# Ensemble Techniques

**Instructions:**

Please share your answers filled in-line in the word document. Submit code separately wherever applicable.

Please ensure you update all the details:

**Name: Anandakrishnan k v ;;;;; Batch ID:** 19042021

**Topic: Ensemble Techniques**

**Grading Guidelines:**

**1. An assignment submission is considered complete only when correct and executable code(s) are submitted along with the documentation explaining the method and results. Failing to submit either of those will be considered an invalid submission and will not be considered for evaluation.**

**2. Assignments submitted after the deadline will affect your grades.**

**Grading:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ans** | **Date** |  |  | **Ans** | **Date** |
| Correct | On time | A | 100 |  |  |
| 80% & above | On time | B | 85 | Correct | Late |
| 50% & above | On time | C | 75 | 80% & above | Late |
| 50% & below | On time | D | 65 | 50% & above | Late |
|  |  | E | 55 | 50% & below |  |
| Copied/No Submission |  | F | 45 |  |  |

* **Grade A: (>= 90):** When all assignments are submitted on or before the given deadline.
* **Grade B: (>= 80 and < 90):** 
  + When assignments are submitted on time but less than 80% of problems are completed.

(OR)

* + All assignments are submitted after the deadline.
* **Grade C: (>= 70 and < 80):** 
  + When assignments are submitted on time but less than 50% of the problems are completed.

(OR)

* + Less than 80% of problems in the assignments are submitted after the deadline.
* **Grade D: (>= 60 and < 70):**
  + Assignments submitted after the deadline and with 50% or less problems.
* **Grade E: (>= 50 and < 60):** 
  + Less than 30% of problems in the assignments are submitted after the deadline.

(OR)

* + Less than 30% of problems in the assignments are submitted before the deadline.
* **Grade F: (< 50):** No submission (or) malpractice.

1. **Business Problem**

Given is the diabetes dataset. Build an ensemble model to correctly classify the outcome variable and improve your model prediction by using GridSearchCV. You must apply Bagging, Boosting, Stacking, and Voting on the dataset.

* 1. **What is the business objective?**

Built a model toPredict diabetes conditions by analysing the given input variables

* 1. **Are there any constraints?**

Maximize: model accuracy

Minimize: model complexity

Minimize: false negative cases of model out

**Python Code:-**

#############################Bagging###########################

import pandas as pd

df = pd.read\_csv("C://Users//user//Downloads//ensamble//Diabeted\_Ensemble.csv")

# Dummy variables

df.head()

df.info()

# Input and Output Split

predictors = df.loc[:, df.columns!=" Class variable"]

type(predictors)

target = df[" Class variable"]

type(target)

# Train Test partition of the data

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(predictors, target, test\_size = 0.2, random\_state=0)

from sklearn import tree

clftree = tree.DecisionTreeClassifier(max\_depth=7)

from sklearn.ensemble import BaggingClassifier

bag\_clf = BaggingClassifier(base\_estimator = clftree, n\_estimators = 500,

bootstrap = True, n\_jobs = 1, random\_state = 42)

bag\_clf.fit(x\_train, y\_train,)

from sklearn.metrics import accuracy\_score, confusion\_matrix

# Evaluation on Testing Data

confusion\_matrix(y\_test, bag\_clf.predict(x\_test))

accuracy\_score(y\_test, bag\_clf.predict(x\_test))

# Evaluation on Training Data

confusion\_matrix(y\_train, bag\_clf.predict(x\_train))

accuracy\_score(y\_train, bag\_clf.predict(x\_train))

###########################Boost#############################

# applied adaboost

import pandas as pd

df = pd.read\_csv("C://Users//user//Downloads//ensamble//Diabeted\_Ensemble.csv")

# Dummy variables

df.head()

df.info()

# Input and Output Split

predictors = df.loc[:, df.columns!=" Class variable"]

type(predictors)

target = df[" Class variable"]

type(target)

# Train Test partition of the data

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(predictors, target, test\_size = 0.2, random\_state=0)

# Refer to the links

# https://scikit-learn.org/stable/modules/classes.html#module-sklearn.ensemble

# https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html#sklearn.ensemble.AdaBoostClassifier

from sklearn.ensemble import AdaBoostClassifier

ada\_clf = AdaBoostClassifier(learning\_rate = 0.5, n\_estimators = 500)

ada\_clf.fit(x\_train, y\_train)

from sklearn.metrics import accuracy\_score, confusion\_matrix

# Evaluation on Testing Data

confusion\_matrix(y\_test, ada\_clf.predict(x\_test))

accuracy\_score(y\_test, ada\_clf.predict(x\_test))

# Evaluation on Training Data

confusion\_matrix(y\_train, ada\_clf.predict(x\_train))

accuracy\_score(y\_train, ada\_clf.predict(x\_train))

############################# voting ##################################

# Import the required libraries

from sklearn import datasets, linear\_model, svm, neighbors, naive\_bayes

from sklearn.ensemble import VotingClassifier

from sklearn.metrics import accuracy\_score

import pandas as pd

import numpy as np

# Dummy variables

df.head()

df.info()

# Input and Output Split

predictors = df.loc[:, df.columns!=" Class variable"]

type(predictors)

target = df[" Class variable"]

type(target)

# Train Test partition of the data

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(predictors, target, test\_size = 0.2, random\_state=0)

# Instantiate the learners (classifiers)

learner\_1 = neighbors.KNeighborsClassifier(n\_neighbors=5)

learner\_2 = linear\_model.Perceptron(tol=1e-2, random\_state=0)

learner\_3 = svm.SVC(gamma=0.001)

# Instantiate the voting classifier

voting = VotingClassifier([('KNN', learner\_1),

('Prc', learner\_2),

('SVM', learner\_3)])

# Fit classifier with the training data

voting.fit(x\_train, y\_train)

# Predict the most voted class

hard\_predictions\_test = voting.predict(x\_test)

hard\_predictions\_train = voting.predict(x\_train)

# Accuracy of hard voting

print('Hard Voting:\n', confusion\_matrix(y\_test, hard\_predictions\_test))

print('Hard Voting:\n', accuracy\_score(y\_test, hard\_predictions\_test))

print('Hard Voting:\n', confusion\_matrix(y\_train, hard\_predictions\_train))

print('Hard Voting:\n', accuracy\_score(y\_train, hard\_predictions\_train))

#################

# Soft Voting #

# Instantiate the learners (classifiers)

learner\_4 = neighbors.KNeighborsClassifier(n\_neighbors = 5)

learner\_5 = naive\_bayes.GaussianNB()

learner\_6 = svm.SVC(gamma = 0.001, probability = True)

# Instantiate the voting classifier

voting = VotingClassifier([('KNN', learner\_4),

('NB', learner\_5),

('SVM', learner\_6)],

voting = 'soft')

# Fit classifier with the training data

voting.fit(x\_train, y\_train)

learner\_4.fit(x\_train, y\_train)

learner\_5.fit(x\_train, y\_train)

learner\_6.fit(x\_train, y\_train)

# Predict the most probable class

soft\_predictions\_test = voting.predict(x\_test)

soft\_predictions\_train = voting.predict(x\_train)

