Customer Churn Analysis

Introduction :

Churn prediction is of the most popular Big Data use cases in business.It consists of detecting customers who are likely to cancel a subscription to a service.Churn is a problem for telecom companies because it is more expensive to acquire a new customer than to keep your existing one from leaving.

Customer churn has become highly important for companies because of increasing competition among companies, increased importance of marketing strategies and conscious behavior of costumers in the recent years. Customers can easily trend toward alternative services. Companies must develop various strategies to prevent these possible trends, depending on the services they provide.

Data Pre-processing Done:

Classification tends to be in favor of majority classes when there exists unbalance in the dataset. Distribution of the used dataset has 14.3% churn customers and 85.7% non-churn customers. For this reason down sampling process is applied to dataset. In the down sampling process subsets are generated that will have x times churn customer and 2x times non churn customer. To generate subsets, firstly 20 empty sets are created and all churn customers are added to these sets. Non churn customers are selected as randomly.

- Hardware and Software Requirements and Tools Used
- Tools: Python 3.8.5, Jupyter Notebook, Numpy, Pandas, Matplotlib, Seaborn, Scikit-learn, Scipy
- Techniques: Logistic Regression, Random Forest Classifier, Ada Boosting classifier, Gradient Boosting Classifier, SVC, Decision Tree Classifier, KNeighbors Classifier
- Hardware: I3 processor, 4GB RAM

- Model/s Development and Evaluation

I have used the below models for Classifier:

1. SVC
2. Random Forest Classifier
3. Decision Tree Classifier
4. KNeighbors Classifier
5. Ada Boost Classifier
6. Gradient Boosting Classifier
7. Logistic Regression

- Identification of possible problem-solving approaches (methods)

8. Read the data (from csv)
9. Identify the dependent and independent variables.
10. Check if the data has missing values or the data is categorical or not.
11. If yes, apply basic data preprocessing operations to bring the data in a go to go format.
12. Now split the data into the groups of training and testing for the respective purpose.
13. After splitting data, fit it to a most suitable model. (How to find a suitable model is answered below)
14. Validate the model. If satisfactory, then go with it, else tune the parameters and keep testing. In a few cases, you can also try different algorithms for the same problem to understand the difference between the accuracies.
15. From step 7 one can also learn about accuracy paradox.
16. Visualize the data.

- Testing of Identified Approaches (Algorithms)

1. KNN
2. Naive Bayes
3. Cross validation
4. Confusion matrix
5. Accuracy score

- Run and Evaluate selected models

I have used the below models for classification:

```
# Running logistic regression model

lg = LogisticRegression()
result = lg.fit(x_train, y_train)
pred=lg.predict(x_test)
lg_accu=accuracy_score(y_test,pred)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.8021769995267393
[[1404  142]
 [ 276  291]]
              precision    recall  f1-score   support

           0       0.84      0.91      0.87      1546
           1       0.67      0.51      0.58       567

    accuracy                           0.80      2113
   macro avg       0.75      0.71      0.73      2113
weighted avg       0.79      0.80      0.79      2113
```

```
# Running SVC model
svc=SVC()
svc.fit(x_train,y_train)
pred=svc.predict(x_test)
svc_accu=accuracy_score(y_test,pred)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.791292001893043
[[1432  114]
 [ 327  240]]
              precision    recall  f1-score   support

           0       0.81      0.93      0.87      1546
           1       0.68      0.42      0.52       567

    accuracy                           0.79      2113
   macro avg       0.75      0.67      0.69      2113
weighted avg       0.78      0.79      0.77      2113
```

```
# Running KNeighborsClassifier model
knn=KNeighborsClassifier()
knn.fit(x_train,y_train)
knn_pred=knn.predict(x_test)
knn_accu=accuracy_score(y_test,pred)

print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.791292001893043
[[1432  114]
 [ 327  240]]
              precision    recall  f1-score   support

           0       0.81      0.93      0.87      1546
           1       0.68      0.42      0.52       567

    accuracy                           0.79      2113
   macro avg       0.75      0.67      0.69      2113
weighted avg       0.78      0.79      0.77      2113
```

```
# Running RandomForestClassifier model
rf=RandomForestClassifier()
rf.fit(x_train,y_train)
pred=rf.predict(x_test)
rf_accu=accuracy_score(y_test,pred)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.7893989588263133
[[1404  142]
 [ 303  264]]
              precision    recall  f1-score   support

           0       0.82      0.91      0.86      1546
           1       0.65      0.47      0.54       567

    accuracy                           0.79      2113
   macro avg       0.74      0.69      0.70      2113
weighted avg       0.78      0.79      0.78      2113
```

```
# Running AdaBoostClassifier model
ad=AdaBoostClassifier()
ad.fit(x_train,y_train)
pred=ad.predict(x_test)
ad_accu=accuracy_score(y_test,pred)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.7950780880265026
[[1395  151]
 [ 282  285]]
              precision    recall  f1-score   support

           0       0.83      0.90      0.87      1546
           1       0.65      0.50      0.57       567

    accuracy                           0.80      2113
   macro avg       0.74      0.70      0.72      2113
weighted avg       0.78      0.80      0.79      2113
```

```
# Running DecisionTreeClassifier model
dt=DecisionTreeClassifier()
dt.fit(x_train,y_train)
pred=dt.predict(x_test)
dt_accu=accuracy_score(y_test,pred)

print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.722669190724089
[[1249  297]
 [ 289  278]]
              precision    recall  f1-score   support

           0       0.81      0.81      0.81      1546
           1       0.48      0.49      0.49       567

    accuracy                           0.72      2113
   macro avg       0.65      0.65      0.65      2113
weighted avg       0.72      0.72      0.72      2113
```

```
: # Running GradientBoostingClassifier model
  gbc=GradientBoostingClassifier()
  gbc.fit(x_train,y_train)
  pred=gbc.predict(x_test)
  gbc_accu=accuracy_score(y_test,pred)

  print(accuracy_score(y_test,pred))
  print(confusion_matrix(y_test,pred))
  print(classification_report(y_test,pred))
```

```
0.7960246095598675
[[1406  140]
 [ 291  276]]
              precision    recall  f1-score   support

           0       0.83      0.91      0.87      1546
           1       0.66      0.49      0.56       567

    accuracy                           0.80      2113
   macro avg       0.75      0.70      0.71      2113
weighted avg       0.78      0.80      0.79      2113
```

- Conclusion:

The importance of these type of research in the telecom market is to help companies make more profit.

It has become know that predicting churn is one of the most important sources of income to telecom company.

Hence, these research aim to build a system that predicts the churn of customer telecom companies.

These prediction models need to achieve high AUC values. To test and train the model, the sample data is divided into 70% for training and 30 % testing.

Gradient Boosting Classifier is giving Maximum Accuracy as compare to other classification algorithm. Hence Gradient Boosting Classifier is the best model of this dataset.

```
conclusion
```

	0	1	2	3	4	5	6	7	8	9	...	2103	2104	2105	2106	2107	2108	2109	2110	2111	2112
predicted	0	1	0	1	1	0	1	0	0	0	...	0	0	0	0	0	1	0	1	0	0
original	0	1	0	0	1	0	1	0	0	0	...	0	0	0	0	0	0	0	1	1	1

2 rows × 2113 columns