Artificial Intelligence Project Report

Title: Cancer Prediction by Deep learning (Bagging and Boosting)

Group: G 4

Team Members:
Amandeep Singh Vohra (502004073),
Babit Abrol (502004129)

Submitted to: -
Dr. Nitin Arvind Shelke

Problem Statement

- According to the world health organization (WHO) Breast cancer is the most frequent cancer among women, impacting 2.1 million women each year, and also causes the greatest number of cancer- related deaths among women.
- Limited resource settings with weak health systems where the majority of women are diagnosed in late stages should prioritize early diagnosis programs based on awareness of early signs and symptoms and prompt referral to diagnosis and treatment.

Solution

- The goal is to increase the proportion of breast cancers identified at an early stage, allowing for more effective treatment to be used and reducing the risks of death from breast cancer. Since early detection of cancer is key to effective treatment of breast cancer we use various machine learning algorithms **to predict if a tumor is benign or malignant**, based on the features provided by the data.

Data Collection

Data Sources-

The data for our project has been collected by the Kaggle.com .

Data Overview (Summary)

For our group project we chose Breast Cancer dataset. for cancer prediction among women. The data includes: -

Attribute Information:

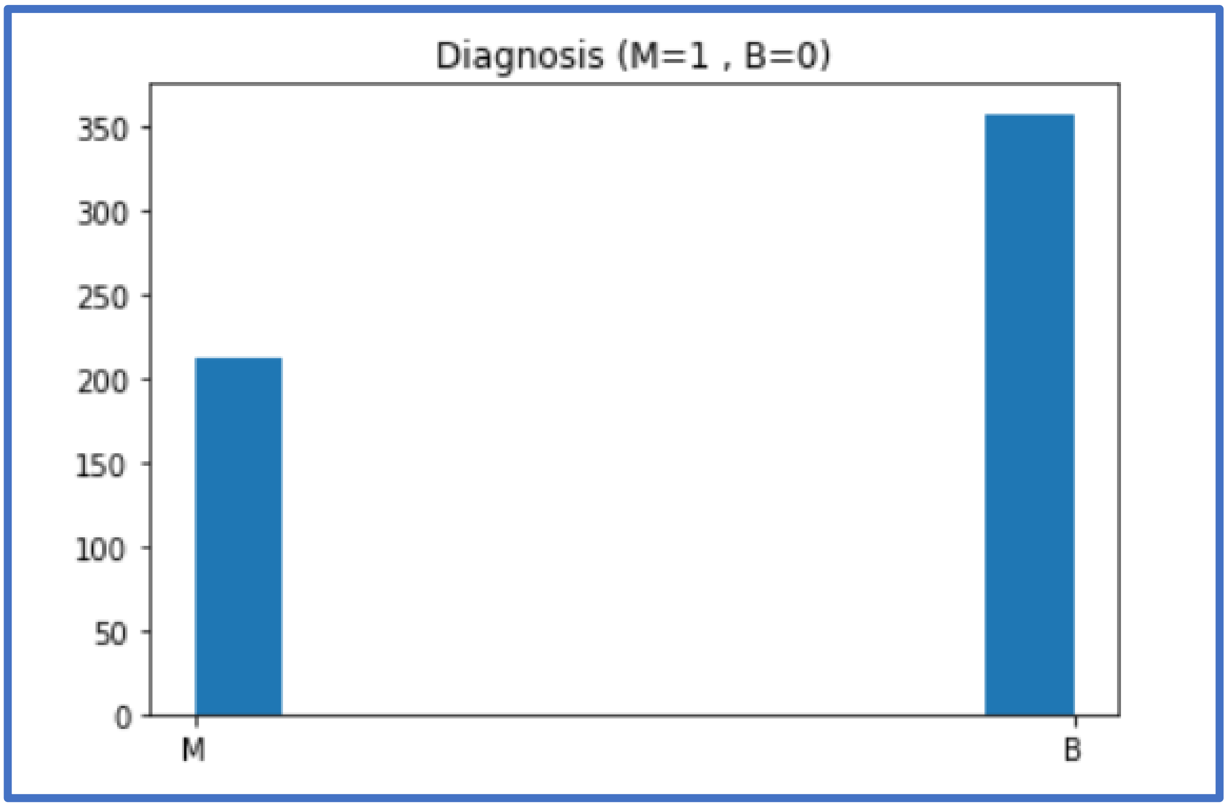
1) ID number

2) Diagnosis (M = malignant, B = benign) -3 -32.