# Get the base learner predictions

predictions\_4 = learner\_4.predict(x\_test)

predictions\_5 = learner\_5.predict(x\_test)

predictions\_6 = learner\_6.predict(x\_test)

# Accuracies of base learners

print('L4:', accuracy\_score(y\_test, predictions\_4))

print('L5:', accuracy\_score(y\_test, predictions\_5))

print('L6:', accuracy\_score(y\_test, predictions\_6))

# Accuracy of Soft voting

print('soft Voting:\n', confusion\_matrix(y\_test, soft\_predictions\_test))

print('soft Voting:\n', accuracy\_score(y\_test, soft\_predictions\_test))

print('soft Voting:\n', confusion\_matrix(y\_train, soft\_predictions\_train))

print('soft Voting:\n', accuracy\_score(y\_train, soft\_predictions\_train))

#################################stacking################################

# Libraries and data loading

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.neural\_network import MLPClassifier

#from sklearn.naive\_bayes import GaussianNB

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import KFold

from sklearn import metrics

import numpy as np

import pandas as pd

from sklearn.preprocessing import LabelEncoder

df1=df.copy(deep=True)

# converting ouput variable to numeric binary format

lb = LabelEncoder()

df1[" Class variable"] =lb.fit\_transform(df1[" Class variable"])

# Input and Output Split

predictors = df1.loc[:, df1.columns!=" Class variable"]

type(predictors)

target = df1[" Class variable"]

type(target)

# Train Test partition of the data

from sklearn.model\_selection import train\_test\_split

train\_x, test\_x, train\_y, test\_y = train\_test\_split(predictors, target, test\_size = 0.2, random\_state=0)

#converting to nu numpy array

train\_x = train\_x.values

test\_x = test\_x.values

train\_y = train\_y.values

test\_y = test\_y.values

# Create the ensemble's base learners and meta learner

# Append base learners to a list

base\_learners = []

# KNN classifier model

knn = KNeighborsClassifier(n\_neighbors=2)

base\_learners.append(knn)

# Decision Tree Classifier model

dtr = DecisionTreeClassifier(max\_depth=4, random\_state=123456)

base\_learners.append(dtr)

# Multi Layered Perceptron classifier

mlpc = MLPClassifier(hidden\_layer\_sizes =(100, ), solver='lbfgs', random\_state=123456)

base\_learners.append(mlpc)

# Meta model using Logistic Regression

meta\_learner = LogisticRegression(solver='lbfgs')

# Create the training meta data

# Create variables to store meta data and the targets

meta\_data = np.zeros((len(base\_learners), len(train\_x )))

meta\_targets = np.zeros(len(train\_x))

# Create the cross-validation folds

KF = KFold(n\_splits = 5)

meta\_index = 0

for train\_indices, test\_indices in KF.split(train\_x):

# Train each learner on the K-1 folds and create meta data for the Kth fold

for i in range(len(base\_learners)):

learner = base\_learners[i]

learner.fit(train\_x[train\_indices], train\_y[train\_indices])

predictions = learner.predict\_proba(train\_x[test\_indices])[:,0]

meta\_data[i][meta\_index:meta\_index+len(test\_indices)] = predictions

meta\_targets[meta\_index:meta\_index+len(test\_indices)] = train\_y[test\_indices]

meta\_index += len(test\_indices)

# Transpose the meta data to be fed into the meta learner

meta\_data = meta\_data.transpose()

# Create the meta data for the test set and evaluate the base learners

test\_meta\_data = np.zeros((len(base\_learners), len(test\_x)))

base\_acc = []

for i in range(len(base\_learners)):

learner = base\_learners[i]

learner.fit(train\_x, train\_y)

predictions = learner.predict\_proba(test\_x)[:,0]

test\_meta\_data[i] = predictions

acc = metrics.accuracy\_score(test\_y, learner.predict(test\_x))

base\_acc.append(acc)

test\_meta\_data = test\_meta\_data.transpose()

# Fit the meta learner on the train set and evaluate it on the test set

meta\_learner.fit(meta\_data, meta\_targets)

ensemble\_predictions = meta\_learner.predict(test\_meta\_data)

acc = metrics.accuracy\_score(test\_y, ensemble\_predictions)

# Print the results

for i in range(len(base\_learners)):

learner = base\_learners[i]

print(f'{base\_acc[i]:.2f} {learner.\_\_class\_\_.\_\_name\_\_}')

print(f'{acc:.2f} Ensemble')

# best leanrner is DecisionTreeclassifier in ensamble model

# Accuracy of best leanrner

print('ensamble\_dtr:\n', confusion\_matrix(test\_y, dtr.predict(test\_x)))

print('ensamble\_dtr:\n', accuracy\_score(test\_y, dtr.predict(test\_x)))

print('ensamble\_dtr:\n', confusion\_matrix(train\_y, dtr.predict(x\_train)))

print('ensamble\_dtr:\n', accuracy\_score(train\_y, dtr.predict(x\_train)))

####### GridSearchCV #####

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import GridSearchCV

rf\_clf\_grid = RandomForestClassifier(n\_estimators=500, n\_jobs=1, random\_state=42,bootstrap=True)

param\_grid = {"max\_features": [4, 5, 6, 7, 8], "min\_samples\_split": [2, 3, 4, 5,6,7,8],"max\_leaf\_nodes":[4,5,6]}

grid\_search = GridSearchCV(rf\_clf\_grid, param\_grid, n\_jobs = -1, cv = 5, scoring = 'accuracy')

grid\_search.fit(x\_train, y\_train)

grid\_search.best\_params\_

cv\_rf\_clf\_grid = grid\_search.best\_estimator\_

from sklearn.metrics import accuracy\_score, confusion\_matrix

confusion\_matrix(y\_test, cv\_rf\_clf\_grid.predict(x\_test))

accuracy\_score(y\_test, cv\_rf\_clf\_grid.predict(x\_test))

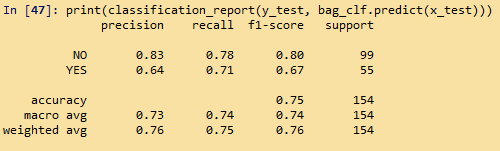
confusion\_matrix(y\_train, cv\_rf\_clf\_grid.predict(x\_train))

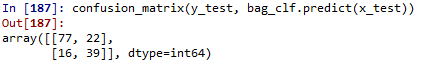
accuracy\_score(y\_train, cv\_rf\_clf\_grid.predict(x\_train))

