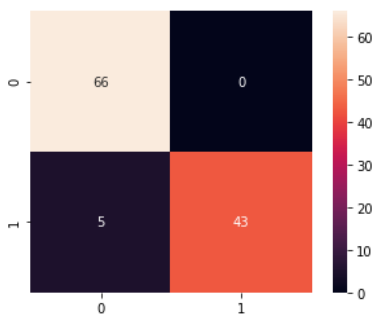
```
0.956140350877193
```

```
cf_matrix=confusion_matrix(y_test , y_predicted)
sns.heatmap(cf_matrix, annot=True,square=True)
print(classification_report(y_test , y_predicted))
```

```
              precision    recall  f1-score   support

           0       0.93      1.00      0.96        66
           1       1.00      0.90      0.95        48

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```



Logistic Regression

Logistic Regression

```
#Create a KNN Classifier
model = LogisticRegression()
# Train the model using the training sets
model.fit(x_train,y_train)

#Predict Output
y_predicted= model.predict(x_test)
```

```
lr_score = accuracy_score(y_test, y_predicted)
lr_score
```

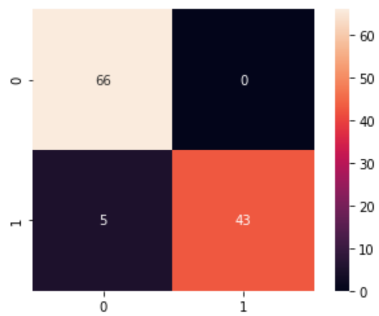
```
0.956140350877193
```

```
cf_matrix=confusion_matrix(y_test , y_predicted)
sns.heatmap(cf_matrix, annot=True,square=True)
print(classification_report(y_test , y_predicted))
```

```
              precision    recall  f1-score   support

           0       0.93      1.00      0.96        66
           1       1.00      0.90      0.95        48

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

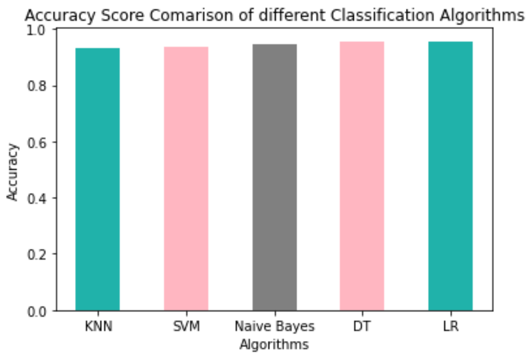


Comparison between Algorithms

Comparison

```
algos = ["KNN", "SVM", "Naive Bayes","DT","LR"]
scores = [knn_score, svm_score, nb_score,dt_score,lr_score]
```

```
plt.bar(algos, scores, width=0.5,color = ['lightseagreen', 'lightpink','grey','lightpink','lightseagreen'])
plt.title("Accuracy Score Comarison of different Classification Algorithms")
plt.xlabel('Algorithms')
plt.ylabel('Accuracy')
plt.show()
```



Conclusion

Proposed project is user-friendly, scalable, reliable and an expandable analysis which can also help in reducing treatment costs by providing initial diagnostics in time. The model can also serve the purpose of training tool for medical students and will be a soft diagnostic tool.

There are many possible improvements that could be explored to improve the scalability and accuracy of this prediction system. As we have developed a generalized system, in future we can use this system for the analysis of different data sets.

The performance of the diagnosis can be improved significantly by handling numerous class labels in the breast cancer prediction process, and it can be another positive direction of research.