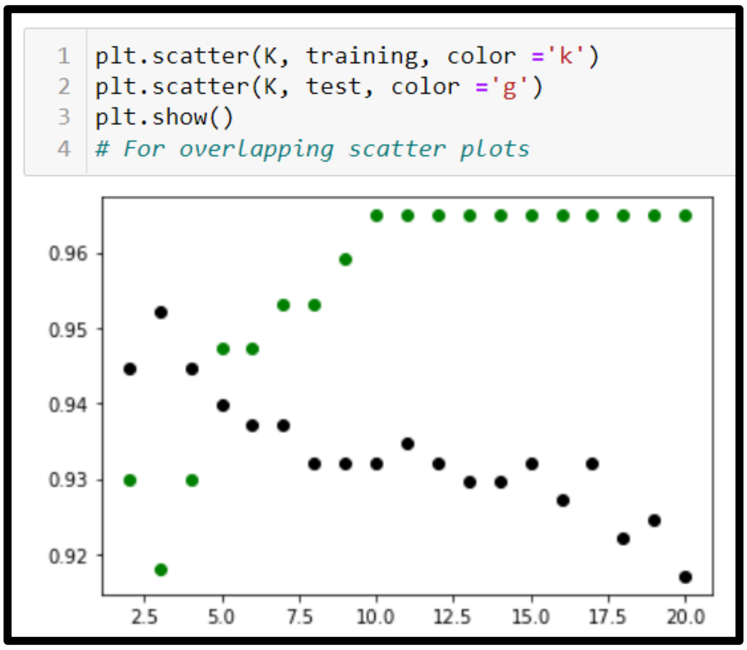
Required Libraries

- ***NUMPY***
- ***PANDAS***
- ***SK- LEARN***

Data Visualization



The above Bar plot shows how many diagnostic patients were Malignant (harmful) and Benign (less harmful). In our data of 569 entries, we had 215 Malignant patient and 354 Benign patients.

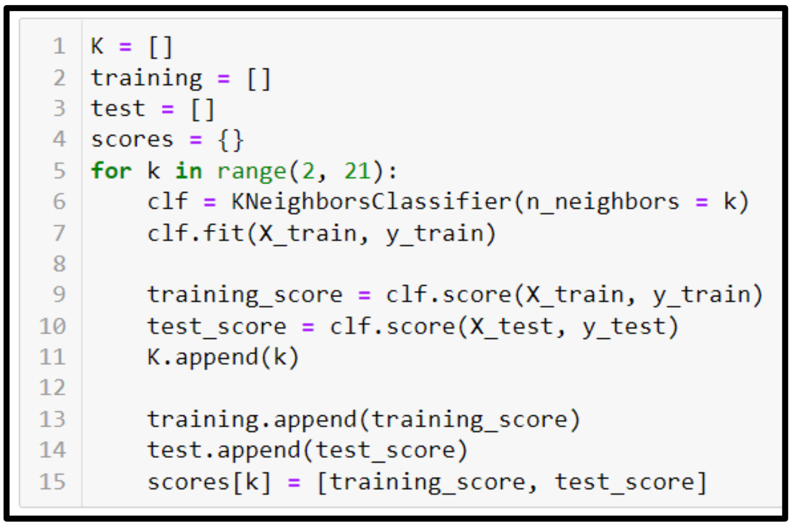


```
plt.scatter(K, training, color ='k')
plt.scatter(K, test, color ='g')
plt.show()
# For overlapping scatter plots
```

The given scatter plot is a summarized view of the above created scatter plots.

From this plot we can see that there is a small cluster around the value 5, so we can take the value of k as 5.

