Arvind Maurya – AIMLCEP-Batch03

Assignment report for Q2:

Location of Assignment in Github:

https://github.com/arvind-maurya/MachineLearning/blob/master/AIMLCEP_ARVIND_MAURYA_ASSIGNMENT/AIMLCEP_ARVIND_MAURYA_Q2.ipynb

---

2 (a) [C, R] Some of the attributes in the data set are text data. Use a suitable procedure to convert them into suitable numerical representations in the training data and test data. Explain the procedure you used for the conversion.

```
spam_class_train_data = pd.read_excel('https://github.com/arvind-maurya/MachineLearning/blob/master/Assignment/Q2/Q2_classification_train.xlsx?raw=true', index_col=F

#print the first few rows of the data
spam_class_train_data.head()
```

from	cc	sent_email	image	attach	dollar	winner	inherit	viagra	password	num_char	line_breaks	format	re_subj	exclaim_subj	urgent_subj	exclaim_mess	number
1	0	0	0	0	0	no	0	0	0	11.370	202	1	0	0	0	0	big
1	0	0	0	0	0	no	0	0	0	10.504	202	1	0	0	0	1	small
1	0	0	0	0	0	no	0	0	0	13.256	255	1	0	0	0	48	small
1	0	0	0	0	0	no	0	0	2	1.231	29	0	0	0	0	1	none
1	0	0	0	0	0	no	0	0	2	1.091	25	0	0	0	0	1	none

After loading the dataset, we found that winner and number are text data (see red highlighted). For classification to work correctly, we need to convert them to numerical representation

To convert them to numerical value, we are going to use **LabelEncoder** provided by library scikit-learn

```
[ ]  from sklearn.preprocessing import LabelEncoder
     gle = LabelEncoder()
     #transform winner column into numerical column
     winner_tf = gle.fit_transform(spam_class_train_data['winner'])
     winner_mappings = {index: label for index, label in
                        enumerate(gle.classes_)}
     winner_mappings
```

```
{0: 'no', 1: 'yes'}
```

```
number_tf = gle.fit_transform(spam_class_train_data['number'])
number_mappings = {index: label for index, label in
                   enumerate(gle.classes_)}
number_mappings
```

```
{0: 'big', 1: 'none', 2: 'small'}
```

```
[ ]  spam_class_train_data['winner_tf'] = winner_tf
     spam_class_train_data['number_tf'] = number_tf
     spam_class_train_data
```

image	attach	dollar	winner	inherit	viagra	password	num_char	line_breaks	format	re_subj	exclaim_subj	urgent_subj	exclaim_mess	number	winner_tf	number_tf
0	0	0	no	0	0	0	11.370	202	1	0	0	0	0	big	0	0
0	0	0	no	0	0	0	10.504	202	1	0	0	0	1	small	0	2
0	0	0	no	0	0	0	13.256	255	1	0	0	0	48	small	0	2
0	0	0	no	0	0	2	1.231	29	0	0	0	0	1	none	0	1
0	0	0	no	0	0	2	1.091	25	0	0	0	0	1	none	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
0	0	2	no	0	0	0	1.597	46	0	0	1	0	3	small	0	2
0	0	1	no	0	0	0	0.333	13	0	0	0	0	0	big	0	0
0	0	0	no	0	0	0	0.332	12	0	0	0	0	0	small	0	2
0	0	2	yes	0	0	0	2.225	65	0	0	1	0	1	small	1	2
0	0	1	no	0	0	0	0.323	15	0	0	0	0	0	small	0	2

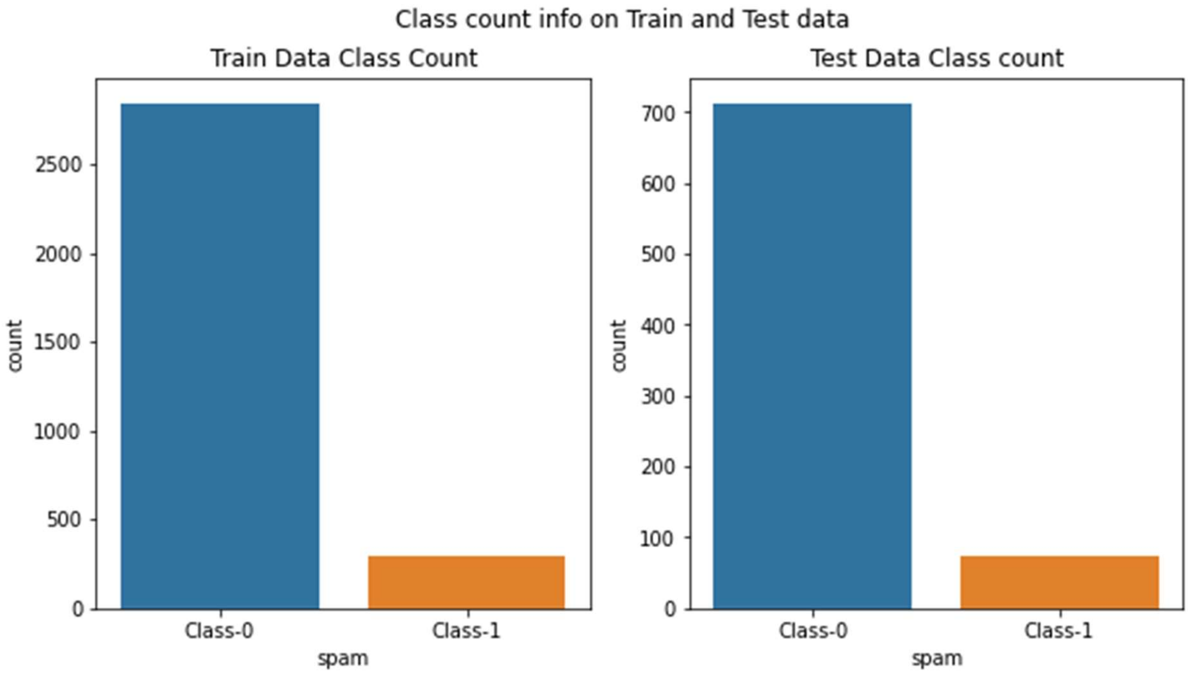
The numerical column is again inserted back to the data frame.

