National University of Computer & Emerging Sciences, Karachi

Fall 2024, LAB MANUAL – 05

ENSEMBLE LEARNING

COURSE CODE :	**AL3002**
INSTRUCTOR :	**Usama Bin Umar**

OBJECTIVE

1. Ensemble Techniques
2. How to apply Voting Classifier
3. Improving Accuracies using Ensemble
4. Random Forest
5. AdaBoost
6. XGBoost
7. Hard, Soft and Weighted Average Voting

ENSEMBLE LEARNING:

“Unity is strength”. This old saying expresses pretty well the underlying idea that rules the very powerful “ensemble methods” in machine learning. Roughly, ensemble learning methods, that often trust the top rankings of many machine learning competitions (including Kaggle’s competitions), are based on the hypothesis that combining multiple models together can often produce a much more powerful model.

Ensemble learning is a machine learning paradigm where multiple models (often called “weak learners”) are trained to solve the same problem and combined to get better results. The main hypothesis is that when weak models are correctly combined we can obtain more accurate and/or robust models

In ensemble learning theory, we call weak learners (or base models) models that can be used as building blocks for designing more complex models by combining several of them. Most of the time, these basics models perform not so well by themselves either because they have a high bias (low degree of freedom models, for example) or because they have too much variance to be robust (high degree of freedom models, for example). Then, the idea of ensemble methods is to try reducing bias and/or variance of such weak learners by combining several of them together in order to create a strong learner (or ensemble model) that achieves better performances.

SIMPLE ENSEMBLE TECHNIQUES

In this section, we will look at a few simple but powerful techniques, namely:

Max Voting, Averaging

Weighted Averaging

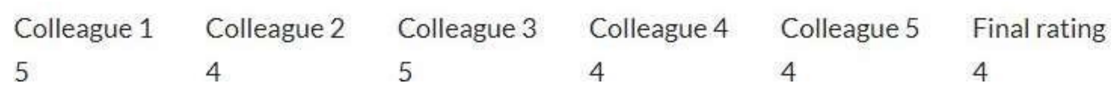
MAX VOTINNG:

The max voting method is generally used for classification problems. In this technique, multiple models are used to make predictions for each data point. The predictions by each model are considered as a 'vote. The predictions which we get from the majority of the models are used as the final prediction.

For example, when you asked 5 of your colleagues to rate your movie (out of 5); we'll assume three of them ated it as 4 while two of them gave it a 5. Since the majority gave a rating of 4, the final rating will be taken a

4. You can consider this as taking the mode of all the predictions.

The result of max voting would be something like this:



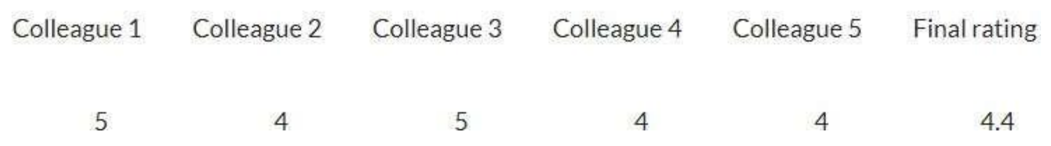
Colleague 1	Colleague 2	Colleague 3	Colleague 4	Colleague 5	Final rating
5	4	5	4	4	4

AVERAGING:

Similar to the max voting technique, multiple predictions are made for each data point in averaging. In this method, we take an average of predictions from all the models and use it to make the final prediction. Averaging can be used for making predictions in regression problems or while calculating probabilities for classification problems.

For example, in the below case, the averaging method would take the average of all the values.