**Summary :-**

**Bagging model:-**

**Evaluation on Test data :-**



****

**C:\Users\user\Documents\paint.png**

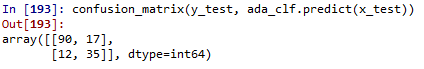
**Evaluation on Train data :-**

**C:\Users\user\Documents\paint.png**

**C:\Users\user\Documents\paint.png**

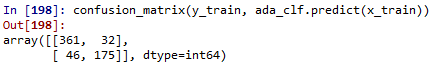
**Boosting**

**Evaluation on Test data :-**



C:\Users\user\Documents\paint.png

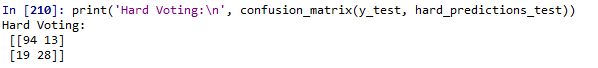
**Evaluation on Train data :-**



C:\Users\user\Documents\paint.png

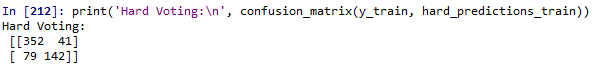
**Voting**

**Hard voting Evaluation on Test data :-**



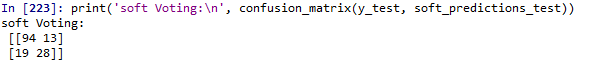
C:\Users\user\Documents\paint.png

**Hard voting Evaluation on Train data :-**



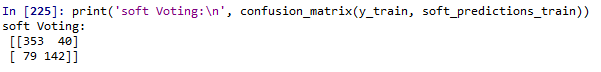
C:\Users\user\Documents\paint.png

**Soft voting Evaluation on Test data :-**



C:\Users\user\Documents\paint.png

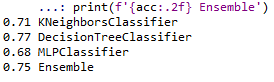
**Soft voting Evaluation on Train data :-**

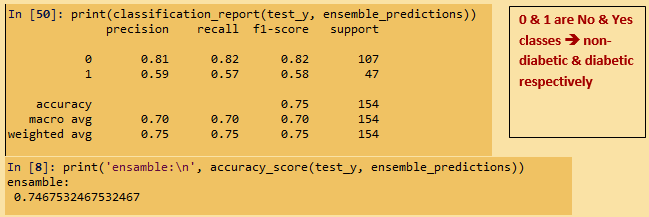


C:\Users\user\Documents\paint.png

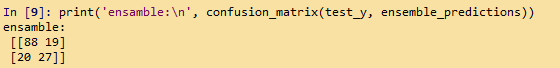
**Stacking:-**

**Evaluation on test data**



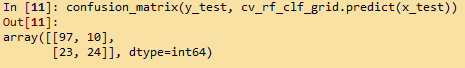


C:\Users\user\Documents\Figure_1.png



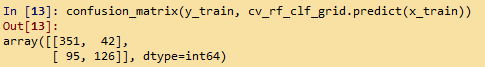
**Grid search cv:-**

**Evaluation on test data**



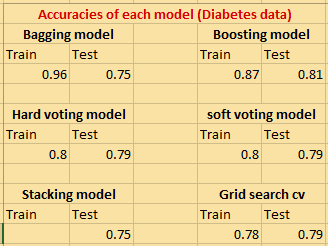
C:\Users\user\Documents\Figure_1.png

**Evaluation on train data**



C:\Users\user\Documents\Figure_1.png

**Over all analysis on each models:-**



Note:-

* ((((all models showing greater than “10” false negative conditions (Yes case predicted as No{here first row and column for No and second for Yes from “classification\_report. So for better performance need to go for further hyper parameter Tuning on each model and check the results. Model which showing better accuracy and least false negative cases is better.)))
* Compare to all other models grid search cv showing least accuracy difference between train and test data. That is no over fitting problem

**Business Benefit:-**

**Model will helps the doctors or any other analyst to predict the chances of a patient strucken by diabetes by analysing the given input variables without going for any expensive diagnosing medical tests.**

1. **Business Problem**

Most cancers form a lump called a tumour. But not all lumps are cancerous. Doctors extract a sample from the lump and examine it to find out if it’s cancer or not. Lumps that are not cancerous are called benign (be-NINE). Lumps that are cancerous are called malignant (muh-LIG-nunt). Obtaining incorrect results (false positives and false negatives) especially in a medical condition such as cancer is dangerous. So, perform Bagging, Boosting, Stacking, and Voting algorithms to increase model performance and provide your insights in the documentation.

* 1. **What is the business objective?**

Built a model toPredict cancer conditions by analysing the given input variables without going for expensive diagnosing medical tests.

* 1. **Are there any constraints?**

Maximize: model accuracy

Minimize: model complexity

Minimize: false negative cases of model out

**Python Code:-**

###############################Bagging##############################

import pandas as pd

df = pd.read\_csv("C://Users//user//Downloads//ensamble//Tumor\_Ensemble.csv")

# Dummy variables

df.head()

df.info()

# Input and Output Split

predictors = df.loc[:, df.columns!="diagnosis"]

type(predictors)

target = df["diagnosis"]

type(target)

# Train Test partition of the data

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(predictors, target, test\_size = 0.2, random\_state=0)

from sklearn import tree

clftree = tree.DecisionTreeClassifier(max\_depth=7)

from sklearn.ensemble import BaggingClassifier

bag\_clf = BaggingClassifier(base\_estimator = clftree, n\_estimators = 500,

bootstrap = True, n\_jobs = 1, random\_state = 42)

bag\_clf.fit(x\_train, y\_train,)

from sklearn.metrics import accuracy\_score, confusion\_matrix

# Evaluation on Testing Data

confusion\_matrix(y\_test, bag\_clf.predict(x\_test))

accuracy\_score(y\_test, bag\_clf.predict(x\_test))

# Evaluation on Training Data

confusion\_matrix(y\_train, bag\_clf.predict(x\_train))

accuracy\_score(y\_train, bag\_clf.predict(x\_train))

############################boosting(adaboost)##########################

# applied adaboost

import pandas as pd

# Input and Output Split

# Input and Output Split

predictors = df.loc[:, df.columns!="diagnosis"]

type(predictors)

target = df["diagnosis"]

type(target)

# Train Test partition of the data

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(predictors, target, test\_size = 0.2, random\_state=0)

# Refer to the links

# https://scikit-learn.org/stable/modules/classes.html#module-sklearn.ensemble

# https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html#sklearn.ensemble.AdaBoostClassifier

from sklearn.ensemble import AdaBoostClassifier

ada\_clf = AdaBoostClassifier(learning\_rate = 0.5, n\_estimators = 500)

ada\_clf.fit(x\_train, y\_train)

from sklearn.metrics import accuracy\_score, confusion\_matrix

# Evaluation on Testing Data

confusion\_matrix(y\_test, ada\_clf.predict(x\_test))

accuracy\_score(y\_test, ada\_clf.predict(x\_test))

# Evaluation on Training Data

accuracy\_score(y\_train, ada\_clf.predict(x\_train))

accuracy\_score(y\_train, bag\_clf.predict(x\_train))

############################# voting ##################################

# Import the required libraries

from sklearn import datasets, linear\_model, svm, neighbors, naive\_bayes

from sklearn.ensemble import VotingClassifier

from sklearn.metrics import accuracy\_score

import pandas as pd

import numpy as np

# Dummy variables

df.head()

df.info()

# Input and Output Split

predictors = df.loc[:, df.columns!=" Class variable"]

type(predictors)

target = df[" Class variable"]

type(target)

# Train Test partition of the data

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(predictors, target, test\_size = 0.2, random\_state=0)

# Instantiate the learners (classifiers)

learner\_1 = neighbors.KNeighborsClassifier(n\_neighbors=5)

learner\_2 = linear\_model.Perceptron(tol=1e-2, random\_state=0)

learner\_3 = svm.SVC(gamma=0.001)

# Instantiate the voting classifier

voting = VotingClassifier([('KNN', learner\_1),

('Prc', learner\_2),

('SVM', learner\_3)])

# Fit classifier with the training data

voting.fit(x\_train, y\_train)