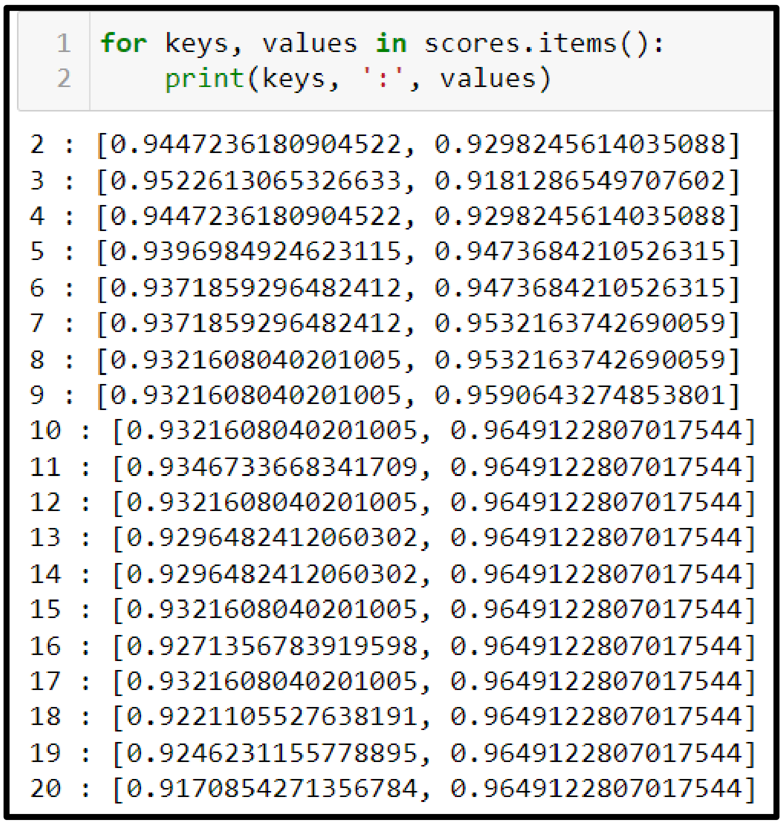
Model Building & Scoring



```python
K = []
training = []
test = []
scores = {}
for k in range(2, 21):
    clf = KNeighborsClassifier(n_neighbors = k)
    clf.fit(X_train, y_train)

    training_score = clf.score(X_train, y_train)
    test_score = clf.score(X_test, y_test)
    K.append(k)

    training.append(training_score)
    test.append(test_score)
    scores[k] = [training_score, test_score]
```

Using Forloop we created KNN Classifier Model.

Also calculated scores for training and testing datasets.



```python
for keys, values in scores.items():
    print(keys, ':', values)
```

```
2 : [0.9447236180904522, 0.9298245614035088]
3 : [0.9522613065326633, 0.9181286549707602]
4 : [0.9447236180904522, 0.9298245614035088]
5 : [0.9396984924623115, 0.9473684210526315]
6 : [0.9371859296482412, 0.9473684210526315]
7 : [0.9371859296482412, 0.9532163742690059]
8 : [0.9321608040201005, 0.9532163742690059]
9 : [0.9321608040201005, 0.9590643274853801]
10 : [0.9321608040201005, 0.9649122807017544]
11 : [0.9346733668341709, 0.9649122807017544]
12 : [0.9321608040201005, 0.9649122807017544]
13 : [0.9296482412060302, 0.9649122807017544]
14 : [0.9296482412060302, 0.9649122807017544]
15 : [0.9321608040201005, 0.9649122807017544]
16 : [0.9271356783919598, 0.9649122807017544]
17 : [0.9321608040201005, 0.9649122807017544]
18 : [0.9221105527638191, 0.9649122807017544]
19 : [0.9246231155778895, 0.9649122807017544]
20 : [0.9170854271356784, 0.9649122807017544]
```

Following are the scores for training and testing datasets.

Implementation of Bagging and Boosting using Decision Tree Classifier as Base Estimator

Ensemble Learning is a machine learning concept in which multiple models are trained using the same learning algorithm.

Bagging is used when the goal is to reduce the variance of a decision tree classifier. Here the objective is to create several subsets of data from training sample chosen randomly with replacement. Each collection of subset data is used to train their decision trees. As a result, we get an ensemble of different models. Average of all the predictions from different trees are used which is more robust than a single decision tree classifier.

Bagging Steps:

- Suppose there are N observations and M features in training data set. A sample from training data set is taken randomly with replacement.
- A subset of M features are selected randomly and whichever feature gives the best split is used to split the node iteratively.
- The tree is grown to the largest.
- Above steps are repeated n times and prediction is given based on the aggregation of predictions from n number of trees.

Boosting is used to create a collection of predictors. In this technique, learners are learned sequentially with early learners fitting simple models to the data and then analysing data for errors. Consecutive trees (random sample) are fit and at every step, the goal is to improve the accuracy from the prior tree. When an input is misclassified by a hypothesis, its weight is increased so that next hypothesis is more likely to classify it correctly. This process converts weak learners into better performing model.

