Preditive Modeling Using Six Machine Learning Techniques To Predict The Exit Polls. Dataset: Survey Data By A News Channel

Group Assignment Machine Learning

11-04-2021

Submitted by:Jayshree, Pragya,Priyanka,Swati

Table of Contents

1 Read the dataset. Do the descriptive statistics and do the null value condition check. Write an inference on it.

1.1 Understanding the case study:

CNBE, one of the leading news channels wants to analyze recent elections based on the dataset created by carrying a survey on 1525 voters. The information gather during the survey is divided into 9 features for each voter. The objective is to predict which party a voter will vote for based on the given information and thus create an exit poll that will help in predicting overall win and seats covered by a particular party.

We are assigned the task to build 6 models and select the best amongst them for the prediction. Following are the models that will be created, studied and compared to find the best model:

- Logistic Regression
- LDA
- KNN
- Naïve Bayes
- Bagging with RF
- Boosting

Following columns are given in the data set and the meaning of each column is interpreted as follows:

vote: This column is the target variable representing the party of choice. It is a binary variable with values as 'Conservative' and 'Labour'.

age: This is the numeric feature representing the age of the voter in years.

economic.cond.national: This variable is categorical/ordinal having 5 values ranging from 1 to 5 and it refers to the assessment of current national economic conditions.

***economic.cond.household*:** This variable is categorical/ordinal having 5 values ranging from 1 to 5 and it refers to the assessment of current household economic conditions.

Blair: This variable is categorical/ordinal having 5 values ranging from 1 to 5 and it refers to the assessment of the 'Labour' leader.

Hague: This variable is categorical/ordinal having 5 values ranging from 1 to 5 and it refers to the assessment of the 'Conservative' leader.

Europe: This is again a categorical/ordinal variable having range between 1 to 11 and it is the indicator of respondents' attitudes toward European integration. High scores represent 'Eurosceptic' sentiment.

political.knowledge: This variable is categorical/ordinal having 4 values ranging from 0 to 3 and it is an indicator of knowledge of parties' positions on European integration

gender: This is a binary variable with 2 values 'female' and 'male' representing the gender of the voter.

1.2 Descriptive Analysis:

1.2.1 Checking the data type for each column in the data.

```
#Checking the data types of each variable
df.dtypes
```

```
vote                        object
age                          int64
economic.cond.national       int64
economic.cond.household      int64
Blair                        int64
Hague                        int64
Europe                       int64
political.knowledge          int64
gender                      object
dtype: object
```

We have checked that there are **9 variables** with different data types like integer, and object. From the first look it seems that except for the 'vote' and 'gender', all other are integer variables but from the closer look in the data, it is observed that except 'age', all variables are categorical in nature, 'age' is continuous numeric column.

1.2.2 Checking the null values in the data

```
#Checking the null value in data
df.isnull().sum()
```

```
vote                        0
age                         0
economic.cond.national      0
economic.cond.household     0
Blair                       0
Hague                       0
Europe                      0
political.knowledge         0
gender                      0
dtype: int64
```

It is checked that there are **no null values** present in any of the columns. Hence no null value treatment is needed over this dataset.

1.2.3 Checking the duplicated values in the data

```
#Checking for the duplicate records in data
df.duplicated().sum()
```

```
0
```

Here, we observe that there are **no duplicated rows** in the data set when we use the original data with 'Unnamed' column. We have all the unique records in the data set.

```
#Checking for the duplicate records in data after dropping the 'Unnamed' column
df.duplicated().sum()
```

```
8
```

But if we drop the 'Unnamed' column, we get 8 duplicate records, but we will not drop them

assuming that the 'Unnamed' column is uniquely identifying each voter's record.

1.2.4 Checking the descriptive statistics of the data.

```
#Checking the descriptive statistics of the data.
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
age	1525.0	54.182295	15.711209	24.0	41.0	53.0	67.0	93.0
economic.cond.national	1525.0	3.245902	0.880969	1.0	3.0	3.0	4.0	5.0
economic.cond.household	1525.0	3.140328	0.929951	1.0	3.0	3.0	4.0	5.0
Blair	1525.0	3.334426	1.174824	1.0	2.0	4.0	4.0	5.0
Hague	1525.0	2.746885	1.230703	1.0	2.0	2.0	4.0	5.0
Europe	1525.0	6.728525	3.297538	1.0	4.0	6.0	10.0	11.0
political.knowledge	1525.0	1.542295	1.083315	0.0	0.0	2.0	2.0	3.0

The final dataset has 1525 records with no null or duplicate values. From the above result we can see that there are 7 numeric features but only 'age' is actually a continuous variable and rest all are categorical in nature.
We can observe that the voter's age lie between 24 to 93 years and this column seems to be quite normal as there is not much difference between mean and median.
For others, the five-point summary doesn't really makes sense as they are not continuous numeric features.
1.7 Checking the target column for the biasness.

```
# Comparing the numbers and percentage of 2 parties for vote

no_of_cases, per_of_cases = df.vote.value_counts(),df.vote.value_counts(normalize=True)*100

print( 'Parties in no:\n',no_of_cases,'\n\n Parties in %:\n', per_of_cases)
```

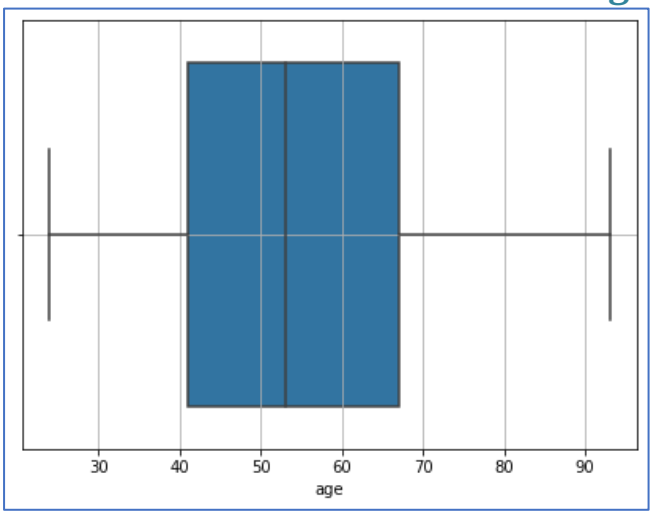
```
Parties in no:
 Labour          1063
Conservative     462
Name: vote, dtype: int64