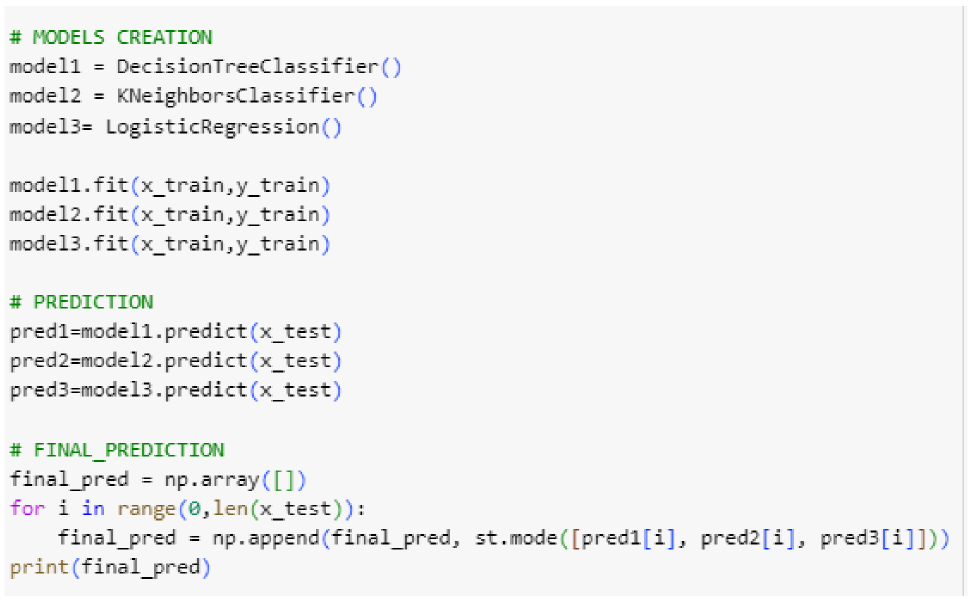
i.e. (5+4+5+4+4)/5 = 4.4



Colleague 1	Colleague 2	Colleague 3	Colleague 4	Colleague 5	Final rating
5	4	5	4	4	4.4