Boosting Steps:

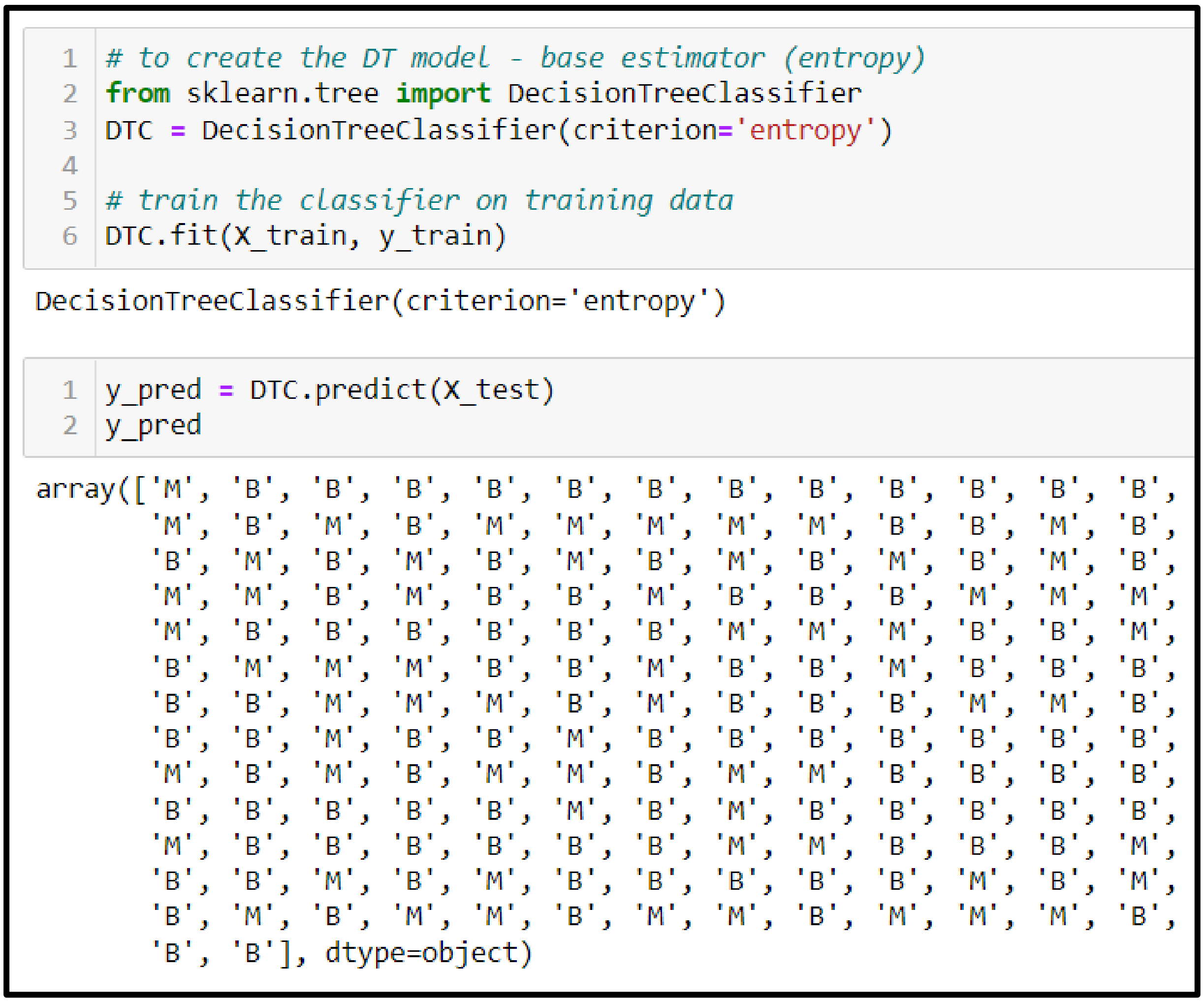
- Draw a random subset of training samples d1 without replacement from the training set D to train a weak learner C1
- Draw second random training subset d2 without replacement from the training set and add 50 percent of the samples that were previously falsely classified/misclassified to train a weak learner C2
- Find the training samples d3 in the training set D on which C1 and C2 disagree to train a third weak learner C3
- Combine all the weak learners via majority voting.

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

In a Decision tree, there are **two nodes**, which are the **Decision Node** and **Leaf Node**. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.