 Parties in %:
 Labour          69.704918
Conservative    30.295082
Name: vote, dtype: float64
```

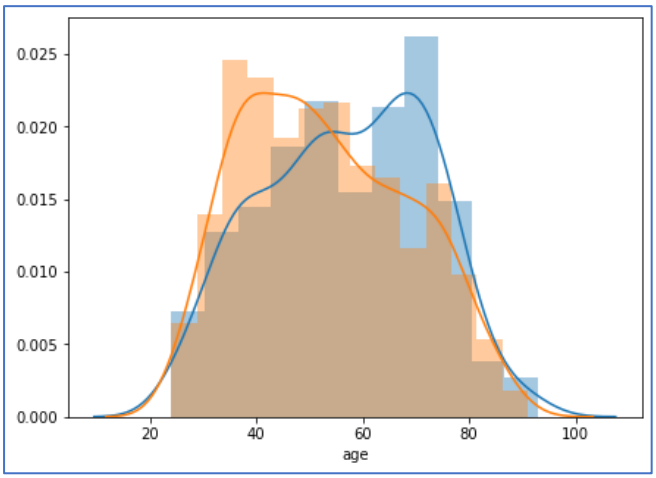
We can see that the values in target column is approximately 70-30 in proportion which means that the data is not biased, and we do not need to apply class balancing techniques here.

2 Perform Univariate and Bivariate Analysis. Do exploratory data analysis. Check for Outliers.

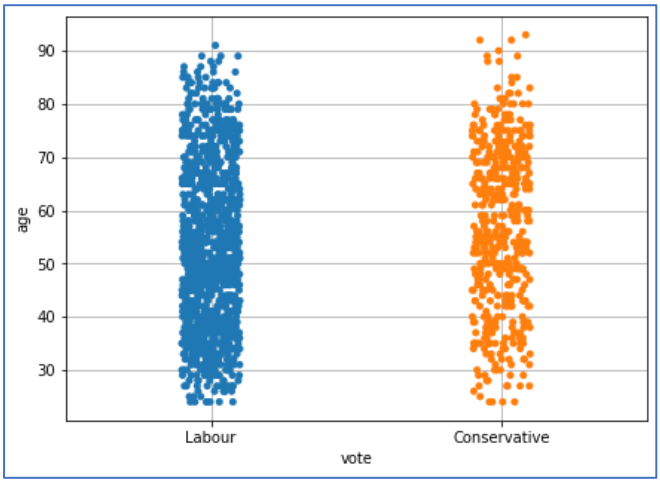
2.1 Checking the outliers and overall distribution of numerical columns and relation with the target column.



We checked for the outliers in the column age, the only numeric continuous feature present in the data using the above boxplot and from this we can see that there are no outliers present for 'age'. Since all other columns are categorical in nature and age has not got any outliers, we need not do any outlier treatment for this data.

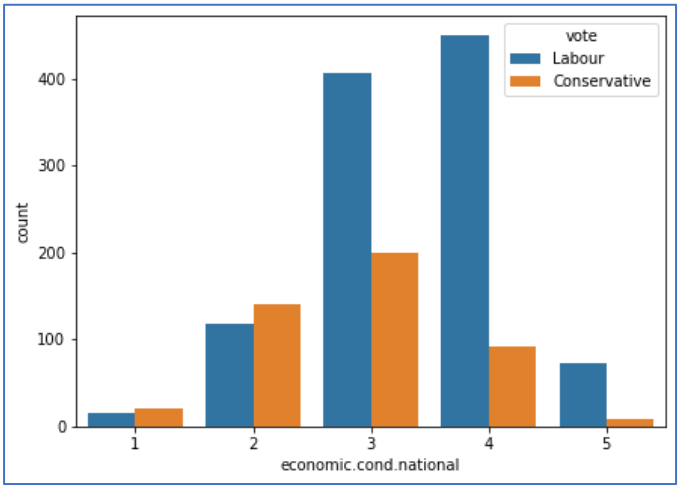


Next, we checked the distribution of 'age' over 2 classes of vote separately, the overall distribution is found to be not skewed, moderately normal but bi-modal as we see 2 peaks around the age 40 and 80 years. Interestingly, for 'Conservative' class the maximum no. of voters is young, and they lie between the age of around 40 years. For 'Labour' class most voters are of the older category lying between the age of 60-80 years.

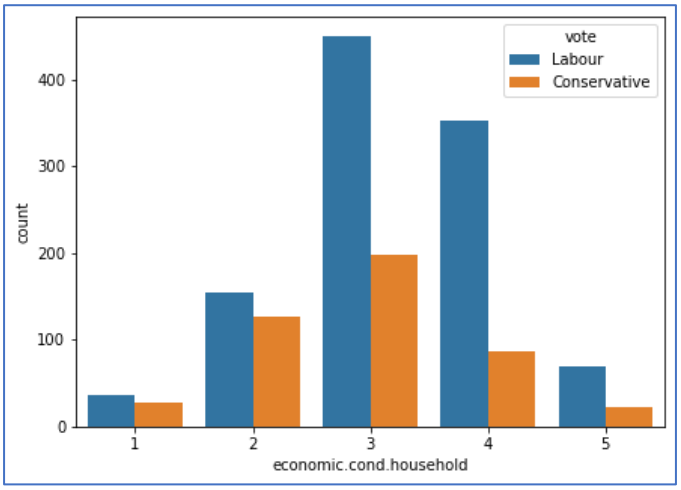


Next, we wanted to check the distribution of 'age' on the 2 classes individually by using the above strip plot. From this, we can see that both the classes have voters from all age groups but class 'Labour' has overall more no. of voters than 'Conservative' class as we have found while checking the biasness in the target column.

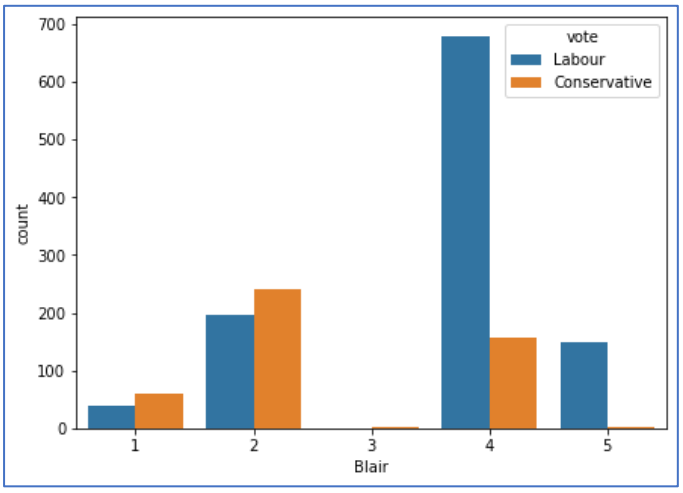
2.2 Checking the distribution of categorical columns w.r.t. target column.



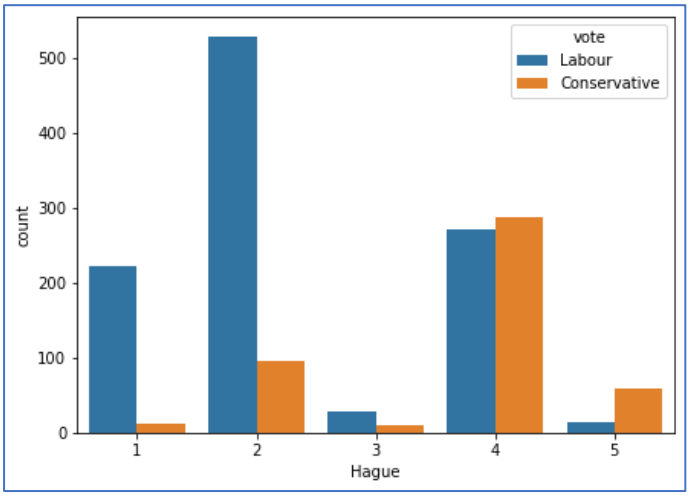
We checked the distribution of each categorical variable with over the two classes for 'vote'. In 'economic.cond.national' we found the voters with current national economic condition 3 and 4 are the most in numbers. Also, voters in the lower economic conditions with value 1 and 2 are more likely to vote to the 'Conservative' party, whereas people in middle to upper economic conditions are voting to the 'Labour' party far more than the 'Conservative' party.



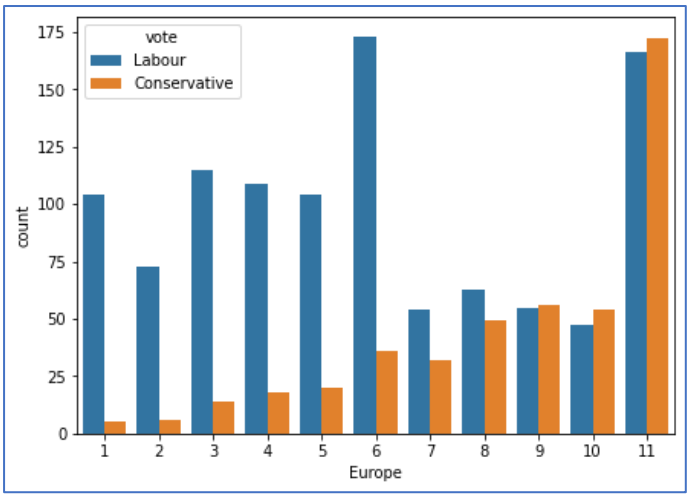
For ‘economic.cond.household’, in all the categories, i.e., 1 to 5, voters are more for the ‘Labour’ class as compared to the ‘Conservative’ class. Again here, most of the voters fall under the economic condition 3 and 4 in numbers.



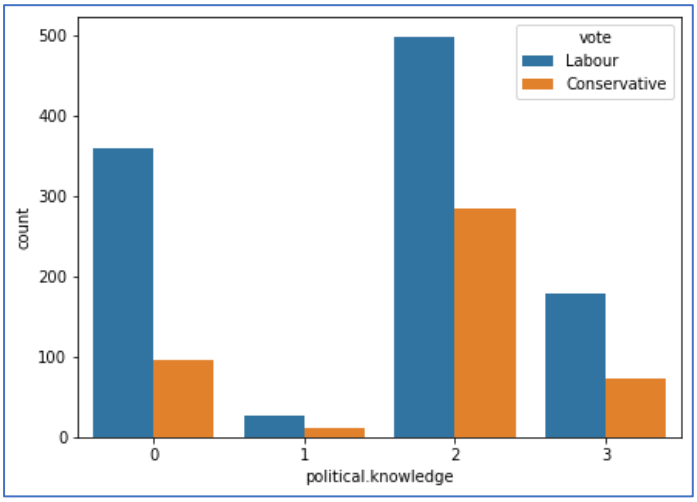
Next, coming to the distribution of ‘Blair’ which is an indicator of the assessment of ‘Labour’ leader, we find the maximum voters in 4 and extremely few voters in 3. For 1 and 2, we see that ‘Labours’ voters are less than the ‘Conservatives’, moving to the higher side, voters for ‘Conservatives’ are more than ‘Labours’.



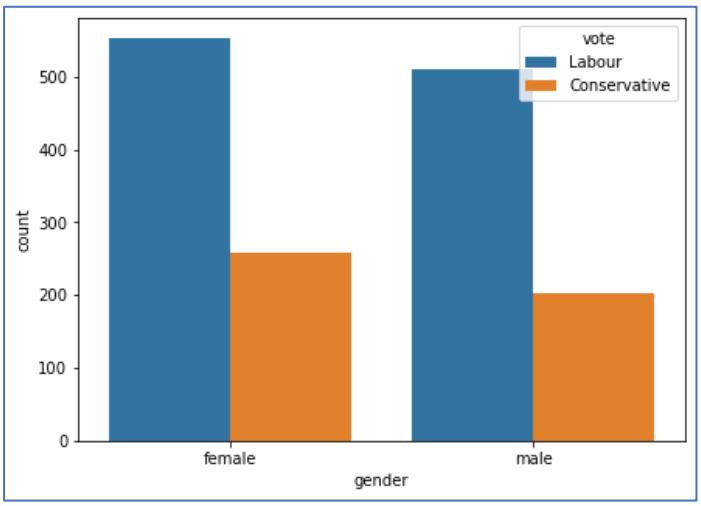
In the distribution of 'Blair' which is an indicator of the assessment of 'Conservative' leader, we find the maximum voters in 2 and very few voters in 3 and 5. For all the categories, 'Labour' has more voters than 'Conservative' except 5.



For the distribution of 'Europe' which is a measure of 'Eurosceptic' sentiments, there is a high number of voters in the level 11, that means high number supporter of European integrations with strongest response. In the lower levels till 8, 'Labour' supporter are higher than the 'Conservative' supporters and as the level increases, i.e., for 9, 10 and 11, 'Conservative' voters are more in number.

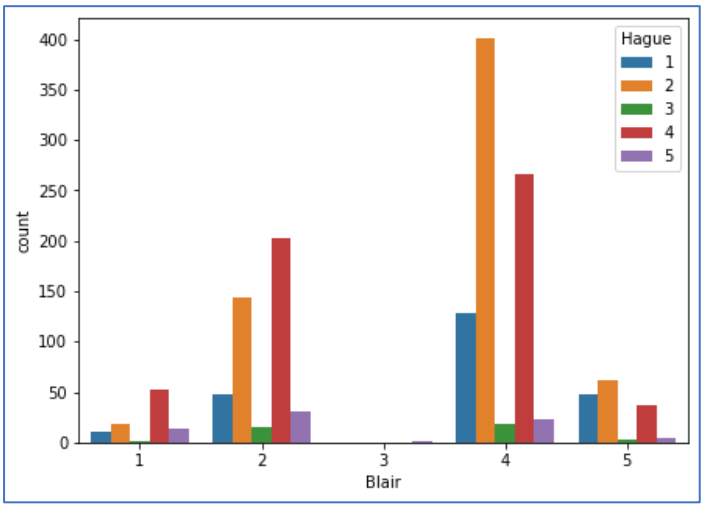


For the ‘political.knowledge’, which signifies the knowledge if parties’ position in European integration, highest number of voters are at the level 2 and very few are in level 1. All levels have more ‘Labour’ supporters than the ‘Conservative’ ones.

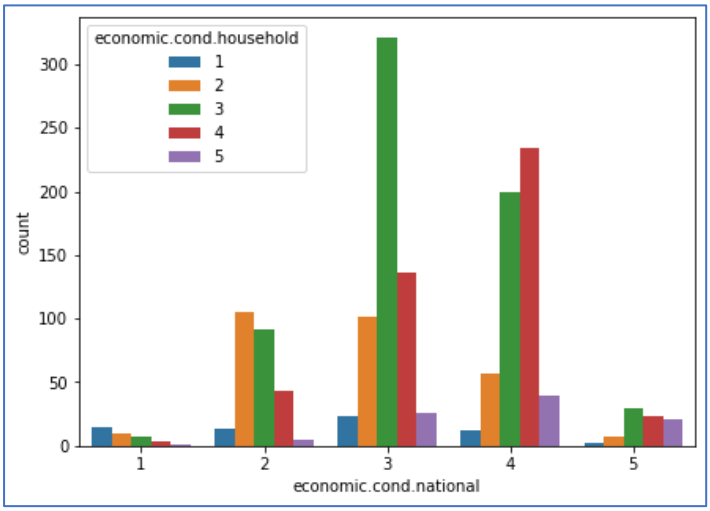


‘gender’ distribution doesn’t much of the varied results when it comes to the number of voters. Both male and female voters are greater supported of ‘Labour’ class than the ‘Conservative’ ones. ‘male’ voters are slightly less in number than the ‘female’ voters.

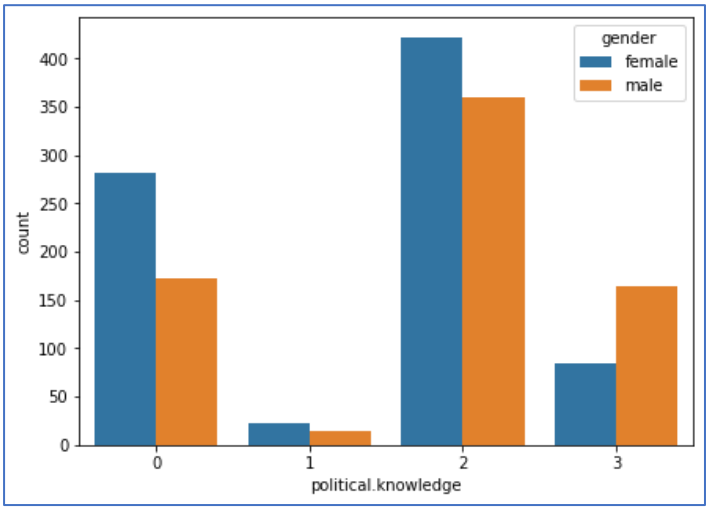
2.3 Checking multivariate analysis to dig deep on our earlier findings



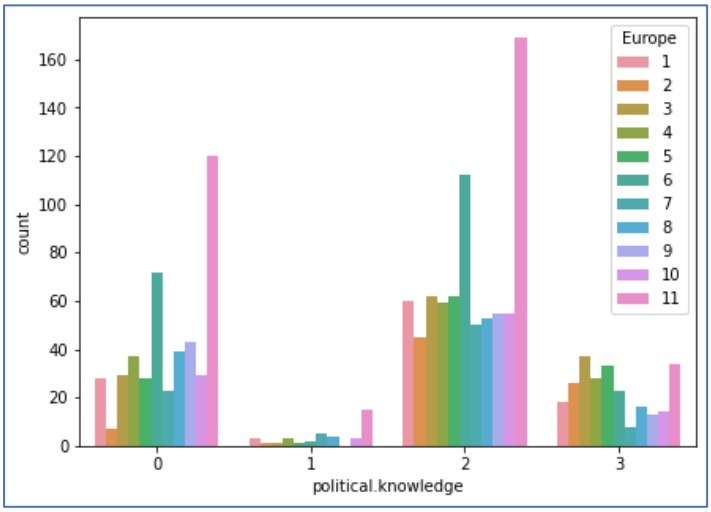
'Blair' and 'Hague' being the assessment of respective 'vote' class leaders have most voters in level 2 and 4 overall, where Hague is higher at 2 and Blair is higher at 4.



For both kinds of 'economic conditions', 3 has the highest voters and 4 also has fairly high number followed by 2. It is observed that the voters which are 1,2,3 and 4 level of 'economic.cond.household' are at the same level respectively for 'economic.cond.national'.



For the ‘political.knowledge’ as we saw that most of the voters stand at level 2, it is interesting to see that for level 0,1 and 2 ‘female’ voters are more that the ‘male’ voters.



As checked in the previous graph, most voters are with ‘political.knowledge’ level 4, we can see that in most of the levels, voters are most with the Eurosceptic sentiment level 11 and fairly high with Eurosceptic sentiment level 6.

3 Encode the data (having string values) for Modelling. Is Scaling necessary here or not? Data Split: Split the data into train and test (70:30).

In order to build the models, we need to divide the data into train data set and test data set, so that we can measure the performance metrics of both the data sets and avoid problems like overfitting, etc.

Before splitting the data into train and test we need to perform couple of data pre-processing steps:

Separating out the target and independent variables into X and y.