# Predict the most voted class

hard\_predictions = voting.predict(x\_test)

# Accuracy of hard voting

print('Hard Voting:', accuracy\_score(y\_test, hard\_predictions))

#################

# Soft Voting #

# Instantiate the learners (classifiers)

learner\_4 = neighbors.KNeighborsClassifier(n\_neighbors = 5)

learner\_5 = naive\_bayes.GaussianNB()

learner\_6 = svm.SVC(gamma = 0.001, probability = True)

# Instantiate the voting classifier

voting = VotingClassifier([('KNN', learner\_4),

('NB', learner\_5),

('SVM', learner\_6)],

voting = 'soft')

# Fit classifier with the training data

voting.fit(x\_train, y\_train)

learner\_4.fit(x\_train, y\_train)

learner\_5.fit(x\_train, y\_train)

learner\_6.fit(x\_train, y\_train)

# Predict the most probable class

soft\_predictions = voting.predict(x\_test)

# Get the base learner predictions

predictions\_4 = learner\_4.predict(x\_test)

predictions\_5 = learner\_5.predict(x\_test)

predictions\_6 = learner\_6.predict(x\_test)

# Accuracies of base learners

print('L4:', accuracy\_score(y\_test, predictions\_4))

print('L5:', accuracy\_score(y\_test, predictions\_5))

print('L6:', accuracy\_score(y\_test, predictions\_6))

# Accuracy of Soft voting

print('Soft Voting:', accuracy\_score(y\_test, soft\_predictions))

#################################stacking################################

# Libraries and data loading

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.neural\_network import MLPClassifier

#from sklearn.naive\_bayes import GaussianNB

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import KFold

from sklearn import metrics

import numpy as np

import pandas as pd

from sklearn.preprocessing import LabelEncoder

df1=df.copy(deep=True)

# Input and Output Split

predictors = df.loc[:, df.columns!="diagnosis"]

type(predictors)

target = df["diagnosis"]

type(target)

# Train Test partition of the data

from sklearn.model\_selection import train\_test\_split

train\_x, test\_x, train\_y, test\_y = train\_test\_split(predictors, target, test\_size = 0.2, random\_state=0)

#converting to nu numpy array

train\_x = train\_x.values

test\_x = test\_x.values

train\_y = train\_y.values

test\_y = test\_y.values

# Create the ensemble's base learners and meta learner

# Append base learners to a list

base\_learners = []

# KNN classifier model

knn = KNeighborsClassifier(n\_neighbors=2)

base\_learners.append(knn)

# Decision Tree Classifier model

dtr = DecisionTreeClassifier(max\_depth=4, random\_state=123456)

base\_learners.append(dtr)

# Multi Layered Perceptron classifier

mlpc = MLPClassifier(hidden\_layer\_sizes =(100, ), solver='lbfgs', random\_state=123456)

base\_learners.append(mlpc)

# Meta model using Logistic Regression

meta\_learner = LogisticRegression(solver='lbfgs')

# Create the training meta data

# Create variables to store meta data and the targets

meta\_data = np.zeros((len(base\_learners), len(train\_x )))

meta\_targets = np.zeros(len(train\_x))

# Create the cross-validation folds

KF = KFold(n\_splits = 5)

meta\_index = 0

for train\_indices, test\_indices in KF.split(train\_x):

# Train each learner on the K-1 folds and create meta data for the Kth fold

for i in range(len(base\_learners)):

learner = base\_learners[i]

learner.fit(train\_x[train\_indices], train\_y[train\_indices])

predictions = learner.predict\_proba(train\_x[test\_indices])[:,0]

meta\_data[i][meta\_index:meta\_index+len(test\_indices)] = predictions

meta\_targets[meta\_index:meta\_index+len(test\_indices)] = train\_y[test\_indices]

meta\_index += len(test\_indices)

# Transpose the meta data to be fed into the meta learner

meta\_data = meta\_data.transpose()

# Create the meta data for the test set and evaluate the base learners

test\_meta\_data = np.zeros((len(base\_learners), len(test\_x)))

base\_acc = []

for i in range(len(base\_learners)):

learner = base\_learners[i]

learner.fit(train\_x, train\_y)

predictions = learner.predict\_proba(test\_x)[:,0]

test\_meta\_data[i] = predictions

acc = metrics.accuracy\_score(test\_y, learner.predict(test\_x))

base\_acc.append(acc)

test\_meta\_data = test\_meta\_data.transpose()

# Fit the meta learner on the train set and evaluate it on the test set

meta\_learner.fit(meta\_data, meta\_targets)

ensemble\_predictions = meta\_learner.predict(test\_meta\_data)

acc = metrics.accuracy\_score(test\_y, ensemble\_predictions)

# Print the results

for i in range(len(base\_learners)):

learner = base\_learners[i]

print(f'{base\_acc[i]:.2f} {learner.\_\_class\_\_.\_\_name\_\_}')

print(f'{acc:.2f} Ensemble')

####### GridSearchCV #####

from sklearn.model\_selection import GridSearchCV

rf\_clf\_grid = RandomForestClassifier(n\_estimators=500, n\_jobs=1, random\_state=42,bootstrap=True)

param\_grid = {"max\_features": [4, 5, 6, 7, 8, 9, 10], "min\_samples\_split": [2, 3, 4, 5,6,7,8,9,10],"max\_leaf\_nodes":[4,5,6]}

grid\_search = GridSearchCV(rf\_clf\_grid, param\_grid, n\_jobs = -1, cv = 5, scoring = 'accuracy')

grid\_search.fit(x\_train, y\_train)

grid\_search.best\_params\_

cv\_rf\_clf\_grid = grid\_search.best\_estimator\_

from sklearn.metrics import accuracy\_score, confusion\_matrix

confusion\_matrix(y\_test, cv\_rf\_clf\_grid.predict(x\_test))

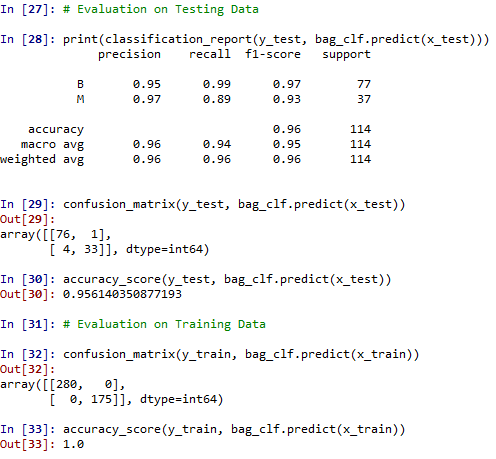
accuracy\_score(y\_test, cv\_rf\_clf\_grid.predict(x\_test))

confusion\_matrix(y\_train, cv\_rf\_clf\_grid.predict(x\_train))

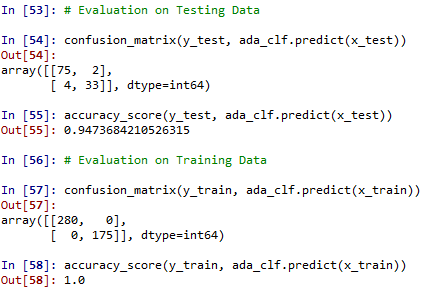
accuracy\_score(y\_train, cv\_rf\_clf\_grid.predict(x\_train))