In order to build a tree, we use the **CART** algorithm, which stands for Classification and Regression Tree algorithm.



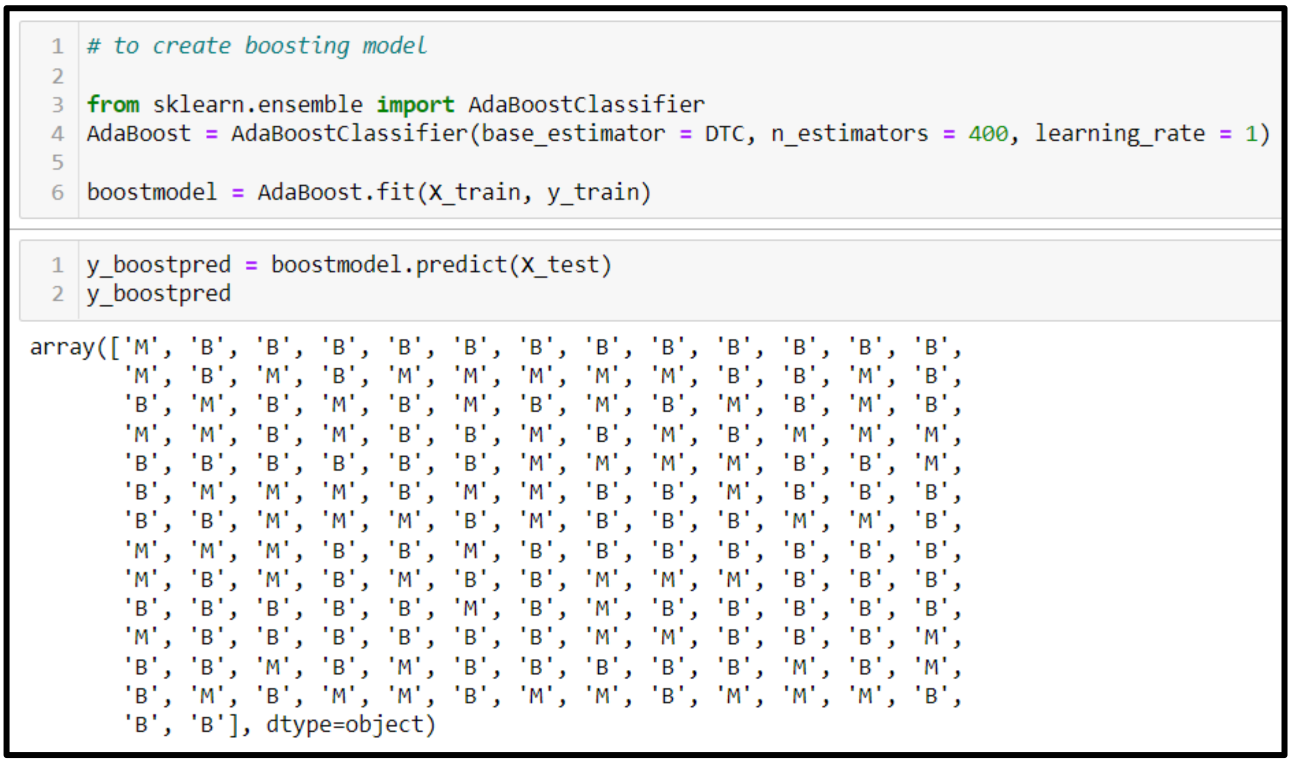
```python
# to create the DT model - base estimator (entropy)
from sklearn.tree import DecisionTreeClassifier
DTC = DecisionTreeClassifier(criterion='entropy')

# train the classifier on training data
DTC.fit(X_train, y_train)
```

```
DecisionTreeClassifier(criterion='entropy')
```

```python
y_pred = DTC.predict(X_test)
y_pred
```