```
X= df.drop('vote',axis=1)
y= df['vote']
```

3.1 Data Encoding

Converting all the columns with object data types to numeric ones because these models cannot process string values in python.

```
## Converting the object Variables into numeric by using the LabelEncoder functionality inside sklearn.
from sklearn.preprocessing import LabelEncoder

## Defining a Label Encoder object instance
LE = LabelEncoder()
df['vote'] = LE.fit_transform(df['vote'])
X['gender'] = LE.fit_transform(X['gender'])
```

3.2 Data Split

Now, the data set is ready to be split into train and test data sets. We have followed the splitting criteria of 70% data in train and 30% data in test with the random_state set to 10 to get same output on each run and stratifying on the target column

```
# Split X and y into training and test set in 70:30 ratio
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30 , random_state=10,stratify=df['vote'])
```

3.3 Scaling the data

In the process of modelling, we will be using models which assigns weights to each independent feature (e.g. Logistic regression and LDA).

We will be also creating models which are distance based (e.g. KNN) which uses Euclidean distance as the basis of the distance calculation between 2 data points.

Euclidean distance between data two points p and q is calculated as below:

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

So, it becomes important for all the variables to be on the same scale and be given equal importance while contributing to the distance calculation.
Scaling can also manage the outliers to some extent if present in the data.
Moreover, we can see in *section 1.2.4* that the data is spread over multiple scales for each column, in this case scaling becomes necessary.