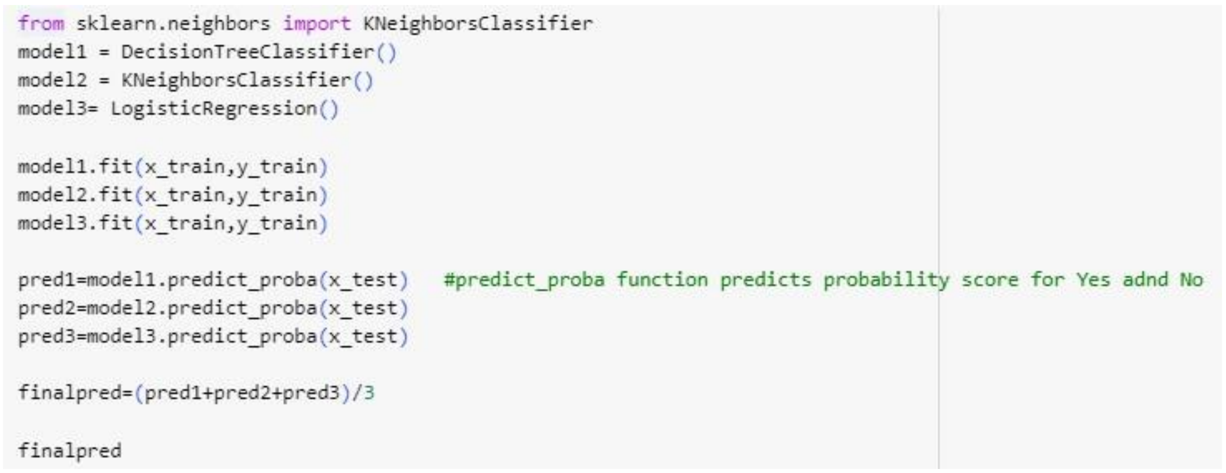
VOTING CLASSIFIER

MAX POOLING



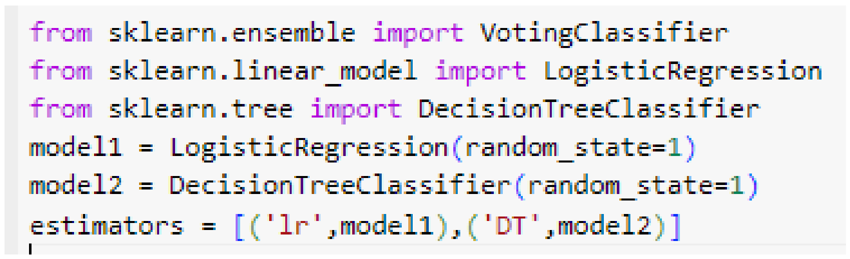
```
# MODELS CREATION
model1 = DecisionTreeClassifier()
model2 = KNeighborsClassifier()
model3= LogisticRegression()

model1.fit(x_train,y_train)
model2.fit(x_train,y_train)
model3.fit(x_train,y_train)

# PREDICTION
pred1=model1.predict(x_test)
pred2=model2.predict(x_test)
pred3=model3.predict(x_test)

# FINAL_PREDICTION
final_pred = np.array([])
for i in range(0,len(x_test)):
    final_pred = np.append(final_pred, st.mode([pred1[i], pred2[i], pred3[i]]))
print(final_pred)
```

AVERAGING



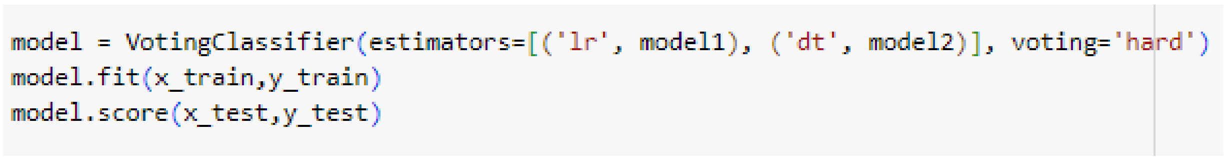
```
from sklearn.neighbors import KNeighborsClassifier
model1 = DecisionTreeClassifier()
model2 = KNeighborsClassifier()
model3= LogisticRegression()

model1.fit(x_train,y_train)
model2.fit(x_train,y_train)
model3.fit(x_train,y_train)

pred1=model1.predict_proba(x_test)   #predict_proba function predicts probability score for Yes adnd No
pred2=model2.predict_proba(x_test)
pred3=model3.predict_proba(x_test)

finalpred=(pred1+pred2+pred3)/3

finalpred
```

VOTING CLASSIFIER USING SKLEARN



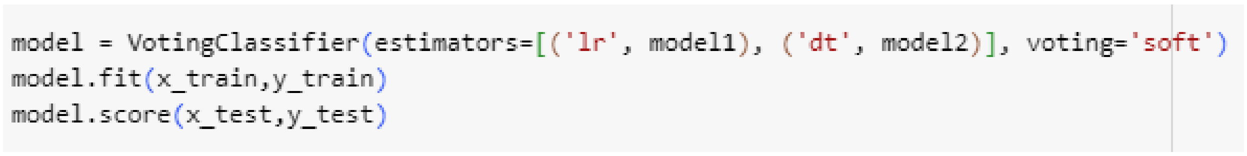
```
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
model1 = LogisticRegression(random_state=1)
model2 = DecisionTreeClassifier(random_state=1)
estimators = [('lr',model1),('DT',model2)]
```

HARD VOTING



```
model = VotingClassifier(estimators=[('lr', model1), ('dt', model2)], voting='hard')
model.fit(x_train,y_train)
model.score(x_test,y_test)
```