**Summary :-**

**Bagging model:-**

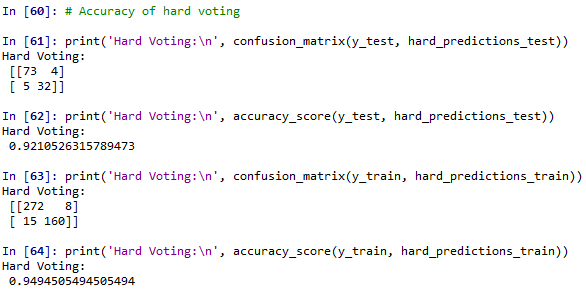
****

**Boosting(adaboost):**

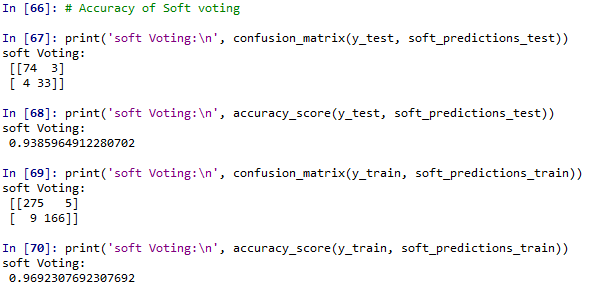
****

**Voting**

**Hard voting :-**

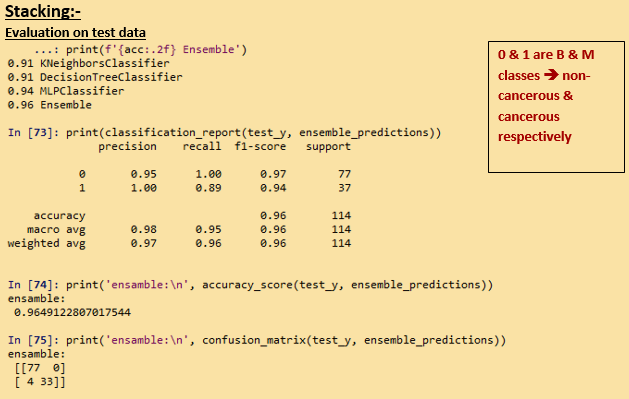


**soft voting :-**

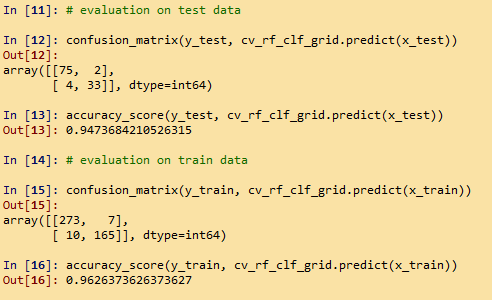


**Stacking:-**

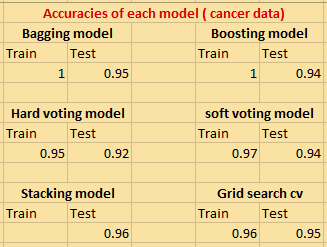
**Evaluation on test data**

****

**Grid search cv:-**

****

**Over all analysis on each models:-**

****

Note:-

* ((((stacking model showing only zero false negative conditions (M case predicted as B{here first row and column for B and second for M from “classification\_report”}) compare to all other model and 96 % accuracy for test data. In this problem for prediction of cancer cells its better to choose those model showing less false negative cases and better accuracy)))
* Bagging and Boosting models showing over fitting problem.
* Voting models performance is also better in terms of better accuracy and have no over fitting problem
* Grid search cv showing better performannce in the case of better accuracy and least diff between test and train accuracies. But there have presence of FN values in test and train preddictions.

**Business Benefit:-**

**Model will helps the doctors or any other analyst to predict the chances of a patient strucken by cancer decease by analysing the given input variables without going for expensive diagnosing tests.**

1. **Business Problem**

A sample of global companies and their ratings are given for the cocoa bean production along with the location of the beans being used. Identify the important features in the analysis and accurately classify the companies based on their ratings and draw insights from the data. Build ensemble models such as Bagging, Boosting, Stacking, and Voting on the dataset given.

* 1. **What is the business objective?**

Build a model to predict the rating class of cocoa bean production companies according to input features given

* 1. **Are there any constraints?**

Maximize: model accuracy

Minimize: model complexity

**Python Code:-**

################################ Bagging##############################

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

import seaborn as sn

df = pd.read\_excel("C://Users//user//Downloads//ensamble//Coca\_Rating\_Ensemble.xlsx")

# Dummy variables

df.head()

df.isnull().sum()

df.dropna()

df.columns

df.info()

# droping "Bean\_Type " & "Origin" since having null value masking problm

df = df.iloc[:, 0:7]

#converting input and output columns into categorical

lb = LabelEncoder()

df["Name"]= lb.fit\_transform(df["Name"])

df["Company\_Location"] = lb.fit\_transform(df["Company\_Location"])

df["Company"] = lb.fit\_transform(df["Company"])

## discretize ouput column

df['Rating'].describe() # max value = 5

bins=[0,1,2,3,4,5]

group\_names= ['low\_[0-1]','medium\_(1-2]','high\_(2-3]','very high\_(3-4]','extra\_very\_high\_(4-5]'] # (a,b] => a not included; but b included

df['Rating']= pd.cut(df['Rating'],bins, labels = group\_names)

# Input and Output Split

predictors = df.loc[:, df.columns!="Rating"]

type(predictors)

target = df["Rating"]

type(target)

# Train Test partition of the data

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(predictors, target, test\_size = 0.2, random\_state=0)

from sklearn import tree

clftree = tree.DecisionTreeClassifier(max\_depth=7)

from sklearn.ensemble import BaggingClassifier

bag\_clf = BaggingClassifier(base\_estimator = clftree, n\_estimators = 500,

bootstrap = True, n\_jobs = 1, random\_state = 42)

bag\_clf.fit(x\_train, y\_train,)

from sklearn.metrics import accuracy\_score, confusion\_matrix

# Evaluation on Testing Data

confusion\_matrix(y\_test, bag\_clf.predict(x\_test))

accuracy\_score(y\_test, bag\_clf.predict(x\_test))

# Evaluation on Training Data

confusion\_matrix(y\_train, bag\_clf.predict(x\_train))

accuracy\_score(y\_train, bag\_clf.predict(x\_train))

##################################Boosting###############################

# applied adaboost

import pandas as pd

# Input and Output Split

predictors = df.loc[:, df.columns!="Company"]

type(predictors)

target = df["Company"]

type(target)

# Train Test partition of the data

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(predictors, target, test\_size = 0.2, random\_state=0)

# Refer to the links

# https://scikit-learn.org/stable/modules/classes.html#module-sklearn.ensemble

# https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html#sklearn.ensemble.AdaBoostClassifier

from sklearn.ensemble import AdaBoostClassifier

ada\_clf = AdaBoostClassifier(learning\_rate = 0.5, n\_estimators = 500)

ada\_clf.fit(x\_train, y\_train)

from sklearn.metrics import accuracy\_score, confusion\_matrix

# Evaluation on Testing Data

confusion\_matrix(y\_test, ada\_clf.predict(x\_test))

accuracy\_score(y\_test, ada\_clf.predict(x\_test))