```
#Using z-score transformation to scale the X_train and X_test
from scipy.stats import zscore
X_train=X_train.apply(zscore)
X_test=X_test.apply(zscore)
print(start+"Head of scaled X_train data"+end)
print(display(X_train.head()))
print(start+"Head of scaled X_test data"+end)
print(display(X_test.head()))
```

4 Modelling

4.1 Apply Logistic Regression and LDA (linear discriminant analysis). (5 marks)

4.1.1 Logistic Regression

4.1.1.1 *Iteration 1: with all independent variables*

Logit Regression Results

Dep. Variable:	vote	**No. Observations:**	1067
Model:	Logit	**Df Residuals:**	1058
Method:	MLE	**Df Model:**	8
Date:	Sun, 11 Apr 2021	**Pseudo R-squ.:**	0.3546
Time:	12:14:05	**Log-Likelihood:**	-422.22
converged:	True	**LL-Null:**	-654.23
Covariance Type:	nonrobust	**LLR p-value:**	3.657e-95

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	3.3806	0.622	5.436	0.000	2.162	4.600
age	-0.0188	0.006	-3.351	0.001	-0.030	-0.008
economic_cond_national	0.3530	0.108	3.256	0.001	0.141	0.566
economic_cond_household	0.0851	0.101	0.846	0.398	-0.112	0.282
Blair	0.5354	0.077	6.969	0.000	0.385	0.686
Hague	-0.8544	0.079	-10.846	0.000	-1.009	-0.700
Europe	-0.2068	0.029	-7.099	0.000	-0.264	-0.150
political_knowledge	-0.3752	0.084	-4.476	0.000	-0.539	-0.211
gender	0.2005	0.178	1.129	0.259	-0.148	0.549

We may observe that gender and economic_cond_household gives us p-value greater than 0.05. Hence, we will drop gender and see check the results again.

4.1.1.2 *Iteration 2:Dropping Gender*

Logit Regression Results

Dep. Variable:	vote	**No. Observations:**	1067
Model:	Logit	**Df Residuals:**	1059
Method:	MLE	**Df Model:**	7
Date:	Sun, 11 Apr 2021	**Pseudo R-squ.:**	0.3537
Time:	12:14:07	**Log-Likelihood:**	-422.86
converged:	True	**LL-Null:**	-654.23
Covariance Type:	nonrobust	**LLR p-value:**	8.132e-96

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	3.4108	0.621	5.490	0.000	2.193	4.628
age	-0.0188	0.006	-3.352	0.001	-0.030	-0.008
economic_cond_national	0.3565	0.108	3.289	0.001	0.144	0.569
economic_cond_household	0.0853	0.101	0.847	0.397	-0.112	0.283
Blair	0.5413	0.077	7.057	0.000	0.391	0.692
Hague	-0.8519	0.079	-10.833	0.000	-1.006	-0.698
Europe	-0.2072	0.029	-7.123	0.000	-0.264	-0.150
political_knowledge	-0.3615	0.083	-4.364	0.000	-0.524	-0.199

We may observe that economic_cond_household still has p-value greater than 0.05. Hence, we will drop this feature too and check the results again.

4.1.1.3 *Iteration 3:Dropping economic_cond_household*

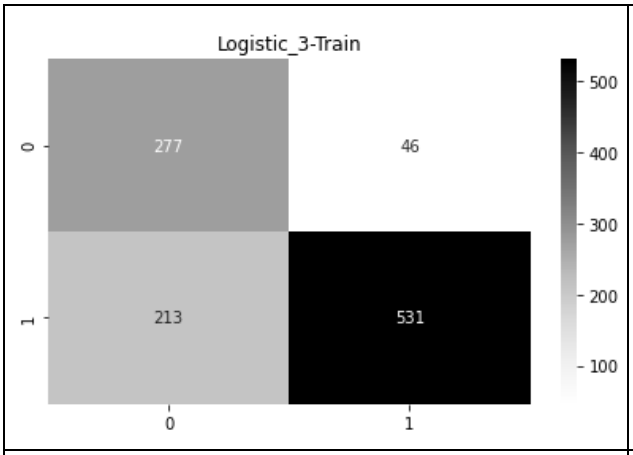
Logit Regression Results

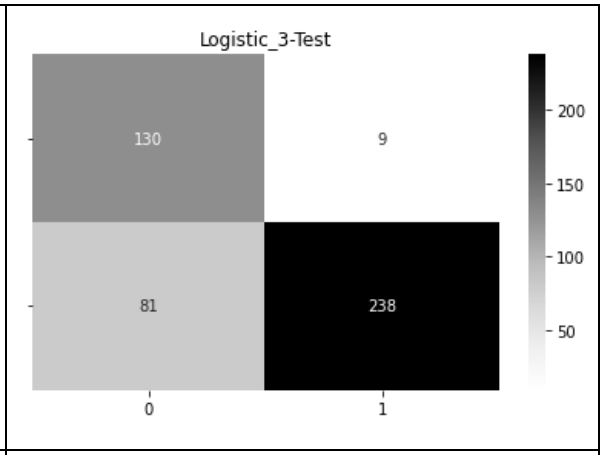
Dep. Variable:	vote	**No. Observations:**	1067
Model:	Logit	**Df Residuals:**	1060
Method:	MLE	**Df Model:**	6
Date:	Sun, 11 Apr 2021	**Pseudo R-squ.:**	0.3531
Time:	12:41:21	**Log-Likelihood:**	-423.22
converged:	True	**LL-Null:**	-654.23
Covariance Type:	nonrobust	**LLR p-value:**	1.264e-96

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	3.5762	0.591	6.050	0.000	2.418	4.735
age	-0.0193	0.006	-3.459	0.001	-0.030	-0.008
economic_cond_national	0.3873	0.102	3.792	0.000	0.187	0.587
Blair	0.5497	0.076	7.232	0.000	0.401	0.699
Hague	-0.8516	0.079	-10.836	0.000	-1.006	-0.698
Europe	-0.2076	0.029	-7.142	0.000	-0.265	-0.151
political_knowledge	-0.3604	0.083	-4.352	0.000	-0.523	-0.198

Now all the p-values are less than .05 confidence intervals. Also, from VIF we have observed that there is no multicollinearity between the independent variables.

4.1.1.4 *Performance Metrics: Check the performance of Predictions on Train and Test sets using Accuracy, Confusion Matrix, Plot ROC curve and get ROC_AUC score.*



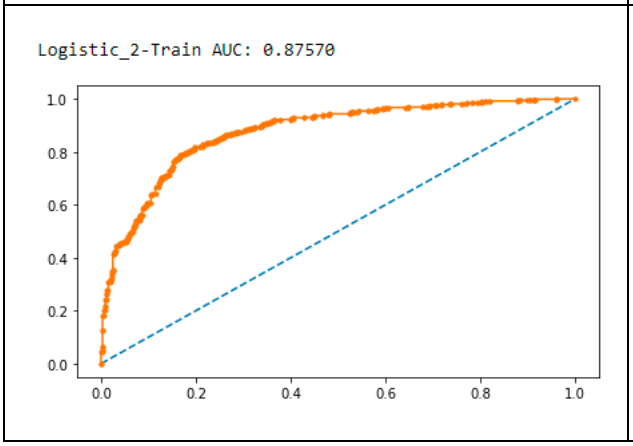


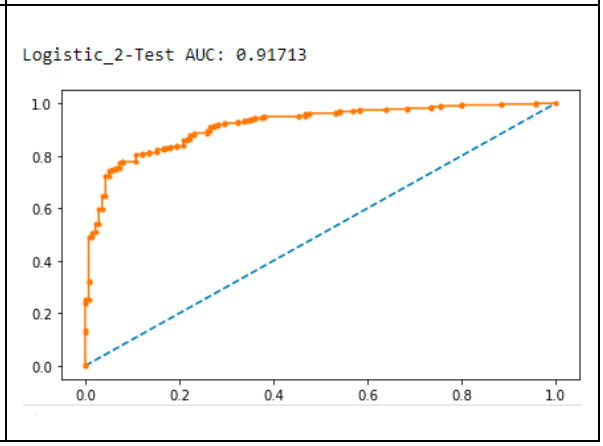
Logistic_3-Train

	precision	recall	f1-score	support
0	0.57	0.86	0.68	323
1	0.92	0.71	0.80	744
accuracy			0.76	1067
macro avg	0.74	0.79	0.74	1067
weighted avg	0.81	0.76	0.77	1067

Logistic_3-Test

	precision	recall	f1-score	support
0	0.62	0.94	0.74	139
1	0.96	0.75	0.84	319
accuracy			0.80	458
macro avg	0.79	0.84	0.79	458
weighted avg	0.86	0.80	0.81	458





4.1.1.5 *Observations:*

Model	Accuracy Train	Accurac Test	ROC-AUC Train	ROC-AUC Test
Iteration 1	0.76	0.81	.88	.92
Iteration 2	0.76	0.81	.88	.92
Iteration 3	0.76	0.80	.86	.92

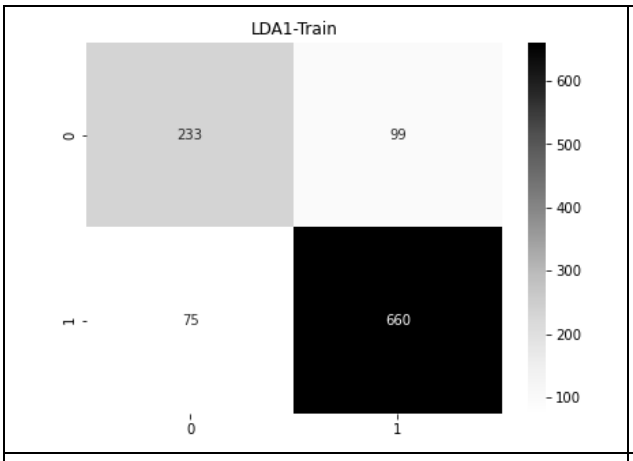
- We ran three iterations of LR as the p-value of gender and economic_cond_household were both greater than .05. We dropped both one by one and also alternatively but the two features seems to have no affect on each other and the results remained same which ever is dropped first.

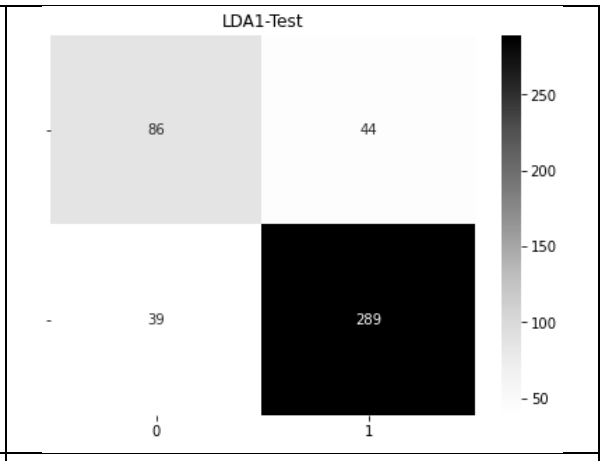
- Next, we observe the model accuracy in all three iterations is 76% only which is not a very good score for exit polls prediction. Secondly, model is performing better for test data than train data, which ideally should not happen. As the model is trained on the train data, it should be more effective on it. The reason in such cases is hard to find but one could be the way split happened, such that test data has randomly fallen into a good position for model to explain. (This may be overcome by changing the split criterion.)
- Next we may notice the variance between train and test is very high and has not reduced even after dropping the features.
- Above observations are also reflected in ROC-AUC score which is very high for test .92 and good but low for train at .83.
- From all the above, we may not recommend LR model as a good enough model for this case study.

4.1.2 Linear Discriminant Analysis (LDA)

Next we use LDA as a classification technique. LDA LDA works by assigning class to a new data-point depending upon the matching of class-specific probability densities.
We use LDA with train test split of 70:30 and check the performance of the model.

4.1.2.1 *Performance Metrics: Check the performance of Predictions on Train and Test sets using Accuracy, Confusion Matrix, Plot ROC curve and get ROC_AUC score.*



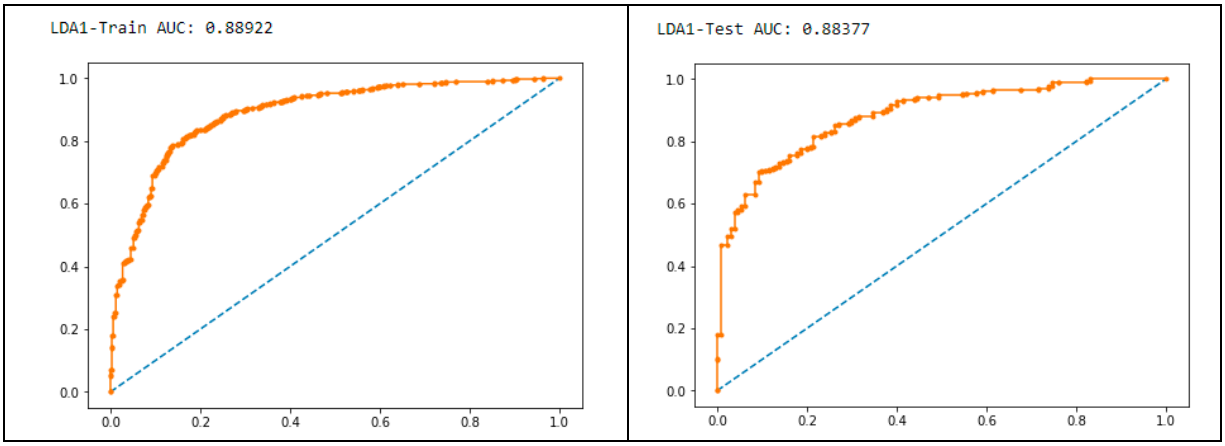


LDA1-Train

	precision	recall	f1-score	support
0	0.76	0.70	0.73	332
1	0.87	0.90	0.88	735
accuracy			0.84	1067
macro avg	0.81	0.80	0.81	1067
weighted avg	0.83	0.84	0.84	1067

LDA1-Test

	precision	recall	f1-score	support
0	0.69	0.66	0.67	130
1	0.87	0.88	0.87	328
accuracy			0.82	458
macro avg	0.78	0.77	0.77	458
weighted avg	0.82	0.82	0.82	458



4.1.2.2 *Observations:*

Model	Accuracy Train	Accurac Test	ROC-AUC Train	ROC-AUC Test
LDA	0.84	0.82	0.89	0.88

Second model that we have is LDA, with train accuracy at 84% and test accuracy at 82%. Although the ROC-AUC values is good at .89 and .88 but the variation in this model is high.

4.2 Apply KNN Model and Naïve Bayes Model. Interpret the results.

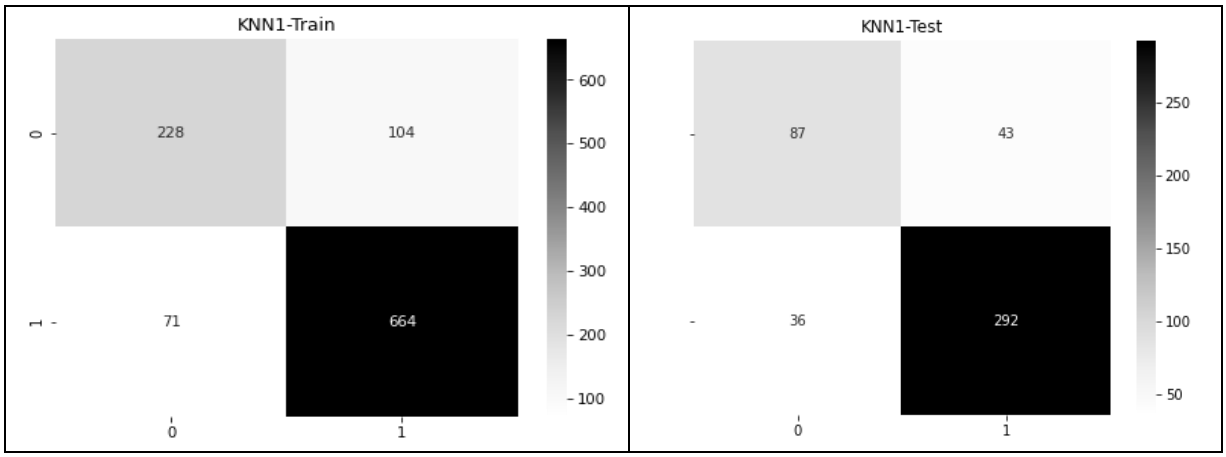
The next two model that we apply are tree-based models.

4.2.1 K-Nearest Neighbors

K-Nearest Neighbour (Knn) is one of the simplest Machine Learning algorithms. It is a Supervised Learning technique which classifies the new records based on similarity to the k nearest similar records in the train dataset. It is a non-parametric model.

Below we may check the output for knn model with and without hyper tuning.

4.2.1.1 *Performance Metrics without hyper tuning: Check the performance of Predictions on Train and Test sets using Accuracy, Confusion Matrix, Plot ROC curve and get ROC_AUC score*

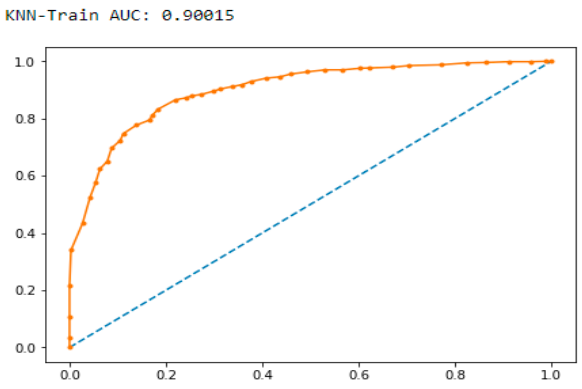


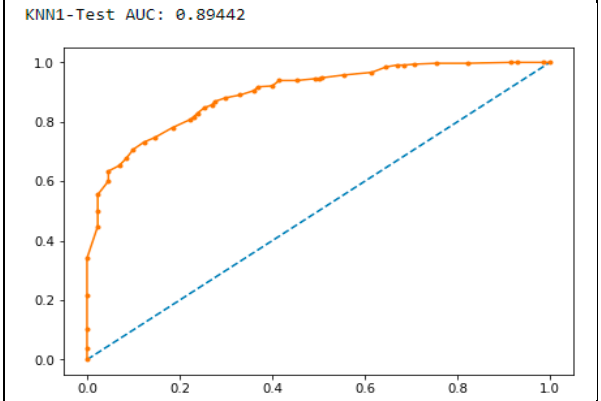
KNN1-Train

	precision	recall	f1-score	support
0	0.76	0.69	0.72	332
1	0.86	0.90	0.88	735
accuracy			0.84	1067
macro avg	0.81	0.80	0.80	1067
weighted avg	0.83	0.84	0.83	1067

KNN1-Test

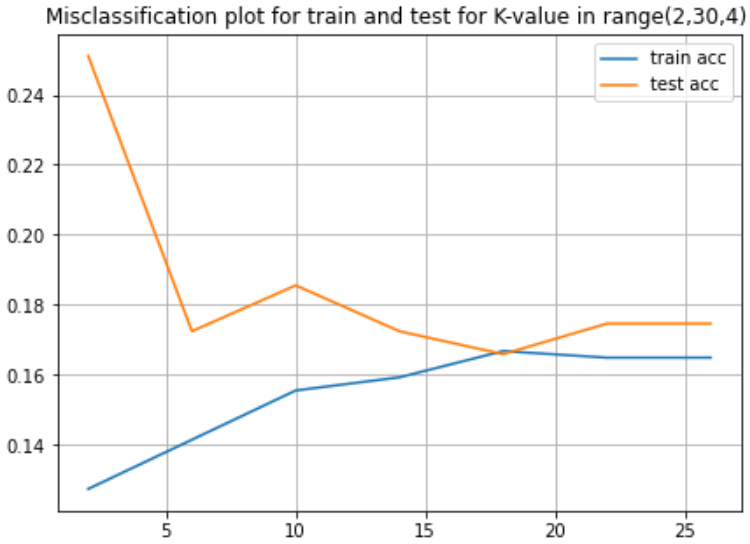
	precision	recall	f1-score	support
0	0.71	0.67	0.69	130
1	0.87	0.89	0.88	328
accuracy			0.83	458
macro avg	0.79	0.78	0.78	458
weighted avg	0.82	0.83	0.83	458



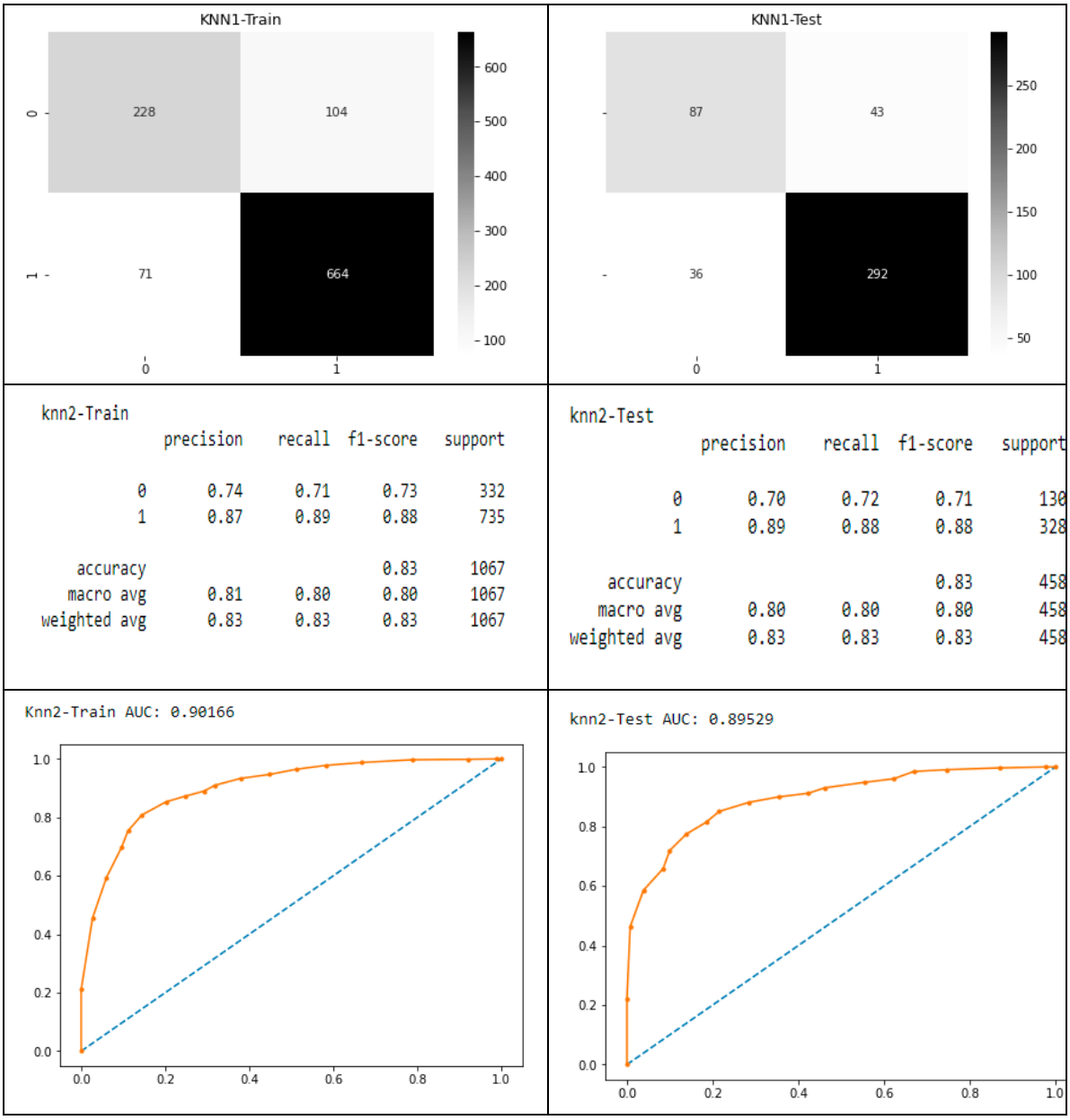


4.2.1.2 *Deciding the value of k*

Let us see how the classification accuracy for train and test changes with change in value of k. As we may see that variance is minimum at k value = 18. We can run the model again with the optimum k value and see if performance improves.



4.2.1.3 *Performance Metrics with 'k' value 18: Check the performance of Predictions on Train and Test sets using Accuracy, Confusion Matrix, Plot ROC curve and get ROC_AUC score*

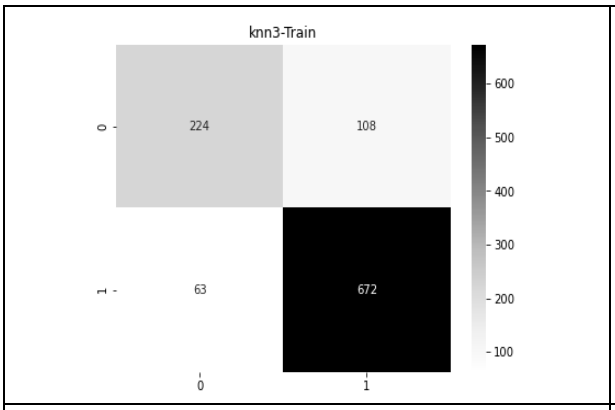


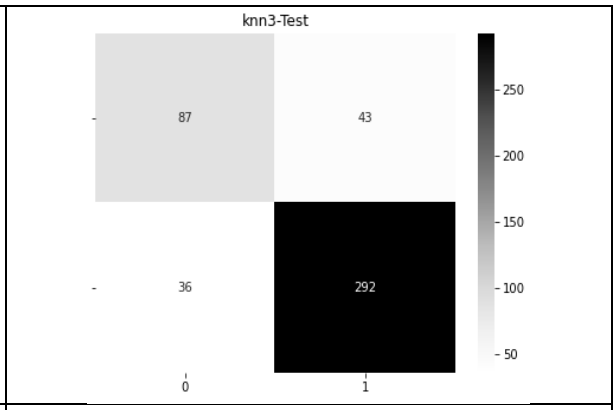
4.2.1.4 *Performance Metrics with hyper tuning using Grid Search CV: Check the performance of Predictions on Train and Test sets using Accuracy, Confusion Matrix, Plot ROC curve and get ROC_AUC score*

```
from sklearn.model_selection import GridSearchCV

params = {'algorithm':['auto', 'ball_tree','kd_tree','brute'],
          'leaf_size' : list(range(3,30,3)),
          'p' : [1,2,3,4,5],
          'n_neighbors' : [17,19]
          }

knn3 = GridSearchCV(KNeighborsClassifier() , param_grid = params , refit = True , cv = 3 )
```





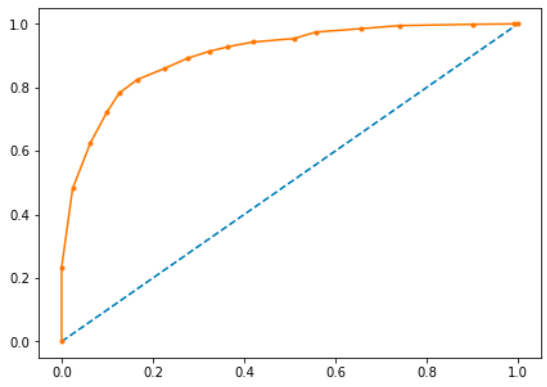
Knn3-Train

	precision	recall	f1-score	support
0	0.78	0.67	0.72	332
1	0.86	0.91	0.89	735
accuracy			0.84	1067
macro avg	0.82	0.79	0.81	1067
weighted avg	0.84	0.84	0.84	1067

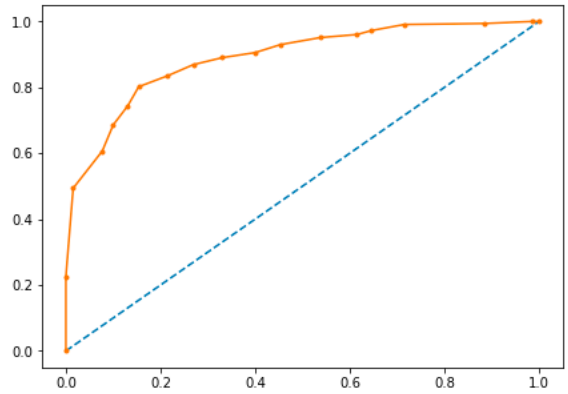
Knn3-Test

	precision	recall	f1-score	support
0	0.71	0.67	0.69	130
1	0.87	0.89	0.88	328
accuracy			0.83	458
macro avg	0.79	0.78	0.78	458
weighted avg	0.82	0.83	0.83	458

Knn3-Train AUC: 0.90385



Knn3-Test AUC: 0.88994



4.2.1.5 *Observations:*

Model	Accuracy Train	Accurac Test	ROC-AUC Train	ROC-AUC Test
KNN with default 'k'	0.84	0.83	0.90	0.89
KNN with optimum k=18	0.83	0.83	0.90	0.90
Knn with Grid Search	0.84	0.83	0.90	0.89

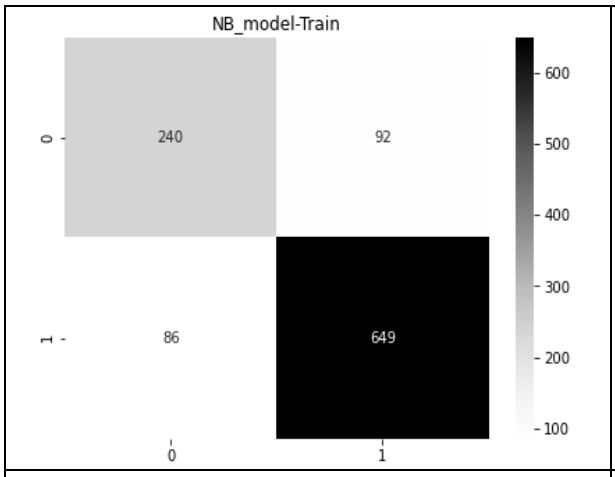
Third model that we have considered is tree-based model KNN. We trained the KNN with three scenarios, where we derived the optimum K neighbors value as 18 and then applied grid

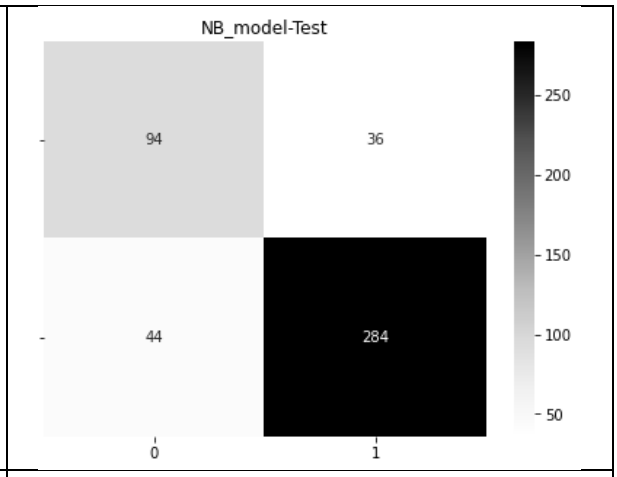
search parameters. We can observe that the performance of model was better after using k = 18 but has decreased after further grid search parameters. After using k=18, model has decreased the variation between train and test (from 84% & 83% to 83% & 83%) and increased ROC-AUC score from .89 to .90. Hence, we may consider the KNN with default parameters and k=18 as the best of the three.

4.2.2 Naïve Bayes

Naive Bayes is a technique which assigns class labels to problem instances, represented as vectors for each feature value, where the class labels are drawn from some finite set. Naïve bayes works on several algorithms at a time based on a common assumption that the value of a particular feature is independent of the value of any other features.

4.2.2.1 *Performance Metrics without hyper tuning: Check the performance of Predictions on Train and Test sets using Accuracy, Confusion Matrix, Plot ROC curve and get ROC_AUC score*



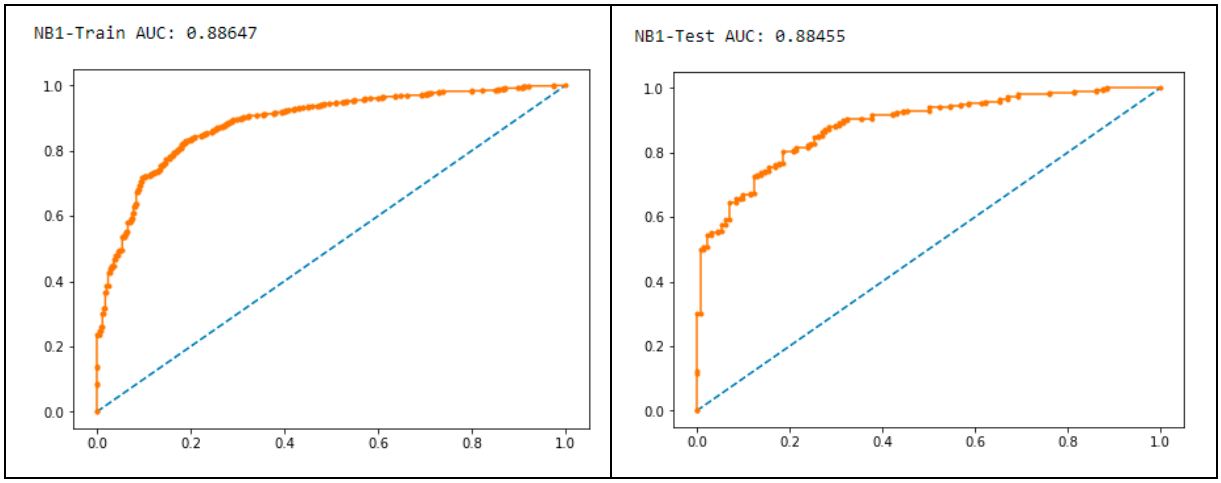


NB_model-Train

	precision	recall	f1-score	support
0	0.74	0.72	0.73	332
1	0.88	0.88	0.88	735
accuracy			0.83	1067
macro avg	0.81	0.80	0.80	1067
weighted avg	0.83	0.83	0.83	1067

NB_model-Test

	precision	recall	f1-score	support
0	0.68	0.72	0.70	130
1	0.89	0.87	0.88	328
accuracy			0.83	458
macro avg	0.78	0.79	0.79	458
weighted avg	0.83	0.83	0.83	458



4.2.2.2 *Performance Metrics with hyper tuning using Cross Validation: Check the performance of Predictions on Train and Test sets using Accuracy, Confusion Matrix, Plot ROC curve and get ROC_AUC score*

```
train_acc = cross_val_score(NB_model,X_train,y_train,cv=2,scoring='accuracy')
test_acc = cross_val_score(NB_model,X_test,y_test,cv=2,scoring='accuracy')
print('Train-Test Accuracy Score after two fold Cross Validation')
print('\nTrain-Acc',train_acc,'\nTest-Acc ',test_acc)
```

```
Train-Test Accuracy Score after two fold Cross Validation

Train-Acc [0.82209738 0.84803002]
Test-Acc  [0.81222707 0.84716157]
```

```
train_roc_auc = cross_val_score(NB_model,X_train,y_train,cv=2,scoring='roc_auc')
test_roc_auc = cross_val_score(NB_model,X_test,y_test,cv=2,scoring='roc_auc')
print('Train-Test ROC-AUC after two fold Cross Validation')
print('\nTrain-roc_auc',train_roc_auc,'\nTest-roc_auc ',test_roc_auc)
```

```
Train-Test ROC-AUC after two fold Cross Validation

Train-roc_auc [0.82209738 0.84803002]
Test-roc_auc  [0.81222707 0.84716157]
```

```
train_acc = cross_val_score(NB_model,X_train,y_train,cv=3,scoring='accuracy')
test_acc = cross_val_score(NB_model,X_test,y_test,cv=3,scoring='accuracy')
print('Train-Test Accuracy Score after three fold Cross Validation')
print('\nTrain-Acc',train_acc,'\nTest-Acc ',test_acc)
```

```
Train-Test Accuracy Score after three fold Cross Validation

Train-Acc [0.83146067 0.83146067 0.85633803]
Test-Acc  [0.83006536 0.79084967 0.86842105]
```

```
train_roc_auc = cross_val_score(NB_model,X_train,y_train,cv=3,scoring='roc_auc')
test_roc_auc = cross_val_score(NB_model,X_test,y_test,cv=3,scoring='roc_auc')
print('Train-Test ROC-AUC after three fold Cross Validation')
print('\nTrain-roc_auc',train_roc_auc,'\nTest-roc_auc ',test_roc_auc)
```

```
Train-Test ROC-AUC after three fold Cross Validation

Train-roc_auc [0.86795367 0.90586505 0.87996289]
Test-roc_auc  [0.882402   0.86596195 0.90868359]
```

```
train_acc = cross_val_score(NB_model,X_train,y_train,cv=4,scoring='accuracy')
test_acc = cross_val_score(NB_model,X_test,y_test,cv=4,scoring='accuracy')
print('Train-Test Accuracy Score after four fold Cross Validation')
print('\nTrain-Acc',train_acc,'\nTest-Acc ',test_acc)
```

```
Train-Test Accuracy Score after four fold Cross Validation

Train-Acc [0.82771536 0.82397004 0.83520599 0.84210526]
Test-Acc  [0.84347826 0.79130435 0.85964912 0.84210526]
```

```
train_roc_auc = cross_val_score(NB_model,X_train,y_train,cv=4,scoring='roc_auc')
test_roc_auc = cross_val_score(NB_model,X_test,y_test,cv=4,scoring='roc_auc')
print('Train-Test ROC-AUC after four fold Cross Validation')
print('\nTrain-roc_auc',train_roc_auc,'\nTest-roc_auc ',test_roc_auc)
```

```
Train-Test ROC-AUC after four fold Cross Validation

Train-roc_auc [0.8731011  0.88881613 0.89929282 0.87767463]
Test-roc_auc  [0.87398374 0.87287509 0.90777439 0.88262195]
```

4.2.2.3 *Observations:*

Model	Accuracy Train	Accurac Test	ROC-AUC Train	ROC-AUC Test
Naïve Bayes	0.83	0.83	0.89	0.88
Naïve Bayes with Cross Validation	0.86	0.87	0.88	0.91

Fourth model that we have tried is Naïve Bayes. Here, we used Naïve Bayes on default parameters and with hyper tuned with cross validation. We may see the score has significantly improved after cross validation. The accuracy of train increased from 83% to 86% and for test accuracy increased from 83% to 87%.

However, there is an increase in accuracy but the challenge after cross validation occurred is over-fitting. This is also reflected in the ROC-AUC scores where ROC-AUC score for test data is more than the train data. Hence, out of the two we see the model is performing better without cross validation.

As we may notice, after cross validation, accuracy score (for both train and test) increases from

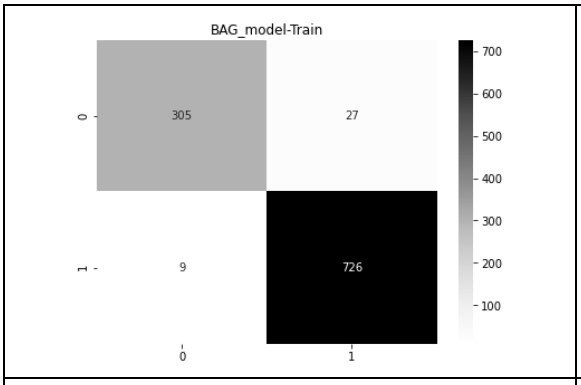
.83 to approx. .85 in two-fold cross validation which further increases to 0.86 and .87 for train and test data respectively in three-fold cross validation but goes down in four-fold back to 0.85. Hence, the best results may be obtained by three-fold cross validation for Naïve Bayes.

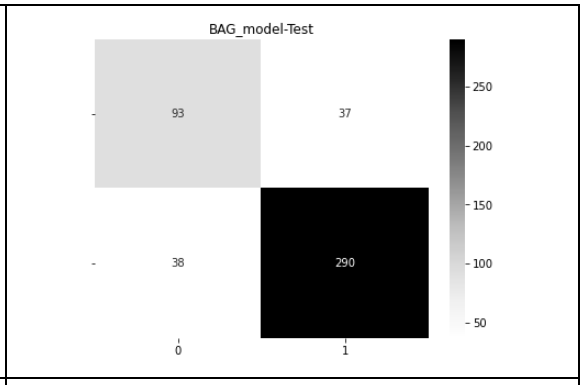
4.3 Bagging (Random Forest should be applied for Bagging) and Boosting. (9 marks)

Next, we apply models based on ensemble techniques for the same data and see the difference in the results.

4.3.1 Bagging

4.3.1.1 *Performance Metrics without hyper tuning: Check the performance of Predictions on Train and Test sets using Accuracy, Confusion Matrix, Plot ROC curve and get ROC_AUC score*



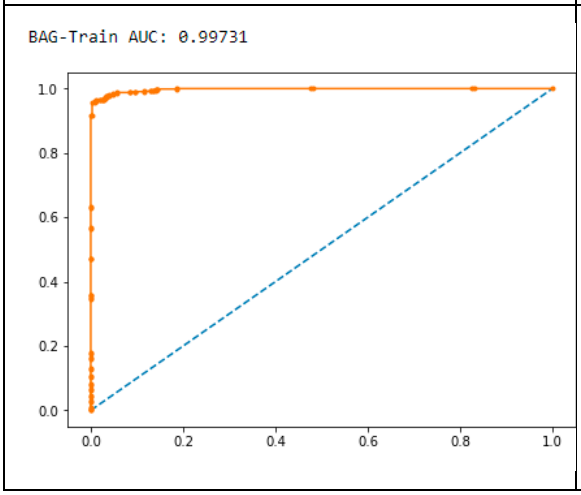


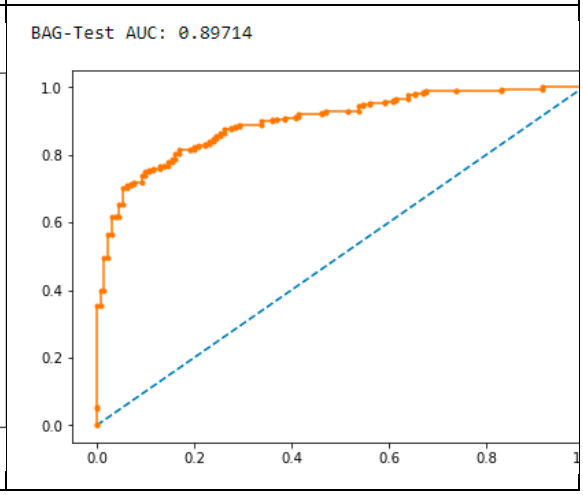
BAG_model-Train

	precision	recall	f1-score	support
0	0.97	0.92	0.94	332
1	0.96	0.99	0.98	735
accuracy			0.97	1067
macro avg	0.97	0.95	0.96	1067
weighted avg	0.97	0.97	0.97	1067

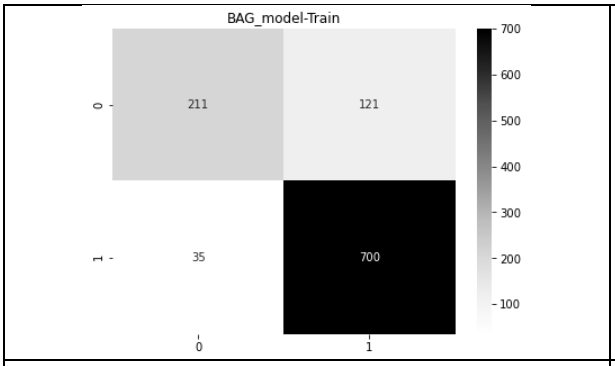
BAG_model-Test

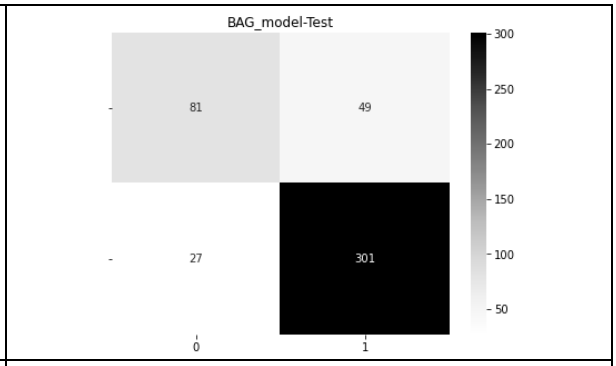
	precision	recall	f1-score	support
0	0.71	0.72	0.71	130
1	0.89	0.88	0.89	328
accuracy			0.84	458
macro avg	0.80	0.80	0.80	458
weighted avg	0.84	0.84	0.84	458





4.3.1.2 *Performance Metrics with hyper tuning: Check the performance of Predictions on Train and Test sets using Accuracy, Confusion Matrix, Plot ROC curve and get ROC_AUC score*



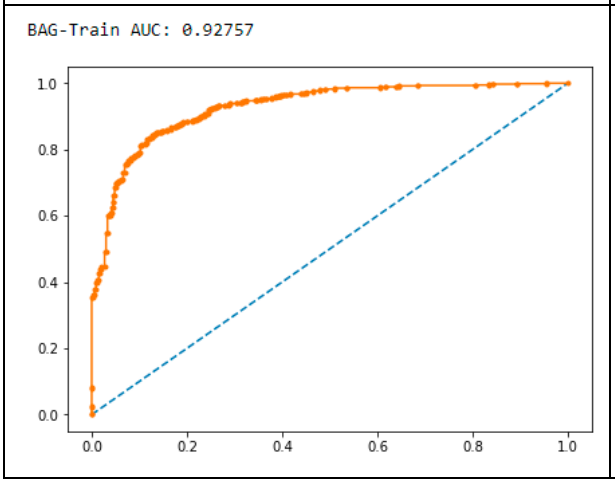


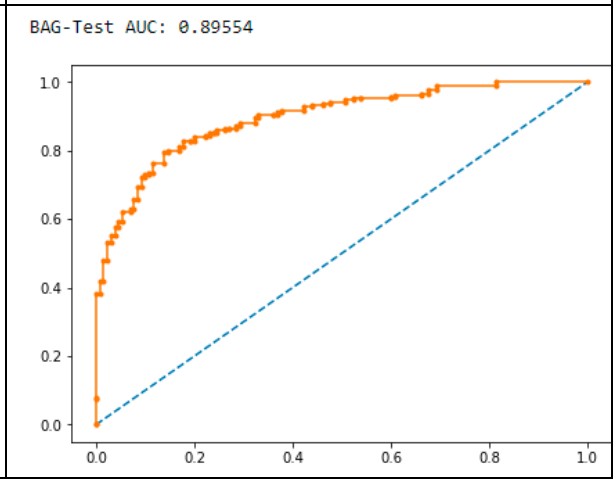
BAG_model-Train

	precision	recall	f1-score	support
0	0.86	0.64	0.73	332
1	0.85	0.95	0.90	735
accuracy			0.85	1067
macro avg	0.86	0.79	0.81	1067
weighted avg	0.85	0.85	0.85	1067

BAG_model-Test

	precision	recall	f1-score	support
0	0.75	0.62	0.68	130
1	0.86	0.92	0.89	328
accuracy			0.83	458
macro avg	0.80	0.77	0.78	458
weighted avg	0.83	0.83	0.83	458





4.3.1.3 *Observations*

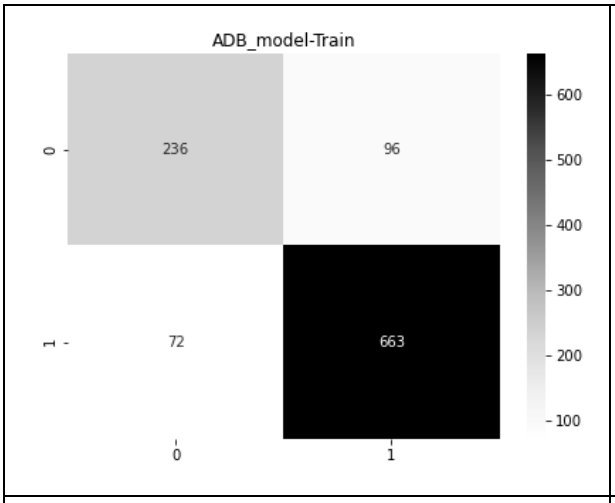
Model	Accuracy Train	Accurac Test	ROC-AUC Train	ROC-AUC Test
Bagging (RF)	0.97	0.84	1.00	0.90
Bagging (RF) with hyper tuning	0.85	0.83	0.93	0.90

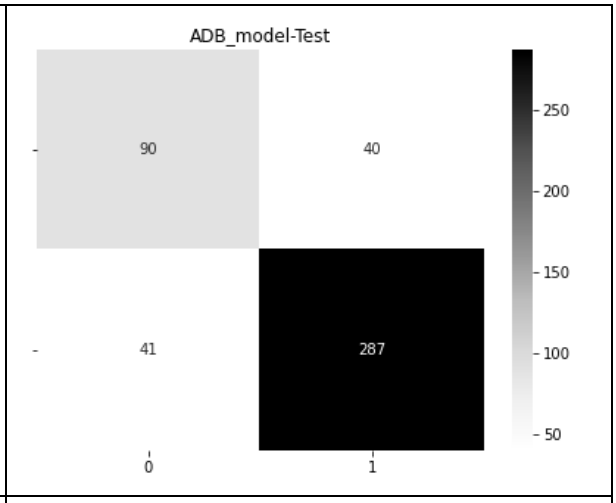
Fifth model that we considered is Bagging using Random Forest Regressor as base estimator. We used Bagging with default parameters first and then applied hyper tuning. Here we may observe that Bagging with default parameters has given a highly under-fitted model. This model has both high bias and high variance (with train and test accuracy of 97%, and 84% respectively). After hyper tuning, the results have improved (with train and test accuracy of

85% and 83% respectively) but the variance is still high in this model. TPR and FPR of this model is also improved from 1.00 and .90 for train and test respectively to .93 and .90. This model does have better accuracy than others but variance should also be kept in mind.

4.3.2 Boosting – ADA Boosting

4.3.2.1 *Performance Metrics without hyper tuning: Check the performance of Predictions on Train and Test sets using Accuracy, Confusion Matrix, Plot ROC curve and get ROC_AUC score*



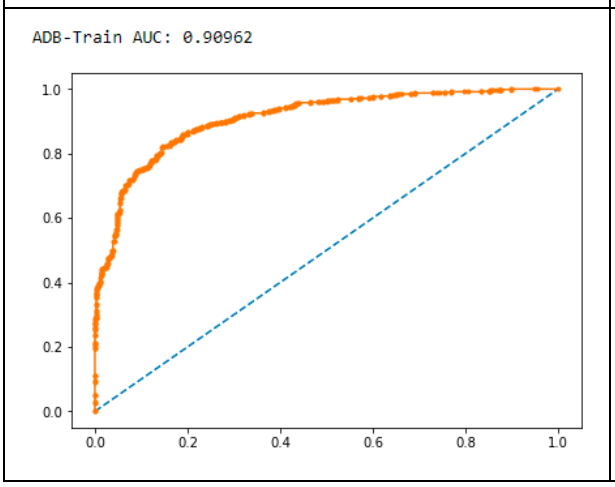


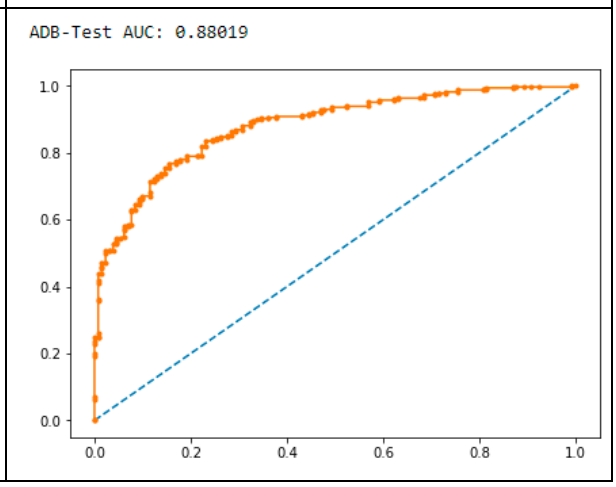
ADB_model-Train

	precision	recall	f1-score	support
0	0.77	0.71	0.74	332
1	0.87	0.90	0.89	735
accuracy			0.84	1067
macro avg	0.82	0.81	0.81	1067
weighted avg	0.84	0.84	0.84	1067

ADB_model-Test

	precision	recall	f1-score	support
0	0.69	0.69	0.69	130
1	0.88	0.88	0.88	328
accuracy			0.82	458
macro avg	0.78	0.78	0.78	458
weighted avg	0.82	0.82	0.82	458





4.3.2.2 *Performance Metrics with hyper tuning: Check the performance of Predictions on Train and Test sets using Accuracy, Confusion Matrix, Plot ROC curve and get ROC_AUC score*

```
ADB_model4 = AdaBoostClassifier(n_estimators=6)
ADB_model4.fit(X_train,y_train)

# evaluate the model
cv = RepeatedStratifiedKFold(n_splits=70, n_repeats=5, random_state=50)
Train_acc = cross_val_score(ADB_model4, X_train, y_train, scoring='accuracy',
                            cv=cv, n_jobs=-1, error_score='raise')
Test_acc = cross_val_score(ADB_model4, X_test, y_test, scoring='accuracy',
                            cv=cv, n_jobs=-1, error_score='raise')
# report performance
print('Train Accuracy: ',np.mean(Train_acc))
print('Test Accuracy: ',np.mean(Test_acc))
```

```
Train Accuracy:  0.8352261904761904
Test Accuracy:  0.8257142857142857
```

```
train_roc_auc = cross_val_score(ADB_model4, X_train, y_train, scoring='roc_auc',
                            cv=cv, n_jobs=-1, error_score='raise')
test_roc_auc = cross_val_score(ADB_model4, X_test, y_test, scoring='roc_auc',
                            cv=cv, n_jobs=-1, error_score='raise')

print('Train ROC-AUC: ',np.mean(train_roc_auc))
print('Test ROC_AUC: ',np.mean(test_roc_auc))
```

```
Train ROC-AUC:  0.8774337662337663
Test ROC_AUC:  0.88
```

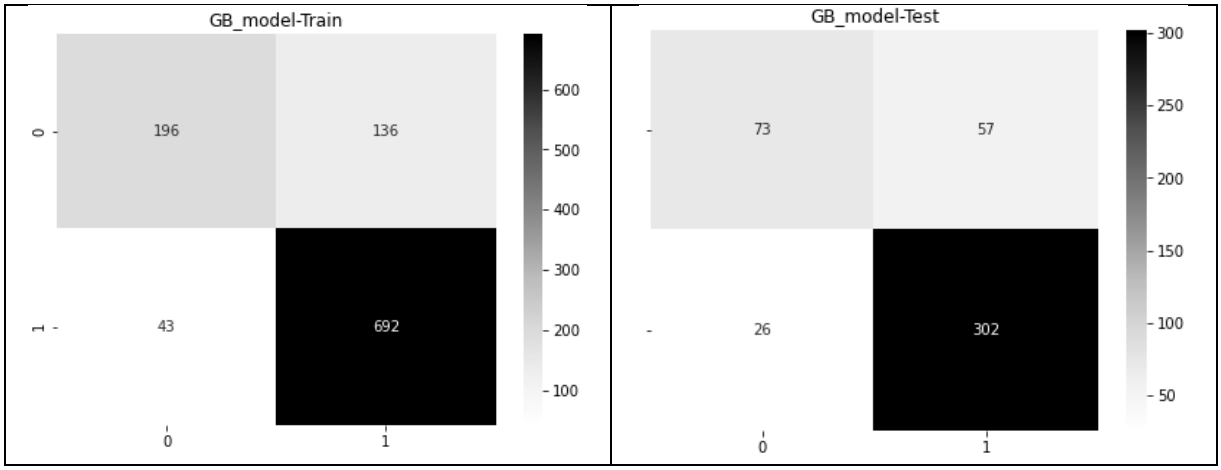
4.3.2.3 *Observations:*

Model	Accuracy Train	Accurac Test	ROC-AUC Train	ROC-AUC Test
ADA Boosting	0.84	0.82	0.91	0.88
ADA Boosting with Cross Validation	0.84	0.83	0.88	0.88

Sixth model that we used is ADA Boosting. We trained the model with default parameters and after cross validation and observed results are improved after cross validation and variance is reduced with same train accuracy. Here the train accuracy remained at 84% both before and after hyper tuning but test accuracy has improved from 82% to 83%. Also, the ROC-AUC score has become more consistent from .91 and .88 for train and test respectively to .88 for both train and test.

4.3.3 Gradient Boosting

4.3.3.1 *Performance Metrics: Check the performance of Predictions on Train and Test sets using Accuracy, Confusion Matrix, Plot ROC curve and get ROC_AUC score*



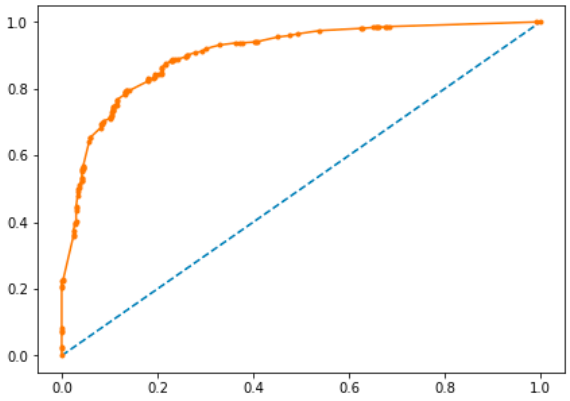
GB_model-Train

	precision	recall	f1-score	support
0	0.82	0.59	0.69	332
1	0.84	0.94	0.89	735
accuracy			0.83	1067
macro avg	0.83	0.77	0.79	1067
weighted avg	0.83	0.83	0.82	1067

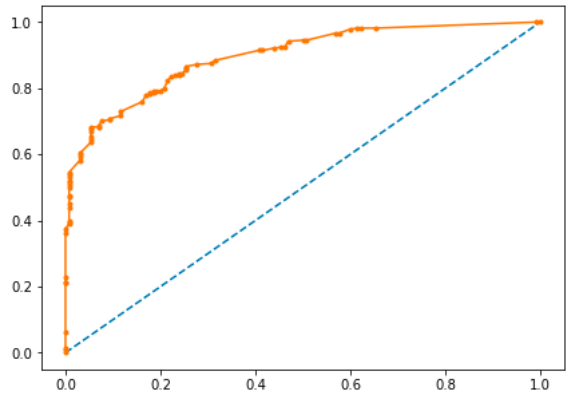
GB_model-Test

	precision	recall	f1-score	support
0	0.74	0.56	0.64	130
1	0.84	0.92	0.88	328
accuracy			0.82	458
macro avg	0.79	0.74	0.76	458
weighted avg	0.81	0.82	0.81	458

GB-Train AUC: 0.90439



GB-Test AUC: 0.89678



4.3.3.2 *Observations:*

Model	Accuracy Train	Accurac Test	ROC-AUC Train	ROC-AUC Test
Gradient Boosting	0.83	0.82	0.90	0.90

The final model that we have tried is Gradient Boosting method. Method gives overall decent performance with train and test accuracy at 83% and 82% and ROC-AUC score of .90 for both test and train.

5 Final Model: Compare the models and write inference which model is best/optimized.

5.1.1.1 *Model Comparison Summary:*

SN	Model	Accuracy Train	Accurac Test	ROC-AUC Train	ROC-AUC Test
1	Logistic Regression	0.76	0.81	0.88	0.92
2	LDA	0.84	0.82	0.89	0.88
3	KNN with optimum k=18	0.83	0.83	0.90	0.90
4	Naïve Bayes	0.83	0.83	0.89	0.88
5	Bagging (RF) with hyper tuning	0.85	0.83	0.93	0.90
6	ADA Boosting with Cross Validation	0.84	0.83	0.88	0.88
7	Gradient Boosting	0.83	0.82	0.90	0.90

- As the data we are analyzing belongs to the poll survey, the **most important metrics** for such business problem is **'accuracy', 'tpr' and 'fpr' which can be assessed by ROC-AUC values.**
- Here, we would also focus on the model with **lowest variance and lowest bias**.
- From the above seven models, we may see that **Logistic and LDA have not performed so well for this data** as compared to other models as both the models have high variance and moderate accuracy in test and train sets. Logistics Regression has a big difference between ROC-AUC score for the train and test data also. In LDA, ROC-AUC score is decently good but other models .02-.03 higher score values.
- Going forward on other models, if we **compare the models by accuracy**, we find **Bagging has highest accuracy for train at 85%** but there is good variance between test and train in this model.
- Next **if we check low variance models**, we find that **Knn ans Naïve Bayes have same accuracy and least variance in the model.** Out of the two, we may see Knn is better performing in ROC-AUC score than Naïve Bayes. So we can consider KNN to be better performing than Naïve Bayes for our dataset.
- From rest of the models, we may observe **Boosting models have a decent performance and very good ROC-AUC score**. Out of ADA and Gradient Boosting, we may see that accuracies are higher for ADA than Gradient Boosting, however, ROC score is better for Gradient Boosting model. The variance in the two models ia similar with train and test differ by 1% in both cases. So, considering these points, we may consider **Gradient Boosting better than ADA** in performance for two resons:
  - In **Gradient Boosting accuracy is only slightly lower, i.e., by 1%** but the ROC score which indicates our TPR and FPR is higher by .02 and that is considerable for predicting problems like polling.
  - **Variance between train and test is same** for both.
- So, from above we see the **two best models by performance are Knn and Gradient Boosting and to bring down the comparison to the one best** we may compare these two again on following:
  - *Accuracy for train* – same in both models at 83%

- *Accuracy for test* – goes down to 82% in Gradient Boosting but remains at 83% for in Knn
- *ROC-AUC* – for both train and test in both the models score is same at .90 which is a very good score industry-wise.
- *Model algorithm* – as we know Knn is an expensive and lazy algorithm, for our business problem, it is not the best suited mode for prediction purposes. Hencewe conclude Gradient Boosting as our model of choice.

6 Based on these predictions, what are the insights?

6.1 Recommendations from the Data Analysis:

- For 'Conservative' party, campaigns should be done to convince people of older age group to vote them as we can see it has maximum voters of age around 40 years but the 75% of voters are above the age of 41 years.
- 'Conservative' party should also check why only people with stronger Eurosceptic sentiments (9-11) are voting more and should plan something to attract voters with lower Eurosceptic sentiments.
- 'Labour' party could focus more on the people with lower 'economic.cond.household' as they are the ones voting more to the 'Conservative' party.
- There are nearly 450 voters from both 'Labour' and 'Conservative' parties with 0 knowledge of parties' position in European integration, this shall be improved so that people are casting more informed votes.

6.2 Learnings from understaking the Project

- Since, the objective of the model is to predict accurate polling sentiments, metrics of importance are accuracy and TPR.
- Out of all the methods used ensemble methods provided more reliable models
- Tree-based model tend to overfit and hence are not sustainable for polling prediction.
- All the models have their strengths and weaknesses and work diffrently with different datasets. In this project, we have used different types of models but each had its limitation in real application on new data which should be considered while choosing the correct model.
- It is always recommended to do sincere work on data cleaning and pre-processing first before applying the models.
- Evaluation of model is very important while selecting the model and knowing the right evaluation metrics for your business problem is the key.

6.3 Conclusion

- Based on above our model of choice is Gradient Boosting and it has given very good results on following evaluation metrics:
  - Excellent ROC-AUC value at .90.
  - A decent train and test accuracy of 83% and 82% respectively
  - Low variance between train and test results.