2. (c) [C, R] How did you handle class imbalance when building the classification models? Explain.

When we analyse the train and test dataset, we found that we have class imbalance.

Original Dataset class imbalance info:

Class/Dataset	Train Data Class count	Test Data class count
Class-0	2842	712
Class-1	294	73



We can see the dataset is highly imbalance and various classifier wouldpredict class-0 with high accuracy totally ignoring the class-1. To avoid such situation, we need to balance the data using various technique available.

We will use the Synthetic Minority Oversampling Technique (SMOTE) to generates synthetic data for the minority class. SMOTE (Synthetic Minority Oversampling Technique) works by randomly picking a point from the minority class and computing the k-nearest neighbours for this point. The synthetic points are added between the chosen point and its neighbours.

SMOTE algorithm works in 4 simple steps:

1. Choose a minority class as the input vector
2. Find its k nearest neighbours (k_neighbors is specified as an argument in the SMOTE() function)
3. Choose one of these neighbours and place a synthetic point anywhere on the line joining the point under consideration and its chosen neighbor
4. Repeat the steps until data is balanced

Post SMOTE algorithm application, we can see the data is balanced.

```
# import library
from imblearn.over_sampling import SMOTE
from collections import Counter
y_train_counter = Counter(y_train.ravel())
print('Original train dataset shape', y_train_counter)
smote = SMOTE(random_state=0)

# fit predictor and target variable
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

print('Resample train dataset shape', Counter(y_train_smote))
```

```
Original train dataset shape Counter({0: 2842, 1: 294})
Resample train dataset shape Counter({0: 2842, 1: 2842})
```

```
[ ] y_test_counter = Counter(y_test.ravel())
    print('Original test dataset shape', y_test_counter)
    smote = SMOTE(random_state=0)

    # fit predictor and target variable
    X_test_smote, y_test_smote = smote.fit_resample(X_test, y_test)

    print('Resample test dataset shape', Counter(y_test_smote))

    n_train_smote = len(X_train_smote)
    n_test_smote = len(X_test_smote)

    print(n_train_smote,n_test_smote)
```

```
Original test dataset shape Counter({0: 712, 1: 73})
Resample test dataset shape Counter({1: 712, 0: 712})
5684 1424
```

With above we see that we have balanced both the train and test dataset.

Balanced Dataset class count info:

Class/Dataset	Train Data Class count	Test Data class count
Class-0	2842	712
Class-1	2842	712

---

2. (d) [C, R] For each of the classification model you have built, explain how you chose the best values of hyperparameters used in the respective classification method.

1. Naïve Bayes hyper parameter

GaussianNB take parameter var_smoothing

var_smoothing is a stability calculation to widen (or smooth) the curve and therefore account for more samples that are further away from the distribution mean. In this case, np.logspace returns numbers spaced evenly on a log scale, starts from 0, ends at -9, and generates 100 samples.

```
#Hyper parameter tunning for Naive Bayes
param_grid_nb = {
    'var_smoothing': np.logspace(0,-9, num=100)
}
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV
nbModel_grid = GridSearchCV(estimator=GaussianNB(), param_grid=param_grid_nb, verbose=1, cv=5, n_jobs=-1)
nbModel_grid.fit(X_train, y_train)
print(nbModel_grid.best_estimator_)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
GaussianNB(var_smoothing=1.0)
```

2. Logistic Regression hyperparameter tunning

```
#HyperParameter Tunning
from sklearn.linear_model import LogisticRegression

solvers = ['newton-cg','lbfgs','liblinear']
c_values = np.array([1e6,1e3,100,10,1.0,0.1,0.01])
best_c_value = {}
cv_k = 5 #5-fold cross validation
for solver in solvers:
  avg_score = np.zeros(len(c_values))
  # print (avg_score)
  for c_value in c_values:
    clf_lr = LogisticRegression(solver = solver, C = c_value, max_iter=100000, random_state=0)
    scores = cross_val_score(clf_lr, X_train_lrm, y_train_lrm.ravel(), cv=cv_k)
    # print ('scores',scores)
    avg_score[np.where(c_values==c_value)] = np.mean(scores)

  # print ('avg score',avg_score)
  max_score_index = np.argmax(avg_score)
  # print (max_score_index)
  best_c_value[solver] = c_values[int(max_score_index)]

print ('Best C value = ', best_c_value)
```

```
Best C value =  {'newton-cg': 1000.0, 'lbfgs': 100.0, 'liblinear': 1000.0}
```

```
[156] #Train the model
      from sklearn.linear_model import LogisticRegression
      logmodel = LogisticRegression(solver = 'lbfgs', C=100, max_iter=100000, random_state=0)
      logmodel.fit(X_train_lrm, y_train_lrm)

      #Predict the y test
      y_train_predictions_lrm = logmodel.predict(X_train_lrm)
      y_test_predictions_lrm = logmodel.predict(X_test_lrm)
```

We see some improvement with the above hyper parameter.