# Evaluation on Training Data

accuracy\_score(y\_train, ada\_clf.predict(x\_train))

accuracy\_score(y\_train, bag\_clf.predict(x\_train))

############################# voting ##################################

# Import the required libraries

from sklearn import datasets, linear\_model, svm, neighbors, naive\_bayes

from sklearn.ensemble import VotingClassifier

from sklearn.metrics import accuracy\_score

import pandas as pd

import numpy as np

# Dummy variables

df.head()

df.info()

# Input and Output Split

predictors = df.loc[:, df.columns!=" Class variable"]

type(predictors)

target = df[" Class variable"]

type(target)

# Train Test partition of the data

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(predictors, target, test\_size = 0.2, random\_state=0)

# Instantiate the learners (classifiers)

learner\_1 = neighbors.KNeighborsClassifier(n\_neighbors=5)

learner\_2 = linear\_model.Perceptron(tol=1e-2, random\_state=0)

learner\_3 = svm.SVC(gamma=0.001)

# Instantiate the voting classifier

voting = VotingClassifier([('KNN', learner\_1),

('Prc', learner\_2),

('SVM', learner\_3)])

# Fit classifier with the training data

voting.fit(x\_train, y\_train)

# Predict the most voted class

hard\_predictions = voting.predict(x\_test)

# Accuracy of hard voting

print('Hard Voting:', accuracy\_score(y\_test, hard\_predictions))

#################

# Soft Voting #

# Instantiate the learners (classifiers)

learner\_4 = neighbors.KNeighborsClassifier(n\_neighbors = 5)

learner\_5 = naive\_bayes.GaussianNB()

learner\_6 = svm.SVC(gamma = 0.001, probability = True)

# Instantiate the voting classifier

voting = VotingClassifier([('KNN', learner\_4),

('NB', learner\_5),

('SVM', learner\_6)],

voting = 'soft')

# Fit classifier with the training data

voting.fit(x\_train, y\_train)

learner\_4.fit(x\_train, y\_train)

learner\_5.fit(x\_train, y\_train)

learner\_6.fit(x\_train, y\_train)

# Predict the most probable class

soft\_predictions = voting.predict(x\_test)

# Get the base learner predictions

predictions\_4 = learner\_4.predict(x\_test)

predictions\_5 = learner\_5.predict(x\_test)

predictions\_6 = learner\_6.predict(x\_test)

# Accuracies of base learners

print('L4:', accuracy\_score(y\_test, predictions\_4))

print('L5:', accuracy\_score(y\_test, predictions\_5))

print('L6:', accuracy\_score(y\_test, predictions\_6))

# Accuracy of Soft voting

print('Soft Voting:', accuracy\_score(y\_test, soft\_predictions))

#################################stacking################################

# Libraries and data loading

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.neural\_network import MLPClassifier

#from sklearn.naive\_bayes import GaussianNB

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import KFold

from sklearn import metrics

import numpy as np

import pandas as pd

from sklearn.preprocessing import LabelEncoder

df1=df.copy(deep=True)

# converting ouput variable to numeric binary format

lb = LabelEncoder()

df1[" Class variable"] =lb.fit\_transform(df1[" Class variable"])

# Input and Output Split

predictors = df1.loc[:, df1.columns!=" Class variable"]

type(predictors)

target = df1[" Class variable"]

type(target)

# Train Test partition of the data

from sklearn.model\_selection import train\_test\_split

train\_x, test\_x, train\_y, test\_y = train\_test\_split(predictors, target, test\_size = 0.2, random\_state=0)

#converting to nu numpy array

train\_x = train\_x.values

test\_x = test\_x.values

train\_y = train\_y.values

test\_y = test\_y.values

# Create the ensemble's base learners and meta learner

# Append base learners to a list

base\_learners = []

# KNN classifier model

knn = KNeighborsClassifier(n\_neighbors=2)

base\_learners.append(knn)

# Decision Tree Classifier model

dtr = DecisionTreeClassifier(max\_depth=4, random\_state=123456)

base\_learners.append(dtr)

# Multi Layered Perceptron classifier

mlpc = MLPClassifier(hidden\_layer\_sizes =(100, ), solver='lbfgs', random\_state=123456)

base\_learners.append(mlpc)

# Meta model using Logistic Regression

meta\_learner = LogisticRegression(solver='lbfgs')

# Create the training meta data

# Create variables to store meta data and the targets

meta\_data = np.zeros((len(base\_learners), len(train\_x )))

meta\_targets = np.zeros(len(train\_x))

# Create the cross-validation folds

KF = KFold(n\_splits = 5)

meta\_index = 0

for train\_indices, test\_indices in KF.split(train\_x):

# Train each learner on the K-1 folds and create meta data for the Kth fold

for i in range(len(base\_learners)):

learner = base\_learners[i]

learner.fit(train\_x[train\_indices], train\_y[train\_indices])

predictions = learner.predict\_proba(train\_x[test\_indices])[:,0]

meta\_data[i][meta\_index:meta\_index+len(test\_indices)] = predictions

meta\_targets[meta\_index:meta\_index+len(test\_indices)] = train\_y[test\_indices]

meta\_index += len(test\_indices)

# Transpose the meta data to be fed into the meta learner

meta\_data = meta\_data.transpose()

# Create the meta data for the test set and evaluate the base learners

test\_meta\_data = np.zeros((len(base\_learners), len(test\_x)))

base\_acc = []

for i in range(len(base\_learners)):

learner = base\_learners[i]

learner.fit(train\_x, train\_y)

predictions = learner.predict\_proba(test\_x)[:,0]

test\_meta\_data[i] = predictions

acc = metrics.accuracy\_score(test\_y, learner.predict(test\_x))

base\_acc.append(acc)

test\_meta\_data = test\_meta\_data.transpose()

# Fit the meta learner on the train set and evaluate it on the test set

meta\_learner.fit(meta\_data, meta\_targets)

ensemble\_predictions = meta\_learner.predict(test\_meta\_data)

acc = metrics.accuracy\_score(test\_y, ensemble\_predictions)

# Print the results

for i in range(len(base\_learners)):

learner = base\_learners[i]

print(f'{base\_acc[i]:.2f} {learner.\_\_class\_\_.\_\_name\_\_}')

print(f'{acc:.2f} Ensemble')

####### GridSearchCV #####

from sklearn.model\_selection import GridSearchCV

rf\_clf\_grid = RandomForestClassifier(n\_estimators=500, n\_jobs=1, random\_state=42,bootstrap=True)

param\_grid = {"max\_features": [4, 5, 6, 7, 8, 9, 10], "min\_samples\_split": [2, 3, 4, 5,6,7,8,9,10],"max\_leaf\_nodes":[4,5,6]}

grid\_search = GridSearchCV(rf\_clf\_grid, param\_grid, n\_jobs = -1, cv = 5, scoring = 'accuracy')

grid\_search.fit(x\_train, y\_train)

grid\_search.best\_params\_

cv\_rf\_clf\_grid = grid\_search.best\_estimator\_

from sklearn.metrics import accuracy\_score, confusion\_matrix

confusion\_matrix(y\_test, cv\_rf\_clf\_grid.predict(x\_test))

accuracy\_score(y\_test, cv\_rf\_clf\_grid.predict(x\_test))

confusion\_matrix(y\_train, cv\_rf\_clf\_grid.predict(x\_train))