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

Boosting Model

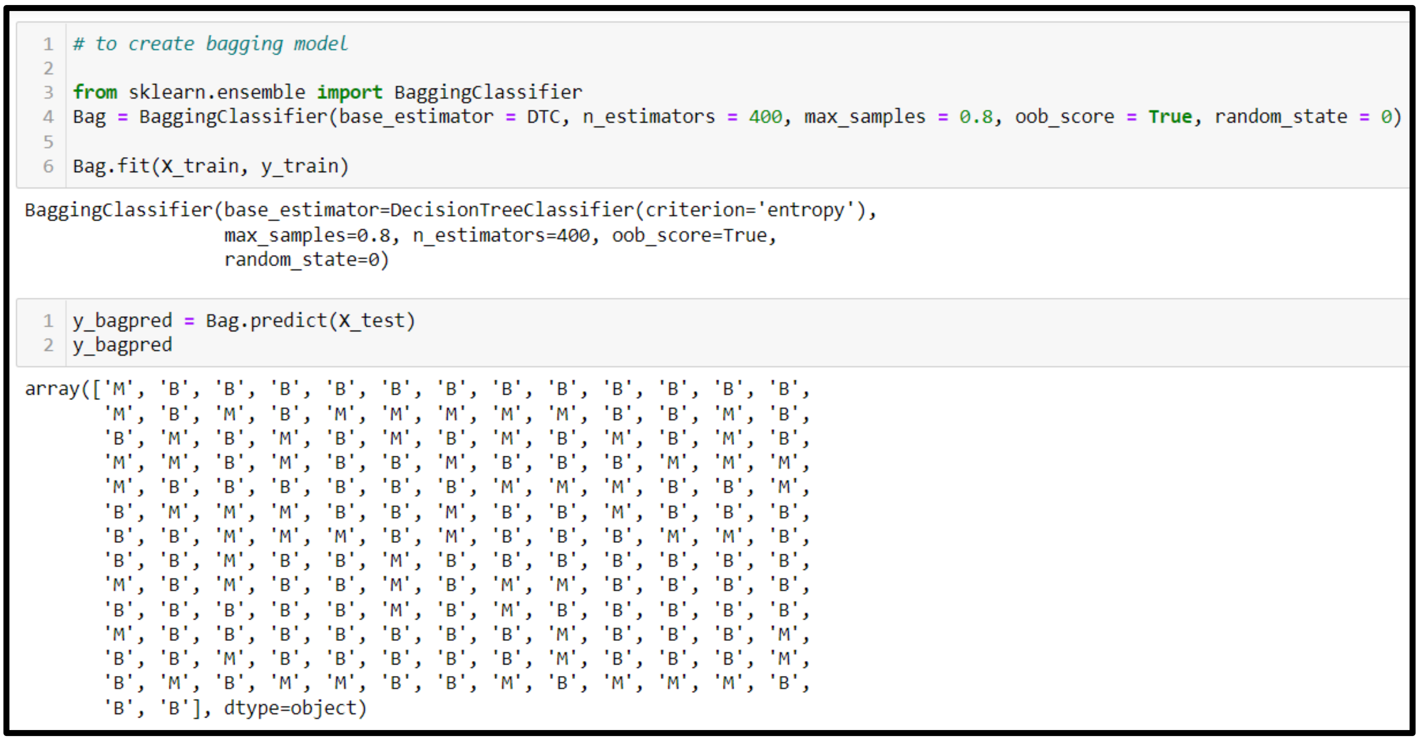


```python
# to create boosting model

from sklearn.ensemble import AdaBoostClassifier
AdaBoost = AdaBoostClassifier(base_estimator = DTC, n_estimators = 400, learning_rate = 1)

boostmodel = AdaBoost.fit(X_train, y_train)
```

```python
y_boostpred = boostmodel.predict(X_test)
y_boostpred
```

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

Bagging Model

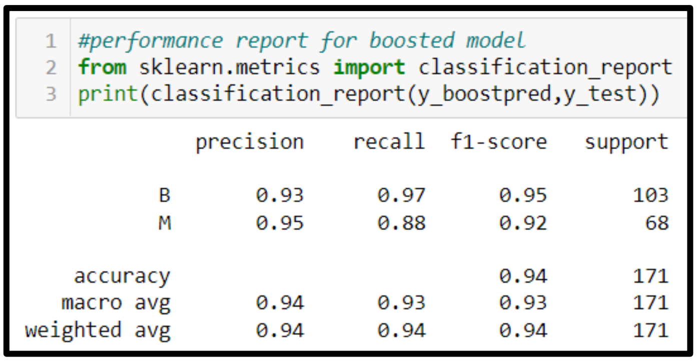


```python
# to create bagging model

from sklearn.ensemble import BaggingClassifier
Bag = BaggingClassifier(base_estimator = DTC, n_estimators = 400, max_samples = 0.8, oob_score = True, random_state = 0)

Bag.fit(X_train, y_train)
```

```
BaggingClassifier(base_estimator=DecisionTreeClassifier(criterion='entropy'),
                  max_samples=0.8, n_estimators=400, oob_score=True,
                  random_state=0)
```

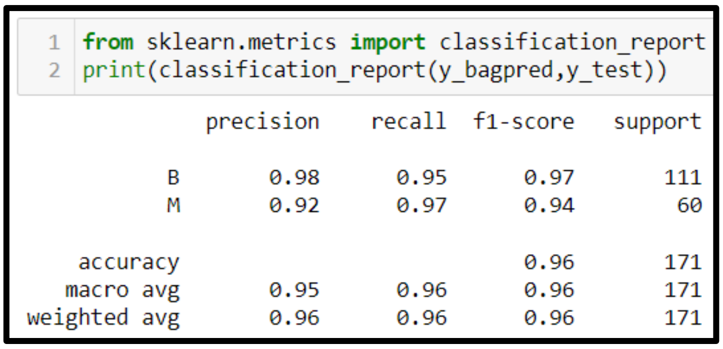
```python
y_bagpred = Bag.predict(X_test)
y_bagpred
```

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



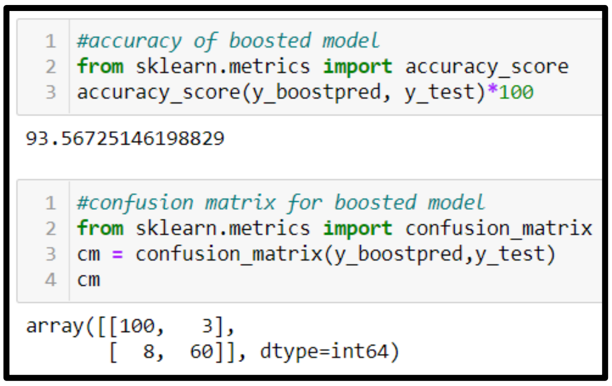
```
#performance report for boosted model
from sklearn.metrics import classification_report
print(classification_report(y_boostpred,y_test))
```

```
              precision    recall  f1-score   support

           B       0.93      0.97      0.95       103
           M       0.95      0.88      0.92        68

    accuracy                           0.94       171
   macro avg       0.94      0.93      0.93       171
weighted avg       0.94      0.94      0.94       171
```



```
from sklearn.metrics import classification_report
print(classification_report(y_bagpred,y_test))
```

```
              precision    recall  f1-score   support

           B       0.98      0.95      0.97       111
           M       0.92      0.97      0.94        60

    accuracy                           0.96       171
   macro avg       0.95      0.96      0.96       171
weighted avg       0.96      0.96      0.96       171
```

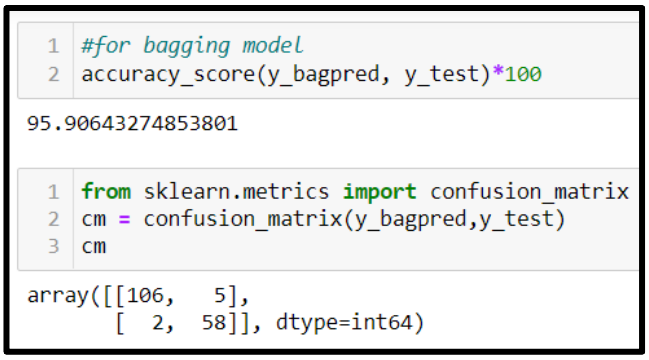


```
#accuracy of boosted model
from sklearn.metrics import accuracy_score
accuracy_score(y_boostpred, y_test)*100
```

```
93.56725146198829
```

```
#confusion matrix for boosted model
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_boostpred,y_test)
cm
```

```
array([[100,   3],
       [  8,  60]], dtype=int64)
```



```
#for bagging model
accuracy_score(y_bagpred, y_test)*100
```

```
95.90643274853801
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_bagpred,y_test)
cm
```

```
array([[106,   5],
       [  2,  58]], dtype=int64)
```

Boosting	Bagging
A way of combining predictions that belong to the different types.	The simplest way of combining predictions that belong to the same type.
Aim to decrease bias, not variance.	Aim to decrease variance, not bias.
Models are weighted according to their performance.	Each model receives equal weight.
If the classifier is stable and simple (high bias) the apply boosting.	If the classifier is unstable (high variance), then apply bagging.
Example: The AdaBoost uses Boosting techniques	Example: The Random forest model uses Bagging.

THANK YOU