3. Hyper parameter Soft Margin SVM

```
#Hyperparameter tunning
from sklearn import svm
from sklearn.svm import LinearSVC #linear svm from scikit learn

c_values = np.array([1e6,1e3,100,10,1.0,0.1,0.01,0.001])
best_c_value = []
cv_k = 5 #5-fold cross validation
avg_score = np.zeros(len(c_values))
for c_value in c_values:
  clf_svm = LinearSVC(C = c_value, max_iter=10000, random_state=0, tol=1e-5,verbose=0 )
  scores = cross_val_score(clf_svm, train_features, train_label.ravel(), cv=cv_k)
  avg_score[np.where(c_values==c_value)] = np.mean(scores)

  max_score_index = np.argmax(avg_score)

  best_c_value = c_values[int(max_score_index)]

print ('Best C value = ', best_c_value)
```

Output:

Best C value = 0.01

4. Hyper parameter for Decision Tree

```
#Lets tune the best depth by tuning some hyperparameter
criteria = ['entropy', 'gini']
max_depth = np.array([1,2,3,5,10,15,20,25])
best_depth = {}
cv_k = 5 #5-fold cross validation
for criterion in criteria:
  avg_score = np.zeros(len(max_depth))

  for depth in max_depth:
    clf_dt = tree.DecisionTreeClassifier(criterion=criterion, max_depth=depth, random_state=0)
    scores = cross_val_score(clf_dt, train_features , train_label.ravel(), cv=cv_k)

    avg_score[np.where(max_depth==depth)] = np.mean(scores)

  print('Criterion:', criterion)
  print ('Avg score',avg_score)
  max_score_index = np.argmax(avg_score)

  best_depth[criterion] = max_depth[int(max_score_index)]

print ('Best hyperparameter for tree depth = ', best_depth)
```

```
Criterion: entropy
Avg score [0.64689819 0.76424787 0.84306304 0.86558261 0.89144648 0.91642898
 0.91871508 0.91818738]
Criterion: gini
Avg score [0.71463451 0.81016714 0.83725721 0.86030386 0.89373304 0.91114884
 0.91466703 0.91255652]
Best hyperparameter for tree depth =  {'entropy': 20, 'gini': 20}
```

```
[ ] clf = tree.DecisionTreeClassifier(criterion='entropy',max_depth=20)
    #train using decision tree classifier and plot the resultant decision tree
    # tree.plot_tree(clf.fit(train_features, train_label))
```

5. Hyperparameter for Random Forest Classification

```
#Let us now use cross validation to find random forest hyperparameters.
# We will first find best max depths for a given set of estimators
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
import math
import numpy as np

num_features = X_train.shape[1]
estimators = [5, 10,50,100,150,200]
constant = math.sqrt(num_features)
max_depth = constant*np.array([0.25,0.5, 0.75,1,1.25,1.50,1.75,2])
max_depth = np.ceil(max_depth)
print ('maximum depth', max_depth)
best_depth = {}
cv_k = 5 #5-fold cross validation
for n_estimate in estimators:
  avg_score = np.zeros(len(max_depth))
  # print (avg_score)
  for depth in max_depth:
    clf_rf = RandomForestClassifier(n_estimators = n_estimate,max_depth = depth,  random_state=0)
    scores = cross_val_score(clf_rf, X_train, y_train.ravel(), cv=cv_k)
    # print ('scores',scores)
    avg_score[np.where(max_depth==depth)] = np.mean(scores)

  # print ('avg score',avg_score)
  max_score_index = np.argmax(avg_score)
  # print (max_score_index)
  best_depth[n_estimate] = max_depth[int(max_score_index)]

print ('maximum depth = ', best_depth)
```

```
maximum depth [2. 3. 4. 5. 6. 7. 8. 9.]
maximum depth =  {5: 9.0, 10: 9.0, 50: 9.0, 100: 9.0, 150: 9.0, 200: 9.0}
```

6. Kernel Machine hyperparameter

```
#Finding the best Hyperparameter
from sklearn.svm import SVC

gammas = [0.0001,0.001, 0.01, 0.1, 1,2, 3,5, 20, 40, 60, 80, 100]
kernels = ['sigmoid','rbf']
best_gamma = {}
cv_k = 5 #5-fold cross validation
for i_kernel in kernels:
  print ('kernel :',i_kernel)
  avg_score = np.zeros(len(gammas))
  for i_gamma in gammas:
    clf = SVC(kernel=i_kernel, gamma=i_gamma, random_state=1)
    scores = cross_val_score(clf, X_train, y_train, cv=cv_k)
    avg_score[gammas.index(i_gamma)] = np.mean(scores)

  print ('Average_score:', avg_score)
  max_score_index = np.argmax(avg_score)

  best_gamma[i_kernel] = gammas[int(max_score_index)]

print ('Best hyper-parameters for Kernel Machine = ', best_gamma)
```

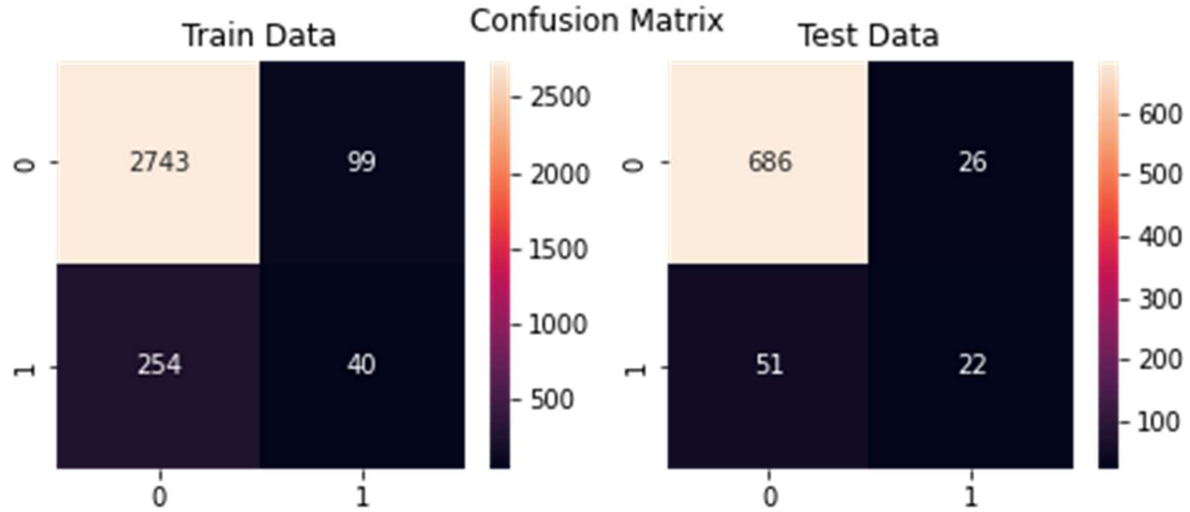
```
kernel : sigmoid
Average_score: [0.61171572 0.70742208 0.37127495 0.38335284 0.49647918 0.49647918
 0.49331017 0.49894397 0.4996482  0.4996482  0.4996482  0.4996482
 0.4996482 ]
kernel : rbf
Average_score: [0.74823944 0.79045998 0.86030665 0.89496466 0.89514568 0.87368105
 0.87280123 0.87121781 0.86505986 0.86013369 0.85591144 0.85204067
 0.84957805]
Best hyper-parameters for Kernel Machine =  {'sigmoid': 0.001, 'rbf': 1}
```

```
[ ] #Implement Gaussian Kernel
from sklearn.svm import SVC
svclassifier = SVC(kernel='rbf',gamma=1, max_iter = 10000, random_state = 0 )
svclassifier.fit(X_train, y_train)
```

3.(e) [C] Report the following performance metrics for each of the classification model:

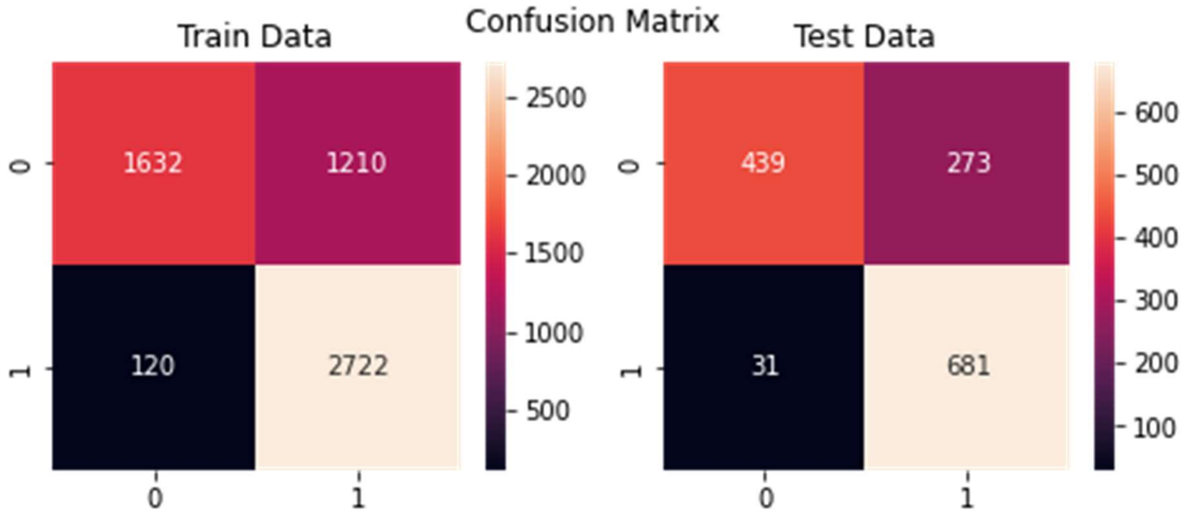
Naive Bayes Classifier statistics with original dataset:

```
=======================================================
Train Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.97      0.94      2842
           1       0.29      0.14      0.18       294

    accuracy                           0.89      3136
   macro avg       0.60      0.55      0.56      3136
weighted avg       0.86      0.89      0.87      3136

=======================================================
Test Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.96      0.95       712
           1       0.46      0.30      0.36        73

    accuracy                           0.90       785
   macro avg       0.69      0.63      0.66       785
weighted avg       0.89      0.90      0.89       785

=======================================================
Train confusion matrix:
 [[2743   99]
 [ 254   40]]
Test confusion matrix:
 [[686  26]
 [ 51  22]]
```



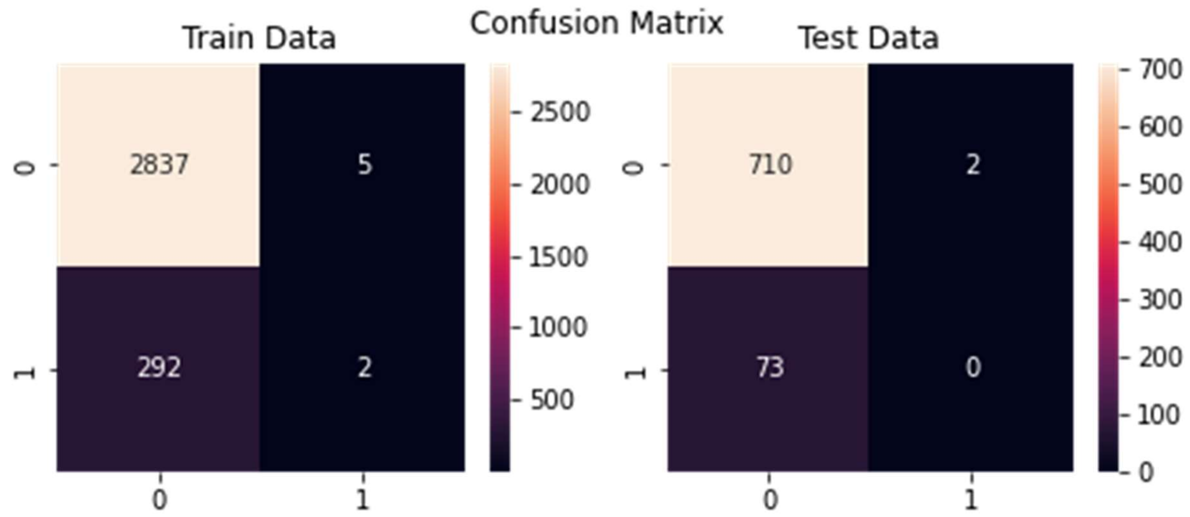
Naive Bayes Classifier statistics with Class Balanced Dataset:

```
=======================================================
Train Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.57      0.71      2842
           1       0.69      0.96      0.80      2842

    accuracy                           0.77      5684
   macro avg       0.81      0.77      0.76      5684
weighted avg       0.81      0.77      0.76      5684

=======================================================
Test Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.62      0.74       712
           1       0.71      0.96      0.82       712

    accuracy                           0.79      1424
   macro avg       0.82      0.79      0.78      1424
weighted avg       0.82      0.79      0.78      1424

=======================================================
Train confusion matrix:
 [[1632 1210]
 [ 120 2722]]
Test confusion matrix:
 [[439 273]
 [ 31 681]]
```



Naive Bayes Classifier statistics with reduced feature dataset:

```
======================================================
Train Classification Report:
              precision    recall  f1-score   support

           0       0.91      1.00      0.95      2842
           1       0.29      0.01      0.01       294

    accuracy                           0.91      3136
   macro avg       0.60      0.50      0.48      3136
weighted avg       0.85      0.91      0.86      3136

======================================================
Test Classification Report:
              precision    recall  f1-score   support

           0       0.91      1.00      0.95       712
           1       0.00      0.00      0.00        73

    accuracy                           0.90       785
   macro avg       0.45      0.50      0.47       785
weighted avg       0.82      0.90      0.86       785

======================================================
Train confusion matrix:
 [[2837    5]
 [ 292    2]]
Test confusion matrix:
 [[710   2]
 [ 73   0]]
```



Conclusion : Reducing the fetaure have hardly any effect on the accurary prediction.

Logistic Regression Model with Class Balanced Dataset:

==
Train Classification Report:

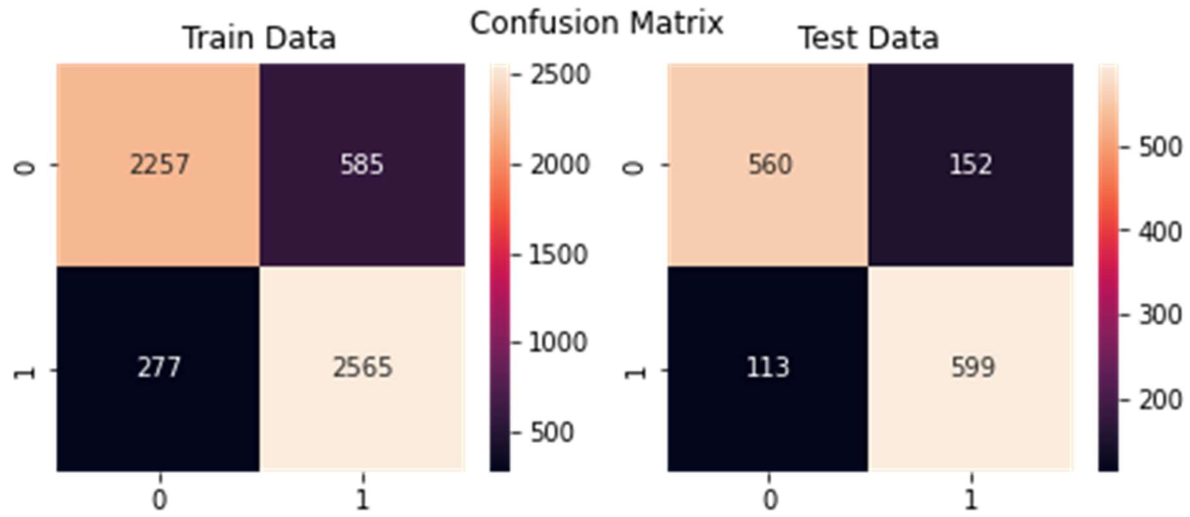
	precision	recall	f1-score	support
0	0.89	0.79	0.84	2842
1	0.81	0.90	0.86	2842
accuracy			0.85	5684
macro avg	0.85	0.85	0.85	5684
weighted avg	0.85	0.85	0.85	5684

==
Test Classification Report:

	precision	recall	f1-score	support
0	0.83	0.79	0.81	712
1	0.80	0.84	0.82	712
accuracy			0.81	1424
macro avg	0.81	0.81	0.81	1424
weighted avg	0.81	0.81	0.81	1424

==

```
Train confusion matrix:
 [[2257  585]
 [ 277 2565]]
Test confusion matrix:
 [[560 152]
 [113 599]]
```



Soft Margin SVM Statistics with balanced dataset:

==

Train Classification Report:

	precision	recall	f1-score	support
0	0.91	0.75	0.82	2842
1	0.79	0.93	0.85	2842
accuracy			0.84	5684
macro avg	0.85	0.84	0.84	5684
weighted avg	0.85	0.84	0.84	5684

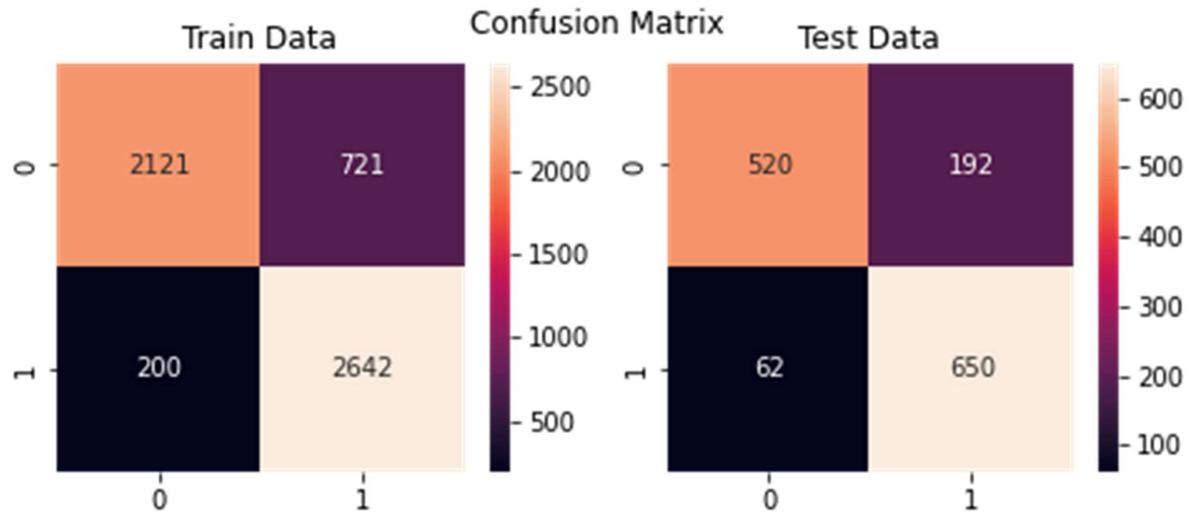
==

Test Classification Report:

	precision	recall	f1-score	support
0	0.89	0.73	0.80	712
1	0.77	0.91	0.84	712
accuracy			0.82	1424
macro avg	0.83	0.82	0.82	1424
weighted avg	0.83	0.82	0.82	1424

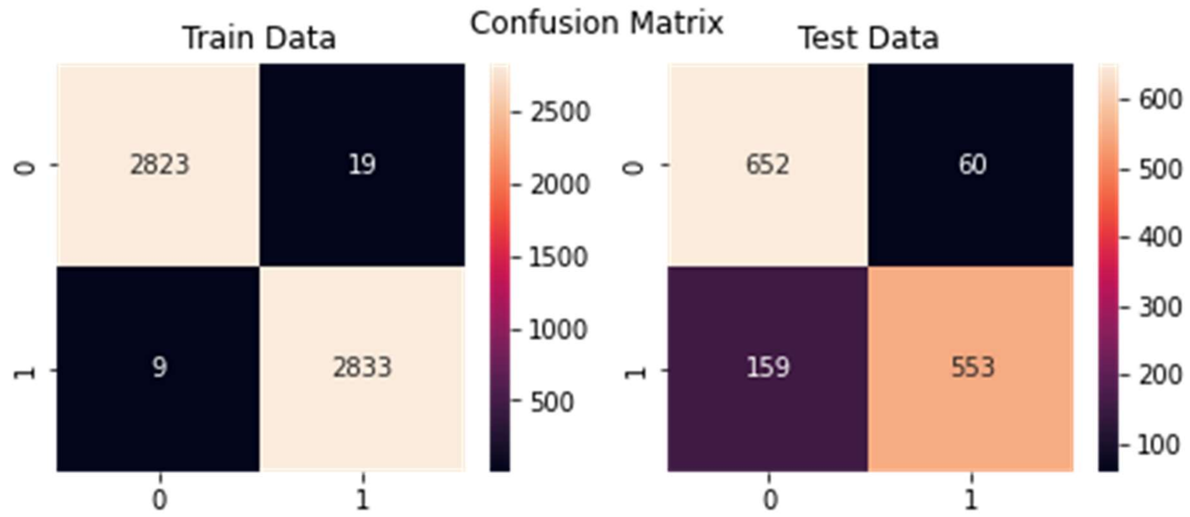
==

```
Train confusion matrix:
 [[2121  721]
 [ 200 2642]]
Test confusion matrix:
 [[520 192]
 [ 62 650]]
```



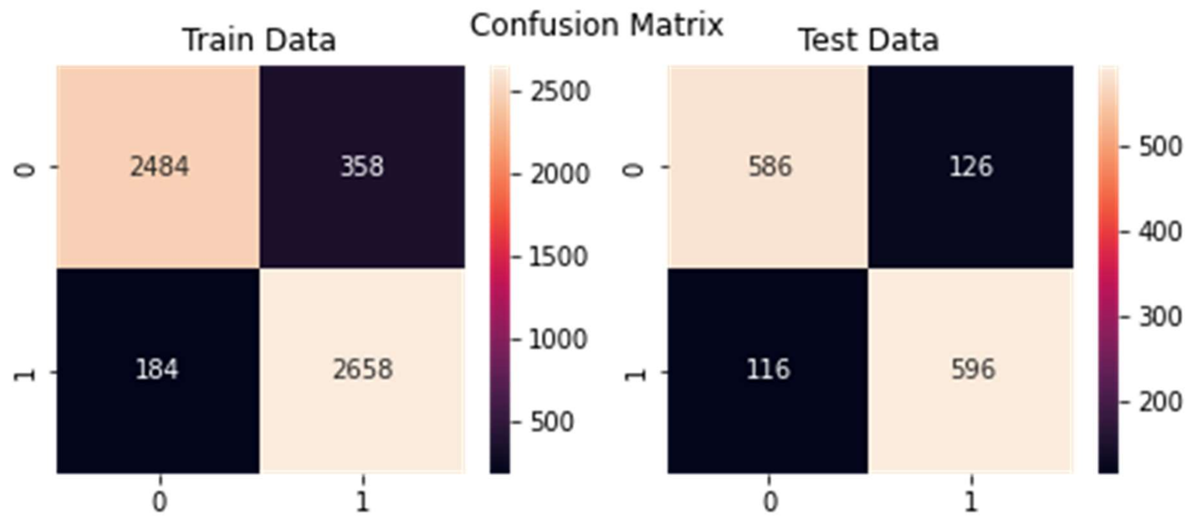
Decision Tree Statistics with balanced dataset:

```
=======================================================
Train Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      2842
           1       0.99      1.00      1.00      2842

    accuracy                           1.00      5684
   macro avg       1.00      1.00      1.00      5684
weighted avg       1.00      1.00      1.00      5684

=======================================================
Test Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.92      0.86       712
           1       0.90      0.78      0.83       712

    accuracy                           0.85      1424
   macro avg       0.85      0.85      0.85      1424
weighted avg       0.85      0.85      0.85      1424

=======================================================
Train confusion matrix:
 [[2823   19]
 [   9 2833]]
Test confusion matrix:
 [[652  60]
 [159 553]]
```



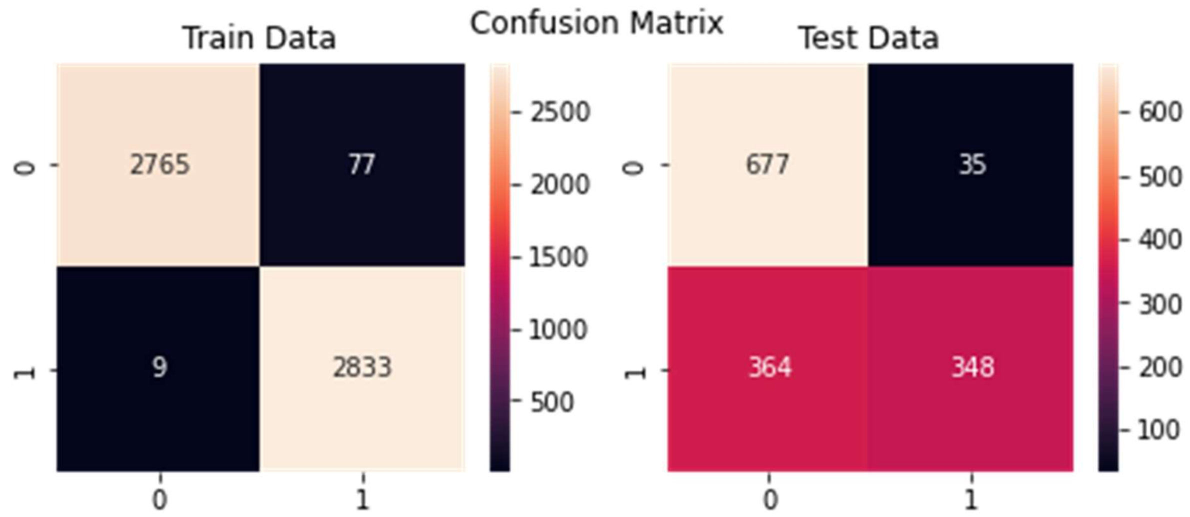
Random Forest Classifier Statistics with balanced dataset:

```
=======================================================
Train Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.87      0.90      2842
           1       0.88      0.94      0.91      2842

    accuracy                           0.90      5684
   macro avg       0.91      0.90      0.90      5684
weighted avg       0.91      0.90      0.90      5684

=======================================================
Test Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.82      0.83       712
           1       0.83      0.84      0.83       712

    accuracy                           0.83      1424
   macro avg       0.83      0.83      0.83      1424
weighted avg       0.83      0.83      0.83      1424

=======================================================
Train confusion matrix:
 [[2484  358]
 [ 184 2658]]
Test confusion matrix:
 [[586 126]
 [116 596]]
```

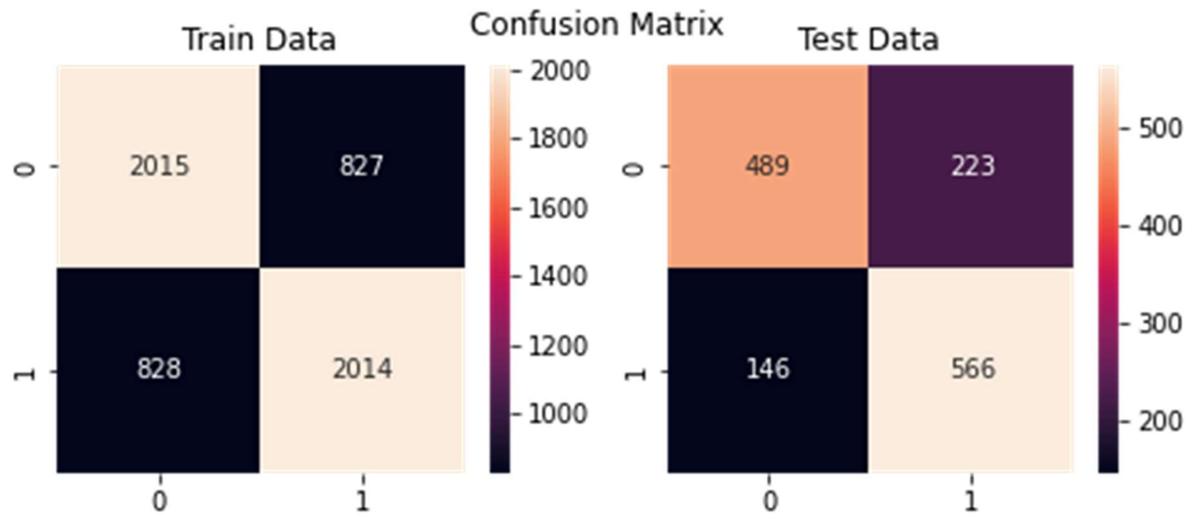


Kernel Machine Classifier Statistics with balanced dataset with rbf kernel:

```
==========================================================
Train Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.97      0.98      2842
           1       0.97      1.00      0.99      2842

    accuracy                           0.98      5684
   macro avg       0.99      0.98      0.98      5684
weighted avg       0.99      0.98      0.98      5684

==========================================================
Test Classification Report:
              precision    recall  f1-score   support

           0       0.65      0.95      0.77       712
           1       0.91      0.49      0.64       712

    accuracy                           0.72      1424
   macro avg       0.78      0.72      0.70      1424
weighted avg       0.78      0.72      0.70      1424

==========================================================
Train confusion matrix:
 [[2765   77]
 [   9 2833]]
Test confusion matrix:
 [[677  35]
 [364 348]]
```



```
Kernel Machine Classifier Statistics with balanced dataset with sigmoid ker
nel:

======================================================
Train Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.71      0.71      2842
           1       0.71      0.71      0.71      2842

    accuracy                           0.71      5684
   macro avg       0.71      0.71      0.71      5684
weighted avg       0.71      0.71      0.71      5684

======================================================
Test Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.69      0.73       712
           1       0.72      0.79      0.75       712

    accuracy                           0.74      1424
   macro avg       0.74      0.74      0.74      1424
weighted avg       0.74      0.74      0.74      1424

======================================================
Train confusion matrix:
 [[2015  827]
 [ 828 2014]]
Test confusion matrix:
 [[489 223]
 [146 566]]
```



2. (f) [C, R] Design a suitable normalization scheme so that the attributes cc, num_char,line_breaks and exclaim mess are scaled between 0 and 1 in both train and test data sets. Explain the normalization scheme you designed. Rebuild the classification models with this modified data set and compare the performance metrics obtained for this modified data set with those obtained before. Using the comparison, comment whether normalization helps in improving the performance metrics for the test data set for each classification method.

Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.

Here's the formula for normalization:

X′=(X−Xmin) / (Xmax−Xmin)

Normalization equation

Here, Xmax and Xmin are the maximum and the minimum values of the feature respectively.

When the value of X is the minimum value in the column, the numerator will be 0, and hence X′ is 0 On the other hand, when the value of X is the maximum value in the column, the numerator is equal to the denominator and thus the value of X′ is 1 If the value of X is between the minimum and the maximum value, then the value of X′ is between 0 and 1

Naive Bayes Classifier statistics with Class Balanced Dataset with Normaliz ation:

Test Classification Report:

	precision	recall	f1-score	support
0	0.92	0.63	0.75	712
1	0.72	0.95	0.82	712
accuracy			0.79	1424
macro avg	0.82	0.79	0.78	1424
weighted avg	0.82	0.79	0.78	1424

==

Observation:

1. **Accuracy is bit lower**
2. **Both classes are predicted where in original dataset class-1 predication rate is very poor.**
3. **Prediction is comparable with the balanced dataset.**

Logistic Regression Model with Class Balance Dataset and normalization:

==

Test Classification Report:

	precision	recall	f1-score	support
0	0.84	0.78	0.81	712
1	0.79	0.85	0.82	712
accuracy			0.82	1424
macro avg	0.82	0.82	0.82	1424
weighted avg	0.82	0.82	0.82	1424

==

Observation:

1. **Performance remains same when compared with class balanced data set.**
2. **However, when compared with original data set performance is good.**

Soft Margin SVM Statistics with balanced dataset and Normalization:

==

Test Classification Report:

	precision	recall	f1-score	support
0	0.85	0.79	0.82	712
1	0.80	0.86	0.83	712
accuracy			0.83	1424
macro avg	0.83	0.83	0.83	1424
weighted avg	0.83	0.83	0.83	1424

==

Observation: Performance remain same

Decision Tree Statistics with balanced dataset and normalization:

==

Test Classification Report:

	precision	recall	f1-score	support
0	0.59	0.53	0.56	712
1	0.58	0.63	0.60	712
accuracy			0.58	1424
macro avg	0.58	0.58	0.58	1424
weighted avg	0.58	0.58	0.58	1424

==

Observation: High Decrease in performance

Random Forest Classifier Statistics with balanced dataset and normalization

==

Test Classification Report:

	precision	recall	f1-score	support
0	0.83	0.82	0.83	712
1	0.82	0.83	0.83	712
accuracy			0.83	1424
macro avg	0.83	0.83	0.83	1424
weighted avg	0.83	0.83	0.83	1424

==

Observation: No variation in performance observed

Kernel Machine Classifier Statistics with balanced dataset with rbf kernel and normalization:

==

Test Classification Report:

	precision	recall	f1-score	support
0	0.94	0.70	0.80	712
1	0.76	0.95	0.85	712
accuracy			0.83	1424
macro avg	0.85	0.83	0.82	1424
weighted avg	0.85	0.83	0.82	1424

==

Result: Observed increase in performance

Kernel Machine Classifier Statistics with balanced dataset with sigmoid kernel and normalization:

===

Test Classification Report:

	precision	recall	f1-score	support
0	0.98	0.45	0.62	712
1	0.64	0.99	0.78	712
accuracy			0.72	1424
macro avg	0.81	0.72	0.70	1424
weighted avg	0.81	0.72	0.70	1424

===

Result: Slight increase in the accuracy with class prediction rate is decreased.