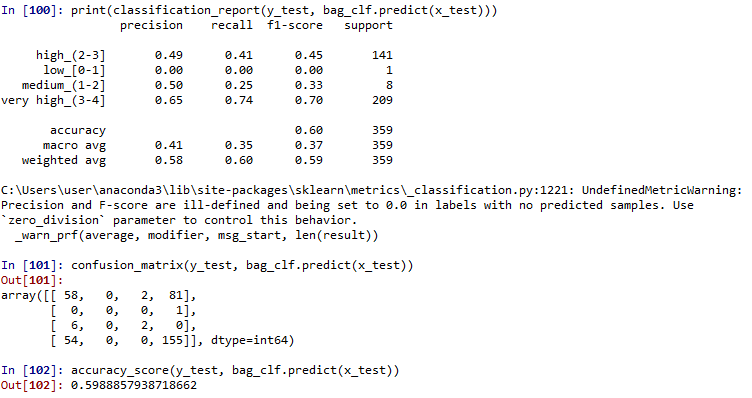
accuracy\_score(y\_train, cv\_rf\_clf\_grid.predict(x\_train))

**Summary :-**

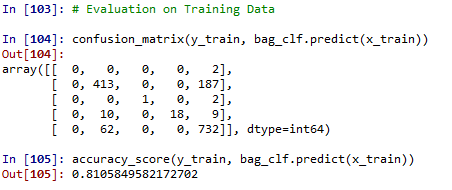
**((((((((( all the models predictions have least accuracy because that we eliminate(drop) two important input variable columns due to masking missing value problem)))))))**

**Bagging model:-**

**Evaluation on Test data :-**

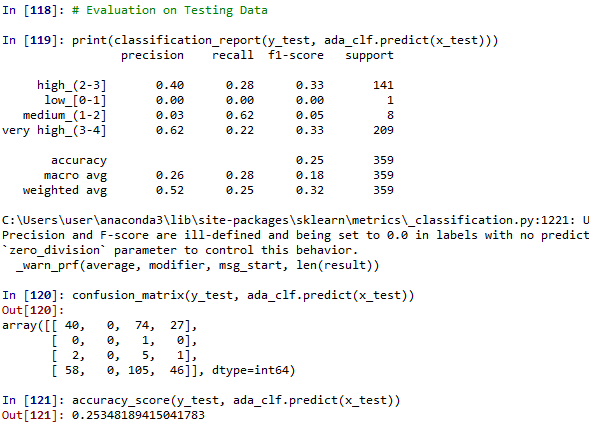
****

**Evaluation on Train data :-**

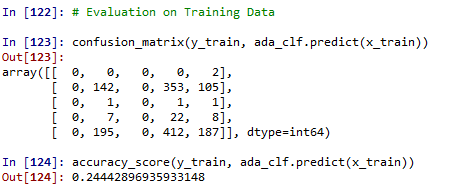


**Boosting**

**Evaluation on Test data :-**

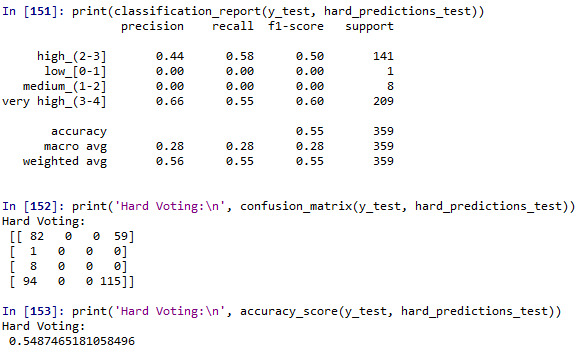


**Evaluation on Train data :-**

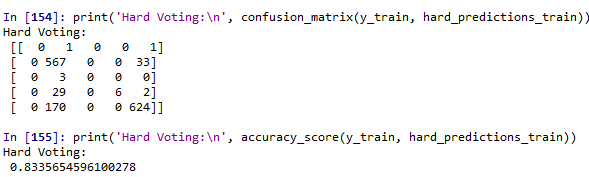


**Voting:-**

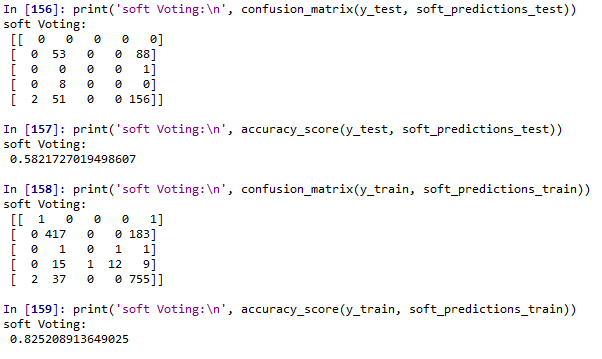
**Hard voting Evaluation on Test data :-**



**Hard voting Evaluation on Train data :-**

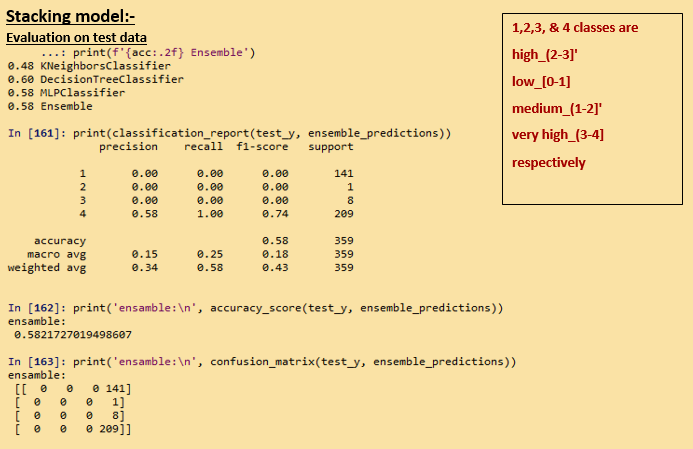


**Soft voting Evaluation on Test data :-**

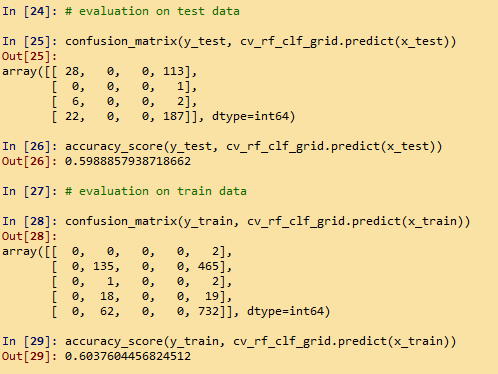


**Stacking model:-**

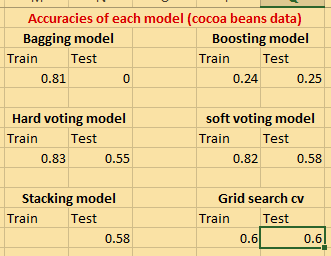
**Evaluation on test data**



**Grid search cv:-**

****

**Over all analysis on each models:-**

****

**Note:-**

* Best result is given by Grid search model (better accuracies for test and train data and having least difference between the accuracies). Means no over fitting problem. But accuracy values are not much better. That are considerable only
* The second best model rather than Grid search cv is Bagging model. That showing considerable accuray and least over fit problem in compare to other model.