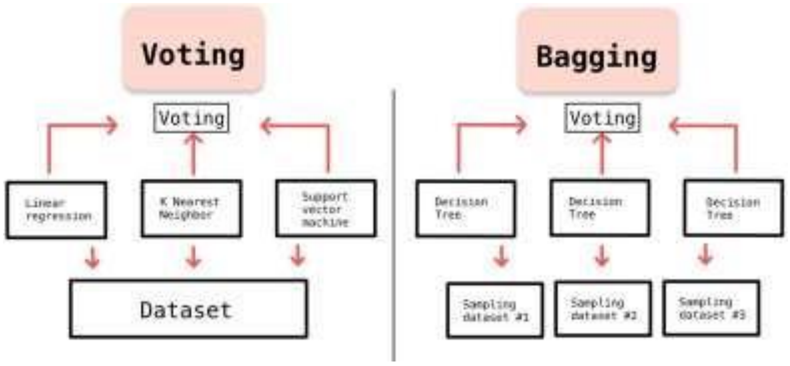
SOFT VOTING



```
model = VotingClassifier(estimators=[('lr', model1), ('dt', model2)], voting='soft')
model.fit(x_train,y_train)
model.score(x_test,y_test)
```

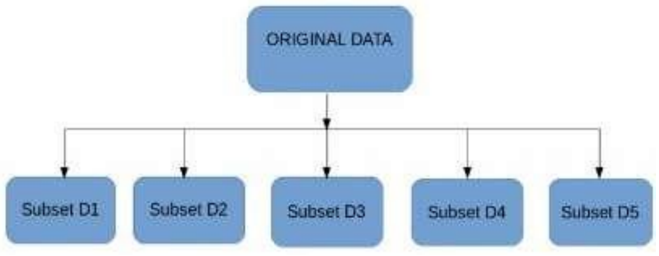
WEIGHTED AVERAGING

```
for i in range(1,4):
    for j in range(1,4):
      vc = VotingClassifier(estimators=estimators,voting='soft',weights=[i,j])
      vc.fit(x_train,y_train)
      x = vc.score(x_test,y_test)
      print("for i={},j={}".format(i,j),np.round(np.mean(x),2))
```



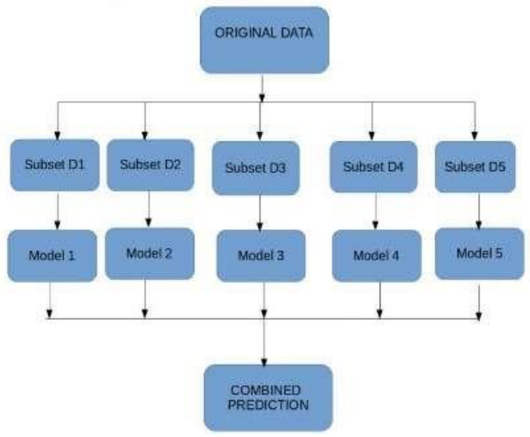
BAGGING ALGORITHMS

The idea behind bagging is combining the results of multiple models (for instance, all decision trees) to get a generalized result. Here's a question: If you create all the models on the same set of data and combine it, will it be useful? There is a high chance that these models will give the same result since they are getting the same input. So how can we solve this problem? One of the techniques is bootstrapping.
Bootstrapping is a sampling technique in which we create subsets of observations from the original dataset, with replacement. The size of the subsets is the same as the size of the original set.
Bagging (or Bootstrap Aggregating) technique uses these subsets (bags) to get a fair idea of the distribution (complete set). The size of subsets created for bagging may be less than the original set.



1. Multiple subsets are created from the original dataset, selecting observations with replacement.

2. A base model (weak model) is created on each of these subsets.
3. The models run in parallel and are independent of each other.
4. The final predictions are determined by combining the predictions from all the models.

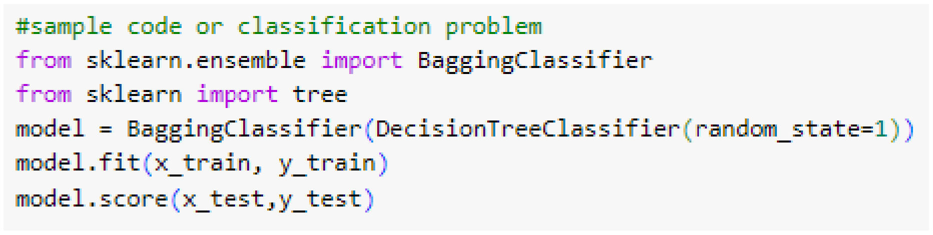


BAGGING ALGORITHMS:

Bagging • meta-estimator

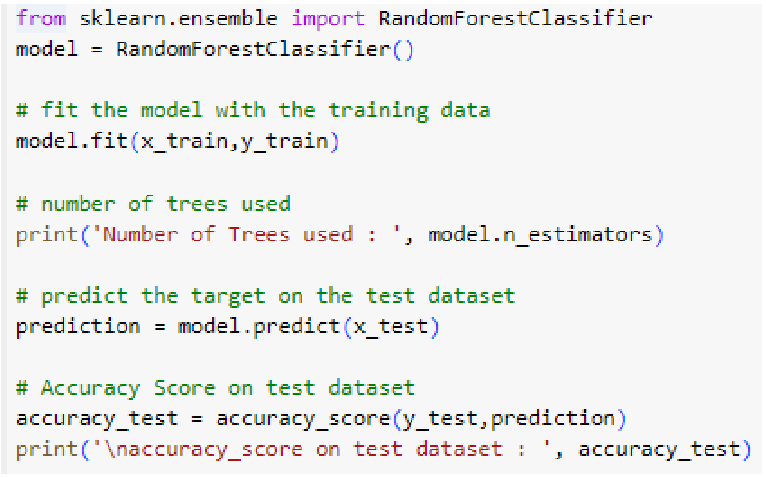
Random forest.

BAGGING META ESTIMATOR



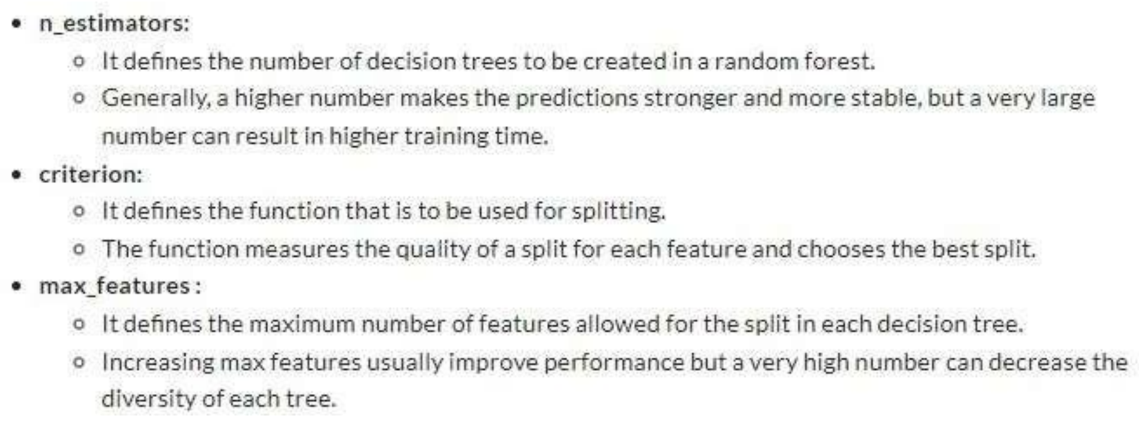
```
#sample code or classification problem
from sklearn.ensemble import BaggingClassifier
from sklearn import tree
model = BaggingClassifier(DecisionTreeClassifier(random_state=1))
model.fit(x_train, y_train)
model.score(x_test,y_test)
```

RANDOM FOREST



```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()

# fit the model with the training data
model.fit(x_train,y_train)

# number of trees used
print('Number of Trees used : ', model.n_estimators)

# predict the target on the test dataset
prediction = model.predict(x_test)

# Accuracy Score on test dataset
accuracy_test = accuracy_score(y_test,prediction)
print('\naccuracy_score on test dataset : ', accuracy_test)
```

PARAMETERS



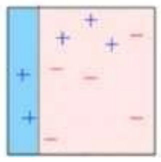
- n_estimators:
  - It defines the number of decision trees to be created in a random forest.
  - Generally, a higher number makes the predictions stronger and more stable, but a very large number can result in higher training time.
- criterion:
  - It defines the function that is to be used for splitting.
  - The function measures the quality of a split for each feature and chooses the best split.
- max_features :
  - It defines the maximum number of features allowed for the split in each decision tree.
  - Increasing max features usually improve performance but a very high number can decrease the diversity of each tree.

Learn more about Random Forest Parameters.

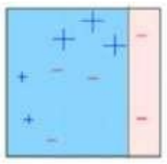
BOOSTING

Before we go further, here's another question for you; If a data point is incorrectly predicted by the first model, and then the next (probably all models), will combining the predictions provide better results? Such situations are taken care of by boosting
Boosting is a sequential process, where each subsequent model attempts to correct the errors of the previous model. The succeeding models are dependent on the previous model. Let's understand the way boosting works in the below steps.

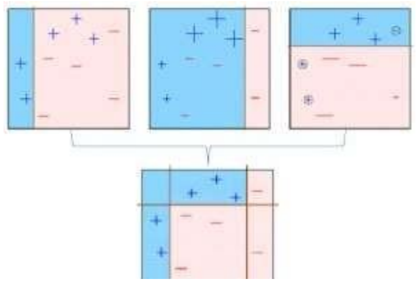
1. A subset is created from the original dataset
2. Initially, all data points are given equal weights.
3. Base model is created on this subset
4. This model is used to make predictions on the whole dataset.



5. Errors are calculated using the actual values and predicted values.
6. The observations which are incorrectly predicted, are given higher weights: (Here, the three misclassified blue-plus points will be given higher weights)
7. Another model is created and predictions are made on the dataset. (This model tries to correct the errors from the previous modell



8. Similarly, multiple models are created, each correcting the errors of the previous model.
9. The final model (strong learner) is the weighted mean of all the models (weak learners).



Thus, the boosting algorithm combines a number of weak learners to form a strong learner. The Individual models would not perform well on the entire dataset, but they work well for some part of the dataset. Thus, each model actually boosts the performance of the ensemble.

BOOSTING ALGORITHMS

- AdaBoost
- XGBoost

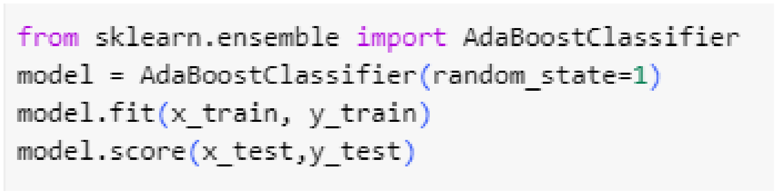
ADABOOST

Adaptive boosting or AdaBoost is one of the simplest boosting algorithms. Usually, decision trees are used for modelling. Multiple sequential models are created, each correcting the errors from the last model. AdaBoost assigns weights to the observations which are incorrectly predicted and the subsequent model works to predict these values correctly.

Below are the steps for performing the AdaBoost algorithm:

1. Initially, all observations in the dataset are given equal weights.
2. A model is built on a subset of data.
3. Using this model. predictions are made on the whole dataset.
4. Errors are calculated by comparing the predictions and actual values.
5. While creating the next model, higher weights are given to the data points which were predicted incorrectly.
6. Weights can be determined using the error value. For instance, higher the error more is the weight assigned to the observation.
7. This process is repeated until the error function does not change, or the maximum limit of the number of estimators is reached.

ADABOOST

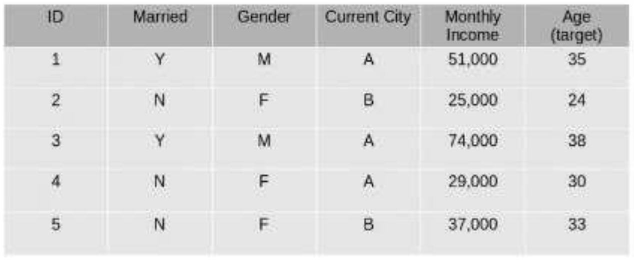


```python
from sklearn.ensemble import AdaBoostClassifier
model = AdaBoostClassifier(random_state=1)
model.fit(x_train, y_train)
model.score(x_test,y_test)
```

GRADIENT BOOSTING

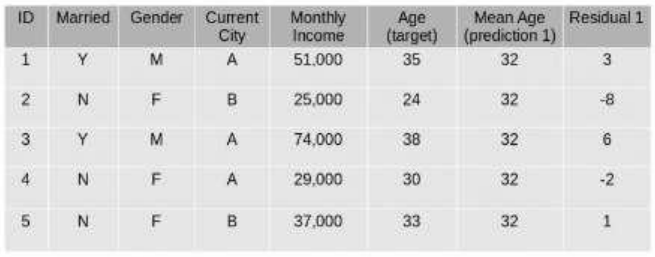
Gradient Boosting or GBM is another ensemble machine learning algorithm that works for both regression and classification problems, GBM uses the boosting technique, combining a number of weak learners to form a strong learner. Regression trees used as a base learner, each subsequent tree in series is built on the errors calculated by the previous tree.

We will use a simple example to understand the GBM algorithm. We have to predict the age of a group of people using the below data:



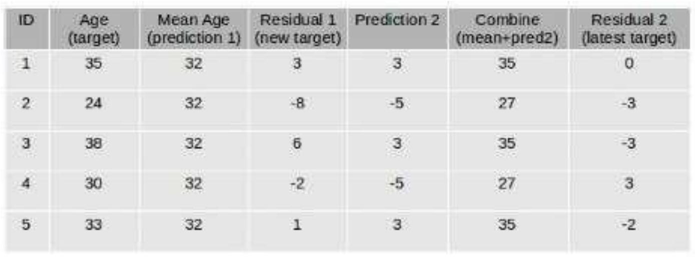
ID	Married	Gender	Current City	Monthly Income	Age (target)
1	Y	M	A	51,000	35
2	N	F	B	25,000	24
3	Y	M	A	74,000	38
4	N	F	A	29,000	30
5	N	F	B	37,000	33

1. The mean age is assumed to be the predicted value for all observations in the dataset.
2. The errors are calculated using this mean prediction and actual values of age



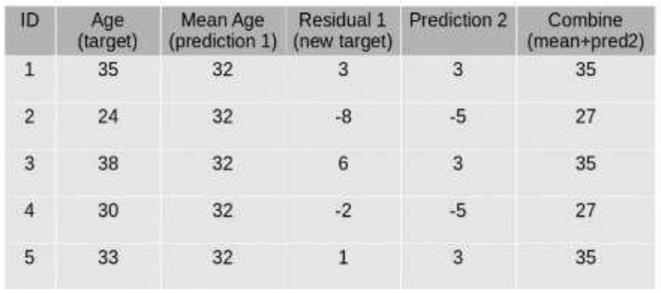
ID	Married	Gender	Current City	Monthly Income	Age (target)	Mean Age (prediction 1)	Residual 1
1	Y	M	A	51,000	35	32	3
2	N	F	B	25,000	24	32	-8
3	Y	M	A	74,000	38	32	6
4	N	F	A	29,000	30	32	-2
5	N	F	B	37,000	33	32	1

3. A tree model is created using the errors calculated above as target variable. Our objective is to find the best split to minimize the error
4. The predictions by this model are combined with the predictions 1



ID	Age (target)	Mean Age (prediction 1)	Residual 1 (new target)	Prediction 2	Combine (mean+pred2)	Residual 2 (latest target)
1	35	32	3	3	35	0
2	24	32	-8	-5	27	-3
3	38	32	6	3	35	-3
4	30	32	-2	-5	27	3
5	33	32	1	3	35	-2

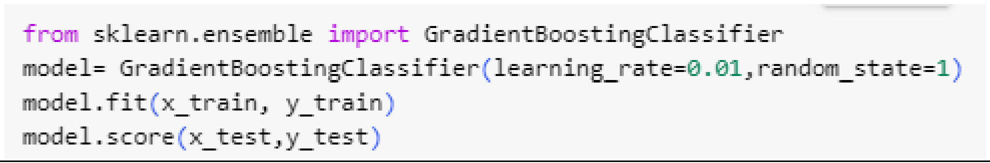
5. This value calculated above is the new prediction
6. New errors are calculated using this predicted value and actual value.



ID	Age (target)	Mean Age (prediction 1)	Residual 1 (new target)	Prediction 2	Combine (mean+pred2)
1	35	32	3	3	35
2	24	32	-8	-5	27
3	38	32	6	3	35
4	30	32	-2	-5	27
5	33	32	1	3	35

7. Steps 2 to 6 are repeated till the maximum number of iterations is reached for error function does not change

GRADIENT BOOSTING



```
from sklearn.ensemble import GradientBoostingClassifier
model= GradientBoostingClassifier(learning_rate=0.01,random_state=1)
model.fit(x_train, y_train)
model.score(x_test,y_test)
```

You can learn for [Ensemble Hyper parameters]() here

LAB 4 EXERCISE

TASK 1 :

Download the Dataset

- Perform EDA
- Check whether the dataset is balanced or not (using target variable “Label”)
- Check whether there is any empty records, categorical feature, or duplicate records, yes Then handle this and give a brief explanation why you have chosen this technique in a text cell or “jupyter/colab”
- Analyze your dataset and think if feature scaling is required or not. If yes then apply any scaling technique based on your distribution.
- Split your dataset in training, testing, and validation. The train split will be 80% and the test will be 20%. In the validation split your training samples will be 70% and the validation set will be 30%. Briefly describe why we use a validation set in a text cell. Declare Random_state=0
- Apply Random Forest, XGBoost, AdaBoost and check model training and testing accuracy.
- Compare the Training and Testing Results of all three algorithms
- Briefly discuss your results in a text cell to justify why you have achieved these result, which algorithm is more prone to overfitting or underfitting in your case and which algorithm is improving Bias Variance Tradeoff more.

TASK 2:

- Use the Same dataset as in Task 1
- Extract Only two Attributes with independent variable to analyze your results (restEcg and Oldpeak)
- Now train a Voting Classifier using (Decision Tree, KNN, Random Forest and XGboost)
- Check which Voting Parameter will give you the best Accuracy either soft or hard
- Check the best weights for these models.
- Plot the Bias and Variance Tradeoff Graph after Voting Classifier

TASK 3:

- Use the Same dataset as in Task 1
- Extract Only two Attributes with independent variable to analyze your results (restEcg and Chol)
- Now train a Voting Classifier using (Random Forest and Adaboost)
- Plot the training and testing accuracy of individual Random Forest and XGBoost + Accuracy graph of Voting Ensemble Technique as well.

NOTE:

Work on jupyter or Colab Only.
Make sure to complete all tasks in Same .ipyn File
Create heading for Each Task
Make sure to include comment for each Task you are performing.
Your Submission should include proper details regarding what you are doing, brief description of what you have achieved.
Plagiarism and not following the rules will result in marks deduction.