**((((((((( all the results have least accuracy because that we eliminate(drop) two important input variable columns due to masking missing value problem)))))))**

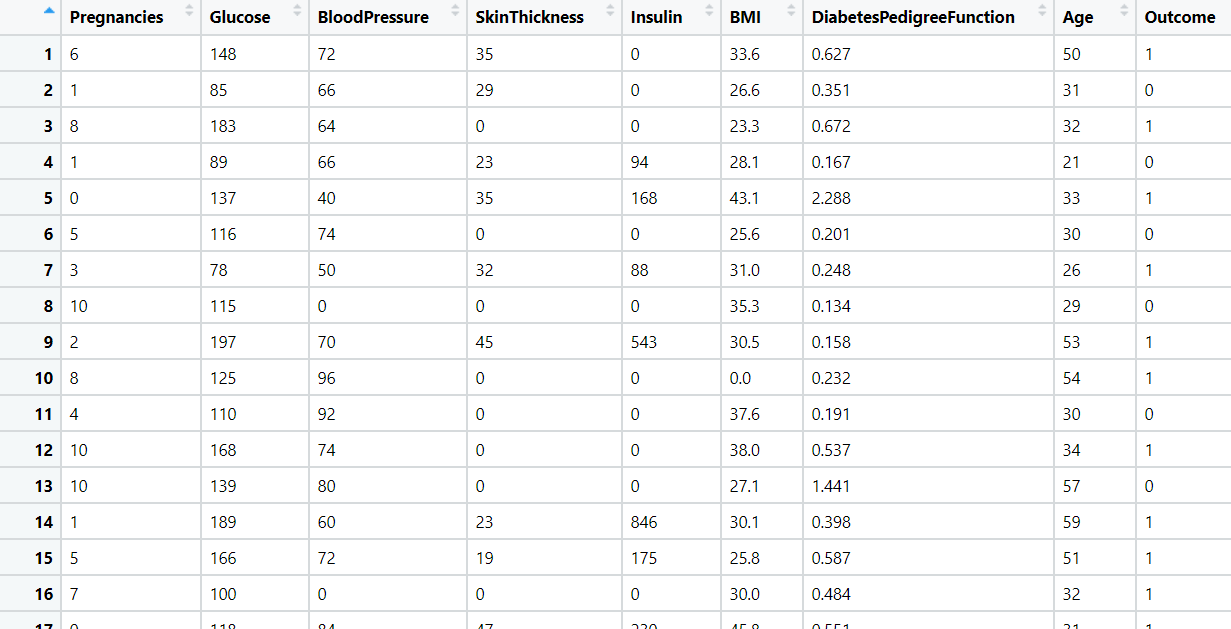
**Business benefit:-**

It will helps the business clients to predict how will be the customers rating thereby customers response to the cocoa beans if they gonna produce it based on by analysing the given input variables like location, type..etc.

For new start up it will helps to produce the best cocoa according to cutomers wishes. And for currently running producers by analying the data helps them going for changes that need to enforce in the cuurent production inorder to catch up customers attraction.

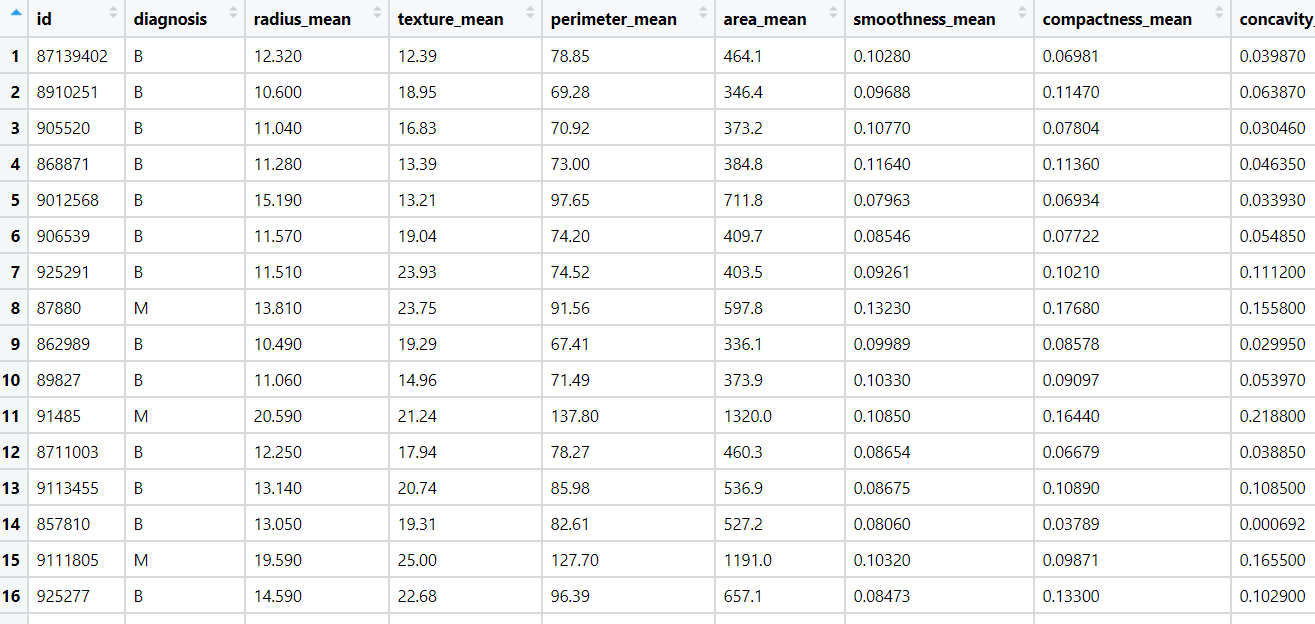
**Problem Statements:**

1. Given is the diabetes dataset. Build an ensemble model to correctly classify the outcome variable and improve your model prediction by using GridSearchCV. You must apply Bagging, Boosting, Stacking, and Voting on the dataset.



1. Most cancers form a lump called a tumour. But not all lumps are cancerous. Doctors extract a sample from the lump and examine it to find out if it’s cancer or not. Lumps that are not cancerous are called benign (be-NINE). Lumps that are cancerous are called malignant (muh-LIG-nunt). Obtaining incorrect results (false positives and false negatives) especially in a medical condition such as cancer is dangerous. So, perform Bagging, Boosting, Stacking, and Voting algorithms to increase model performance and provide your insights in the documentation.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

1. A sample of global companies and their ratings are given for the cocoa bean production along with the location of the beans being used. Identify the important features in the analysis and accurately classify the companies based on their ratings and draw insights from the data. Build ensemble models such as Bagging, Boosting, Stacking, and Voting on the dataset given.

**A screenshot of a computer

Description automatically generated**

1. Data privacy is always an important factor to safeguard their customers' details. For this, password strength is an important metric to track. Build an ensemble model to classify the user’s password strength.A screenshot of a cell phone

   Description automatically generated

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated