CHURN PREDICTION FOR THE TELECOMMUNICATION SECTOR

Business Understanding

According to Wikipidea the telecommunications industries within the sector of information and communication technology is made up of all telecommunications/telephone companies and internet service providers and plays a crucial role in the evolution of mobile communications and the information society.

Traditional telephone calls continue to be the industry's biggest revenue generator, but thanks to advances in network technology, telecom today is less about voice and increasingly about text (messaging, email) and images (e.g. video streaming). High-speed internet access for computer-based data applications such as broadband information services and interactive entertainment is pervasive. Digital subscriber line (DSL) is the main broadband telecom technology. The fastest growth comes from (value-added) services delivered over mobile networks.

Business Problem

The telecommunication industry is quite capital intensive and requires recruitment and retention of a wide customer base in order to spread overheads. The industry guidance is that the cost of acquiring a new customer is more than the cost of retaining an existing customer. Consequently, churn prediction ,which is detecting which customers are likely to leave , is imperative because the company can then focus on retention of existing customers who are likely to leave.

Objective

Develop a classification model for churn prediction in order to guide customer retention strategies.

Data understanding

Our data is a CSV file of 3,333 records and 20 features. The features comprise static data of state ,area code phone number. The other features a dynamic data on number of calls, minutes and charges recorded during the daytime, evening, night and international calls.

Data Preparation

importation of the necessary libraries.

In [171]:
```
# import necessary libraries.
import pandas as pd
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import OneHotEncoder
from sklearn import tree
from sklearn.metrics import roc_curve,auc
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('darkgrid')
%matplotlib inline
```

Upload and display of the data.

In [172]:
```
df = pd.read_csv('bigml_59c28831336c6604c800002a.csv')
df.head()
```

Out[172]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...

5 rows × 21 columns

In [173]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   state                   3333 non-null   object 
 1   account length          3333 non-null   int64  
 2   area code               3333 non-null   int64  
 3   phone number            3333 non-null   object 
 4   international plan      3333 non-null   object 
 5   voice mail plan         3333 non-null   object 
 6   number vmail messages   3333 non-null   int64  
 7   total day minutes       3333 non-null   float64
 8   total day calls         3333 non-null   int64  
 9   total day charge        3333 non-null   float64
 10  total eve minutes       3333 non-null   float64
 11  total eve calls         3333 non-null   int64  
 12  total eve charge        3333 non-null   float64
 13  total night minutes     3333 non-null   float64
 14  total night calls       3333 non-null   int64  
 15  total night charge      3333 non-null   float64
 16  total intl minutes      3333 non-null   float64
 17  total intl calls        3333 non-null   int64  
 18  total intl charge       3333 non-null   float64
 19  customer service calls  3333 non-null   int64  
 20  churn                   3333 non-null   bool   
dtypes: bool(1), float64(8), int64(8), object(4)
memory usage: 524.2+ KB
```

Deteriming the class proportions of the target variable.

In [174]:
```
df['churn'].value_counts()
```

Out[174]:
```
False    2850
True      483
Name: churn, dtype: int64
```

In order to make the calls and minutes features more interpretable we passing in a minutes per call column known as call duration.

In [175]:
```
# Average minutes per call.

def minutes_per_call (calls, minutes):
    average_duration=minutes/calls
    return average_duration
```

In [176]:
```
df['day call duration']=minutes_per_call(df['total day calls'],df['total day 
df['eve call duration']=minutes_per_call(df['total eve calls'],df['total eve 
df['night call duration']=minutes_per_call(df['total night calls'],df['total 
df['intl call duration']=minutes_per_call(df['total intl calls'],df['total in
```

In [177]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 25 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   state                   3333 non-null   object 
 1   account length          3333 non-null   int64  
 2   area code               3333 non-null   int64  
 3   phone number            3333 non-null   object 
 4   international plan      3333 non-null   object 
 5   voice mail plan         3333 non-null   object 
 6   number vmail messages   3333 non-null   int64  
 7   total day minutes       3333 non-null   float64
 8   total day calls         3333 non-null   int64  
 9   total day charge        3333 non-null   float64
 10  total eve minutes       3333 non-null   float64
 11  total eve calls         3333 non-null   int64  
 12  total eve charge        3333 non-null   float64
 13  total night minutes     3333 non-null   float64
 14  total night calls       3333 non-null   int64  
 15  total night charge      3333 non-null   float64
 16  total intl minutes      3333 non-null   float64
 17  total intl calls        3333 non-null   int64  
 18  total intl charge       3333 non-null   float64
 19  customer service calls  3333 non-null   int64  
 20  churn                   3333 non-null   bool   
 21  day call duration       3331 non-null   float64
 22  eve call duration       3332 non-null   float64
 23  night call duration     3333 non-null   float64
 24  intl call duration      3315 non-null   float64
dtypes: bool(1), float64(12), int64(8), object(4)
memory usage: 628.3+ KB
```

Definition of the dependent and independent variables.

In [178]:
```
X=df[['international plan','day call duration','eve call duration','night cal
y=df[['churn']]
```

In [179]:
```
X
```

Out[179]:

	international plan	day call duration	eve call duration	night call duration	intl call duration
0	no	2.410000	1.993939	2.689011	3.333333
1	no	1.313821	1.898058	2.469903	4.566667
2	no	2.135088	1.101818	1.563462	2.440000
3	yes	4.216901	0.703409	2.212360	0.942857
4	yes	1.475221	1.215574	1.544628	3.366667
...	...	...	...	...	...
3328	no	2.028571	1.710317	3.362651	1.650000
3329	no	4.054386	2.789091	1.555285	2.400000
3330	no	1.658716	4.979310	2.108791	2.350000
3331	yes	2.036190	1.900000	1.016058	0.500000
3332	no	2.074336	3.242683	3.135065	3.425000

3333 rows × 5 columns

In [180]:
```
y
```

Out[180]:

	churn
0	False
1	False
2	False
3	False
4	False
...	...
3328	False
3329	False
3330	False
3331	False
3332	False

3333 rows × 1 columns

Splitting of the features into train and test data sets.

In [181]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=0.3,random_stat
```

Logistic Regression Train Data set

Data cleaning and transforming of the categorical features for the train data set.

In [182]:
```
X_train.isna().sum()
```

Out[182]:
```
international plan      0
day call duration       0
eve call duration       1
night call duration     0
intl call duration     13
dtype: int64
```

In [183]:
```
X_train.dtypes
```

Out[183]:
```
international plan      object
day call duration      float64
eve call duration      float64
night call duration    float64
intl call duration     float64
dtype: object
```

In [184]:
```
X_train_fill_na= X_train.copy()
X_train_fill_na.fillna({'eve call duration':0,'intl call duration':0},inplace
X_train_fill_na.isna().sum()
```

Out[184]:
```
international plan     0
day call duration      0
eve call duration      0
night call duration    0
intl call duration     0
dtype: int64
```

In [185]:
```
categorical=['international plan']
X_train_categorical= X_train_fill_na[categorical].copy()
X_train_categorical
```

Out[185]:

	international plan
2016	no
1362	no
2670	no
2210	no
1846	no
...	...
1095	no
1130	no
1294	no
860	no
3174	no

2333 rows × 1 columns

In [186]:
```
ohe=OneHotEncoder(handle_unknown='ignore',sparse=False)
ohe.fit(X_train_categorical)
X_train_ohe=pd.DataFrame(ohe.transform(X_train_categorical),index=X_train_cat
X_train_ohe
```

Out[186]:

	0	1
2016	1.0	0.0
1362	1.0	0.0
2670	1.0	0.0
2210	1.0	0.0
1846	1.0	0.0
...	...	...
1095	1.0	0.0
1130	1.0	0.0
1294	1.0	0.0
860	1.0	0.0
3174	1.0	0.0

2333 rows × 2 columns

In [187]:
```
X_train_noncategorical = X_train_fill_na.select_dtypes(exclude=['object']).co
X_train_noncategorical
```

Out[187]:

	day call duration	eve call duration	night call duration	intl call duration
2016	1.715254	3.883582	1.583929	1.840000
1362	1.089344	1.080645	1.325620	4.950000
2670	2.046296	1.279661	2.237500	1.500000
2210	2.535455	2.838806	3.038095	1.671429
1846	1.520513	2.501429	1.378632	2.875000
...	...	...	...	...
1095	2.286667	2.421951	2.593548	2.000000
1130	0.566129	2.031461	4.337931	6.350000
1294	1.152632	2.360360	1.476800	1.840000
860	1.614414	1.347692	2.484783	1.650000
3174	0.243089	1.103419	3.103810	1.433333

2333 rows × 4 columns

In [188]:
```
X_train_full=pd.concat([X_train_ohe,X_train_noncategorical],axis=1)
X_train_full
```

Out[188]:

	0	1	day call duration	eve call duration	night call duration	intl call duration
2016	1.0	0.0	1.715254	3.883582	1.583929	1.840000
1362	1.0	0.0	1.089344	1.080645	1.325620	4.950000
2670	1.0	0.0	2.046296	1.279661	2.237500	1.500000
2210	1.0	0.0	2.535455	2.838806	3.038095	1.671429
1846	1.0	0.0	1.520513	2.501429	1.378632	2.875000
...	...	...	...	...	...	...
1095	1.0	0.0	2.286667	2.421951	2.593548	2.000000
1130	1.0	0.0	0.566129	2.031461	4.337931	6.350000
1294	1.0	0.0	1.152632	2.360360	1.476800	1.840000
860	1.0	0.0	1.614414	1.347692	2.484783	1.650000
3174	1.0	0.0	0.243089	1.103419	3.103810	1.433333

2333 rows × 6 columns

In [189]:
```
y_train
```

Out[189]:

	churn
2016	False
1362	False
2670	False
2210	True
1846	False
...	...
1095	False
1130	False
1294	False
860	False
3174	False

2333 rows × 1 columns

In [190]:
```
from sklearn import preprocessing
lb=preprocessing.LabelBinarizer()
lb.fit(y_train)
y_train_lb=pd.DataFrame(lb.transform(y_train))
y_train_lb
```

Out[190]:

	0
0	0
1	0
2	0
3	1
4	0
...	...
2328	0
2329	0
2330	0
2331	0
2332	0

2333 rows × 1 columns

Logistic Regression model initiation and fitting for the train dataset.

In [191]:
```
from sklearn.linear_model import LogisticRegression
logreg= LogisticRegression(fit_intercept=False,C=1e12,solver='liblinear')
model_log= logreg.fit(X_train_full,y_train_lb)
model_log
```

```
C:\Users\asaav\anaconda3\envs\learn-env\lib\site-packages\sklearn\utils\vali
dation.py:72: DataConversionWarning: A column-vector y was passed when a 1d
array was expected. Please change the shape of y to (n_samples, ), for examp
le using ravel().
  return f(**kwargs)
```

Out[191]:
```
LogisticRegression(C=1000000000000.0, fit_intercept=False, solver='liblinea
r')
```

In [192]:
```
y_train_lb_hat = logreg.predict(X_train_full).reshape(2333,1)

train_residuals = np.abs(y_train_lb - y_train_lb_hat)
train_residuals.value_counts()
```

Out[192]:
```
0    2011
1     322
dtype: int64
```

In [193]:
```
train_residuals.value_counts(normalize=True)
```

Out[193]:
```
0    0.86198
1    0.13802
dtype: float64
```

In [194]:
```
accuracy_score(y_train_lb,y_train_lb_hat)
```

Out[194]:
```
0.8619802828975568
```

The logistic model train data accuracy score is 0.86.This is a good score, however, given the classification of the target variable the model may be affected by class imbalance.

Logistic Regression Test Data set

Data cleaning and transformation of categorical variable for the test data set.

In [195]:
```
X_test_fill_na= X_test.copy()
X_test_fill_na.fillna({'eve call duration':0,'intl call duration':0,'day call
X_test_fill_na.isna().sum()
```

Out[195]:
```
international plan     0
day call duration      0
eve call duration      0
night call duration    0
intl call duration     0
dtype: int64
```

In [196]:
```
categorical=['international plan']
X_test_categorical= X_test_fill_na[categorical].copy()
X_test_categorical
```

Out[196]:

	international plan
438	no
2674	no
1345	no
1957	no
2148	no
...	...
3080	no
2548	no
2916	no
2655	no
1159	no

1000 rows × 1 columns

In [197]:

```
X_test_ohe=pd.DataFrame(ohe.transform(X_test_categorical),index=X_test_catego
X_test_ohe
```

Out[197]:

	0	1
438	1.0	0.0
2674	1.0	0.0
1345	1.0	0.0
1957	1.0	0.0
2148	1.0	0.0
...	...	...
3080	1.0	0.0
2548	1.0	0.0
2916	1.0	0.0
2655	1.0	0.0
1159	1.0	0.0

1000 rows × 2 columns

In [198]:

```
X_test_noncategorical = X_test_fill_na.select_dtypes(exclude=['object']).copy
X_test_noncategorical
```

Out[198]:

	day call duration	eve call duration	night call duration	intl call duration
438	1.666667	3.118868	1.539837	4.500000
2674	0.932479	1.753226	1.336170	2.133333
1345	0.000000	1.227692	1.898864	6.800000
1957	2.693671	2.242857	1.382301	5.100000
2148	1.411765	3.078082	2.502198	1.428571
...	...	...	...	...
3080	1.327451	1.943443	1.300000	4.375000
2548	1.367857	2.393636	1.588235	1.700000
2916	1.305747	1.618367	2.157471	1.750000
2655	3.002740	2.569231	1.356303	2.000000
1159	0.846721	1.999187	1.695789	0.914286

1000 rows × 4 columns

In [199]:
```
X_test_full=pd.concat([X_test_ohe,X_test_noncategorical],axis=1)
X_test_full
```

Out[199]:

	0	1	day call duration	eve call duration	night call duration	intl call duration
438	1.0	0.0	1.666667	3.118868	1.539837	4.500000
2674	1.0	0.0	0.932479	1.753226	1.336170	2.133333
1345	1.0	0.0	0.000000	1.227692	1.898864	6.800000
1957	1.0	0.0	2.693671	2.242857	1.382301	5.100000
2148	1.0	0.0	1.411765	3.078082	2.502198	1.428571
...	...	...	...	...	...	...
3080	1.0	0.0	1.327451	1.943443	1.300000	4.375000
2548	1.0	0.0	1.367857	2.393636	1.588235	1.700000
2916	1.0	0.0	1.305747	1.618367	2.157471	1.750000
2655	1.0	0.0	3.002740	2.569231	1.356303	2.000000
1159	1.0	0.0	0.846721	1.999187	1.695789	0.914286

1000 rows × 6 columns

In [200]:
```
y_test
```

Out[200]:

	churn
438	False
2674	False
1345	True
1957	False
2148	False
...	...
3080	False
2548	False
2916	False
2655	False
1159	False

1000 rows × 1 columns

In [201]:
```
y_test_lb=pd.DataFrame(lb.transform(y_test))
y_test_lb
```

Out[201]:

	0
0	0
1	0
2	1
3	0
4	0
...	...
995	0
996	0
997	0
998	0
999	0

1000 rows × 1 columns

Logistic regression modelling of the test data set.

In [202]:
```
y_test_lb_hat = model_log.predict(X_test_full).reshape(1000,1)

test_residuals = np.abs(y_test_lb - y_test_lb_hat)
test_residuals.value_counts()
```

Out[202]:
```
0    862
1    138
dtype: int64
```

In [203]:
```
test_residuals.value_counts(normalize=True)
```

Out[203]:
```
0    0.862
1    0.138
dtype: float64
```

In [204]:
```
accuracy_score(y_test_lb,y_test_lb_hat)
```

Out[204]:
```
0.862
```

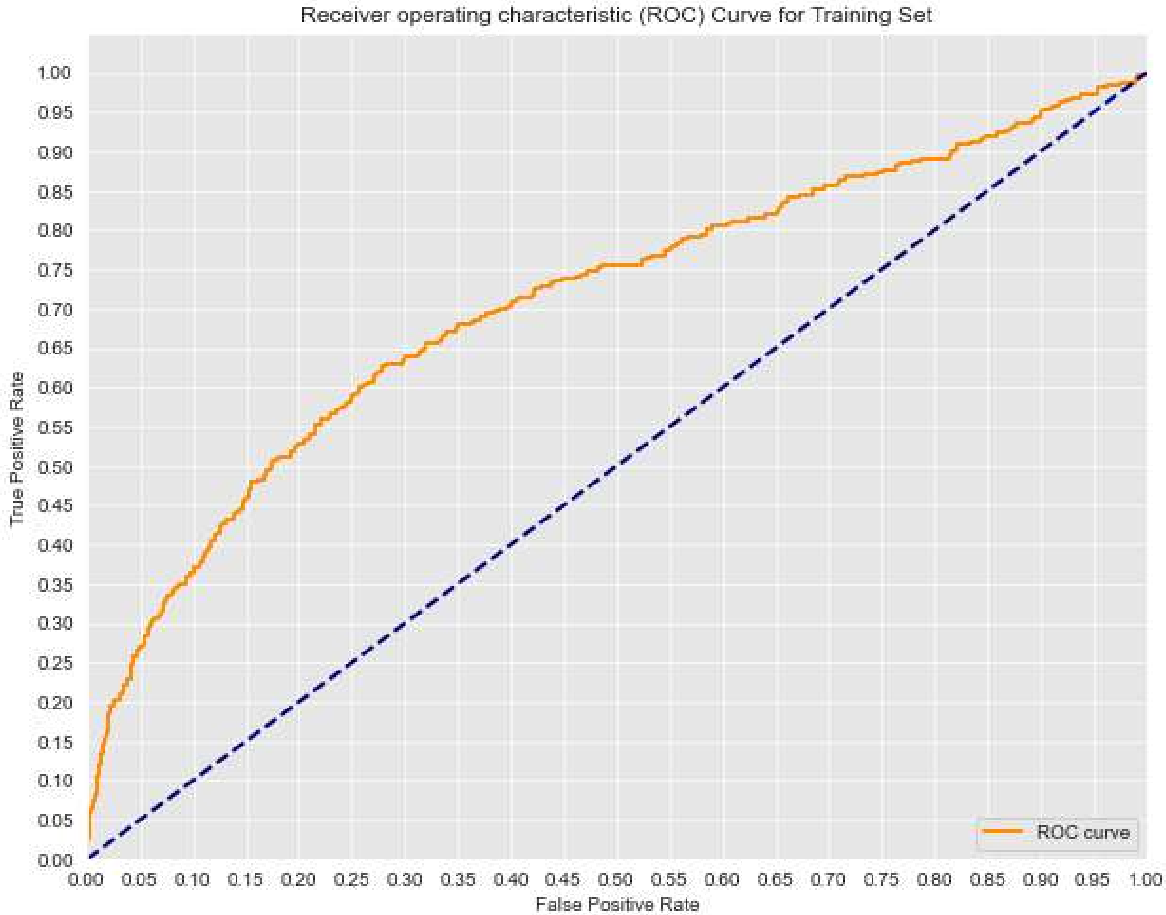
The accuracy score for the test data set almost matches the train data set.

ROC curve and AUC score for Logistic regression Train data set

In [205]:

```
y_train_score_1 = model_log.decision_function(X_train_full)


train_fpr_1, train_tpr_1, thresholds_1 = roc_curve(y_train_lb, y_train_score_


y_test_score_1 = model_log.decision_function(X_test_full)


test_fpr_1, test_tpr_1, test_thresholds_1 = roc_curve(y_test_lb, y_test_score

sns.set_style('darkgrid', {'axes.facecolor': '0.9'})


plt.figure(figsize=(10, 8))
lw = 2
plt.plot(train_fpr_1, train_tpr_1, color='darkorange',
         lw=lw, label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC) Curve for Training Set')
plt.legend(loc='lower right')
print('Training AUC: {}'.format(auc(train_fpr_1, train_tpr_1)))
plt.show()
```

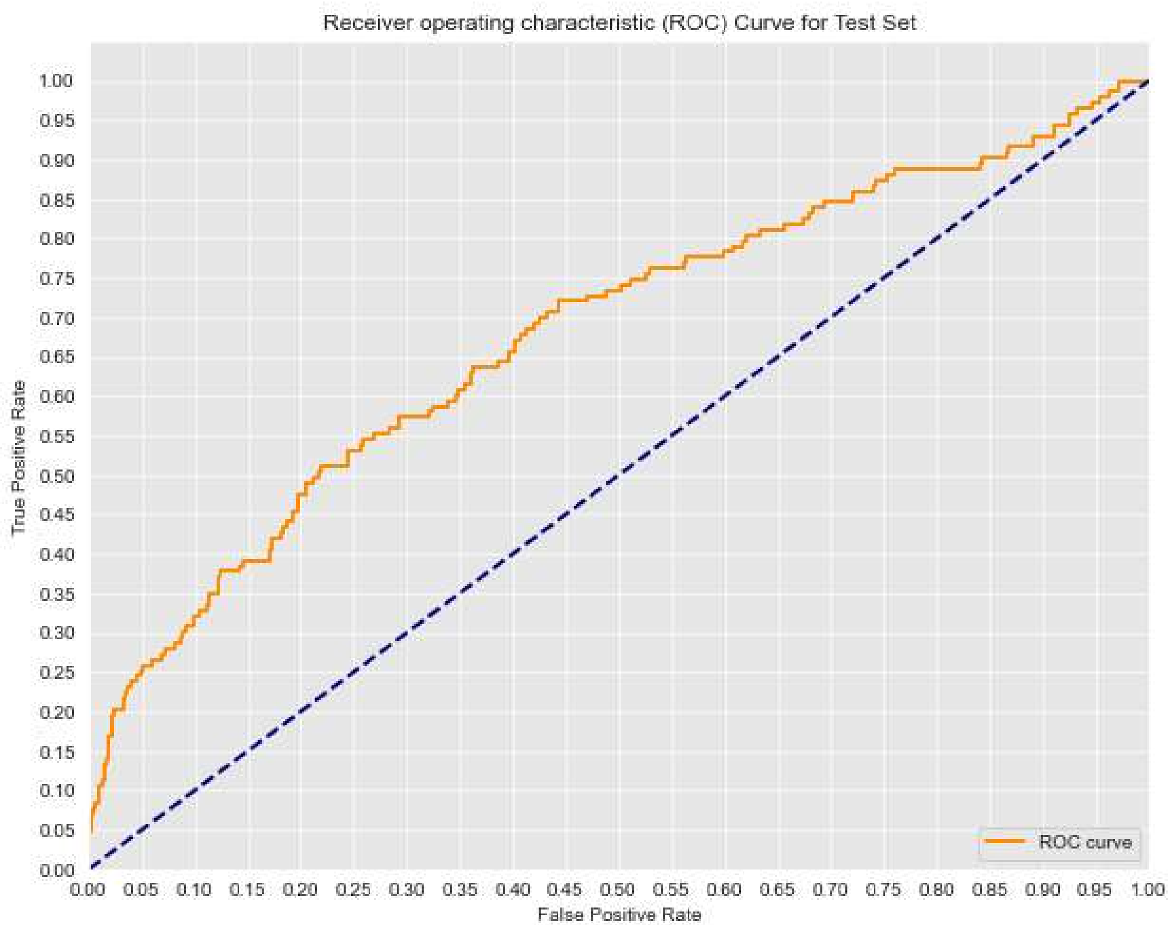
```
Training AUC: 0.7073344942593194
```



ROC Curve and AUC score for the Logistic Regression test data set.

In [206]:
```
plt.figure(figsize=(10, 8))
lw = 2
plt.plot(test_fpr_1, test_tpr_1, color='darkorange',
         lw=lw, label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC) Curve for Test Set')
plt.legend(loc='lower right')
print('Test AUC: {}'.format(auc(test_fpr_1, test_tpr_1)))
print('')
plt.show()
```

```
Test AUC: 0.6784603960799994
```



Decision Tree Initial Model

The Decision Tree is supervised learning technique mostly prereffred for classification.We will now employ the classification method and consider the results.

In [207]:
```
# Initiating the model.
clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(X_train_full,y_train_lb)
```

Out[207]:
```
DecisionTreeClassifier(criterion='entropy')
```

In [208]:
```
# Decision Tree with the train data.
y_train_lb_hat_dt=clf.predict(X_train_full)
```

In [209]:
```
accuracy_score(y_train_lb,y_train_lb_hat_dt)
```

Out[209]:
```
1.0
```

In [210]:
```
# Decision Trees with the Test data.
y_test_lb_hat_dt=clf.predict(X_test_full)
```

In [211]:
```
accuracy_score(y_test_lb_hat_dt,y_test_lb)
```

Out[211]:
```
0.794
```

The train accuracy score is 1 and the test score is 0.78 .These suggests overfitting because the model has performed really well on the train data but a bit poorly on the test data.

Action: We would need to optimize the decision tree to determine if it would predict suitably.

Addressing class imbalance under logistic regression.

A model is susceptible to class imbalance when a greater proportion of the target variable is of a certain class. The model's predictions most likely have a high accuracy score because mots of the predictions would be towards the dominant class. In our scenario our target variable churn comprises 85% false and 15% true. Consequently, the logistic regression model is affected by class imbalance.We will address this by introducing class weights.

In [212]:
```
# Introduction of class weights into the logistic regression model.
logreg= LogisticRegression(fit_intercept=False,C=1e12,class_weight='balanced'
model_log_bal= logreg.fit(X_train_full,y_train_lb)
model_log_bal
```

```
C:\Users\asaav\anaconda3\envs\learn-env\lib\site-packages\sklearn\utils\vali
dation.py:72: DataConversionWarning: A column-vector y was passed when a 1d
array was expected. Please change the shape of y to (n_samples, ), for examp
le using ravel().
  return f(**kwargs)
```

Out[212]:
```
LogisticRegression(C=1000000000000.0, class_weight='balanced',
                   fit_intercept=False, solver='liblinear')
```

In [213]:
```
y_train_lb_hat_bal = model_log_bal.predict(X_train_full).reshape(2333,1)

train_residuals_bal = np.abs(y_train_lb - y_train_lb_hat_bal)
train_residuals_bal.value_counts()
```

Out[213]:
```
0    1758
1     575
dtype: int64
```

In [214]:
```
train_residuals_bal.value_counts(normalize=True)
```

Out[214]:
```
0    0.753536
1    0.246464
dtype: float64
```

In [215]:
```
accuracy_score(y_train_lb,y_train_lb_hat_bal)
```

Out[215]:
```
0.7535362194599229
```

In [216]:
```
# Class weighted Logistic Regression model applied on the test data.
y_test_lb_hat_bal = model_log_bal.predict(X_test_full).reshape(1000,1)
```

In [217]:
```
accuracy_score(y_test_lb,y_test_lb_hat_bal)
```

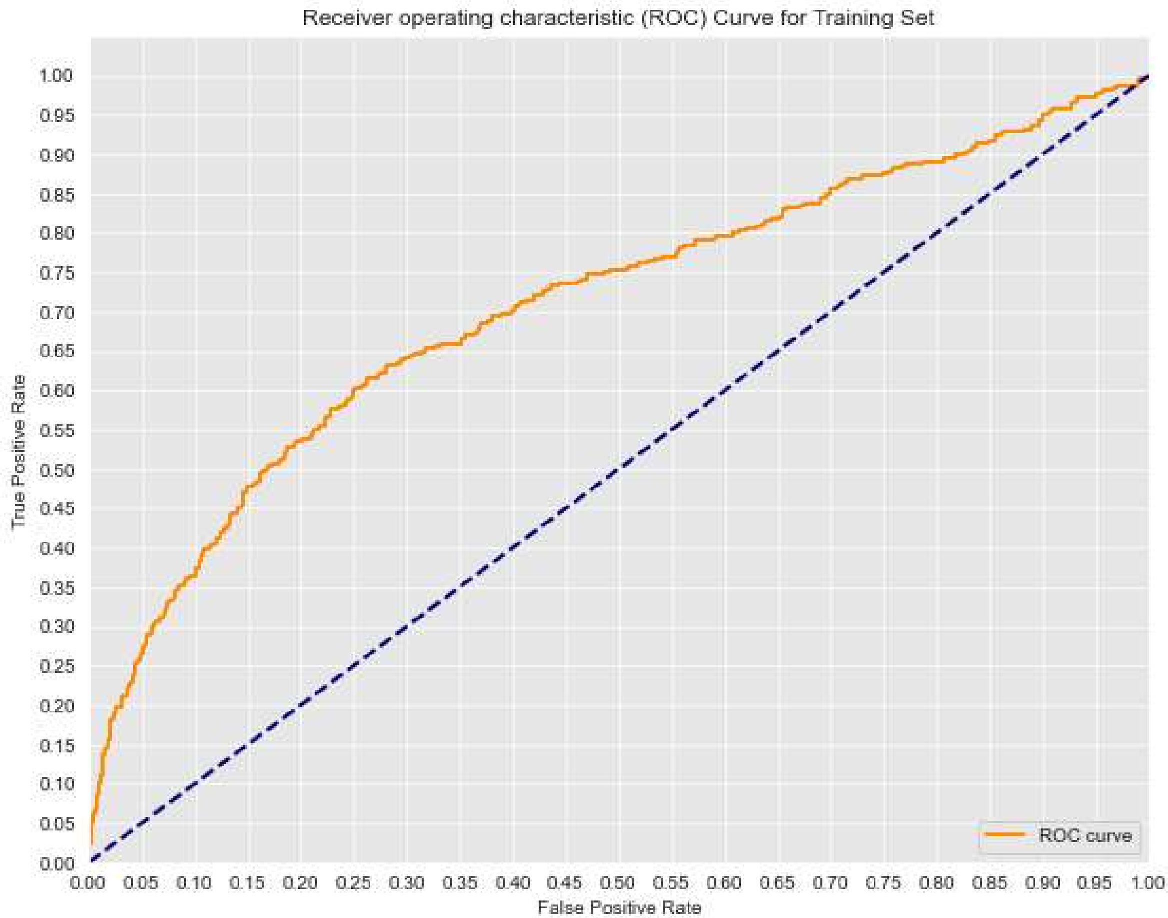
Out[217]:
```
0.738
```

ROC curve and AUC score for class weighted logistic regression model for train data set.

In [218]:

```
y_train_score = model_log_bal.decision_function(X_train_full)


train_fpr, train_tpr, thresholds = roc_curve(y_train_lb, y_train_score)


y_test_score = model_log_bal.decision_function(X_test_full)


test_fpr, test_tpr, test_thresholds = roc_curve(y_test_lb, y_test_score)

sns.set_style('darkgrid', {'axes.facecolor': '0.9'})


plt.figure(figsize=(10, 8))
lw = 2
plt.plot(train_fpr, train_tpr, color='darkorange',
         lw=lw, label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC) Curve for Training Set')
plt.legend(loc='lower right')
print('Training AUC: {}'.format(auc(train_fpr, train_tpr)))
plt.show()
```

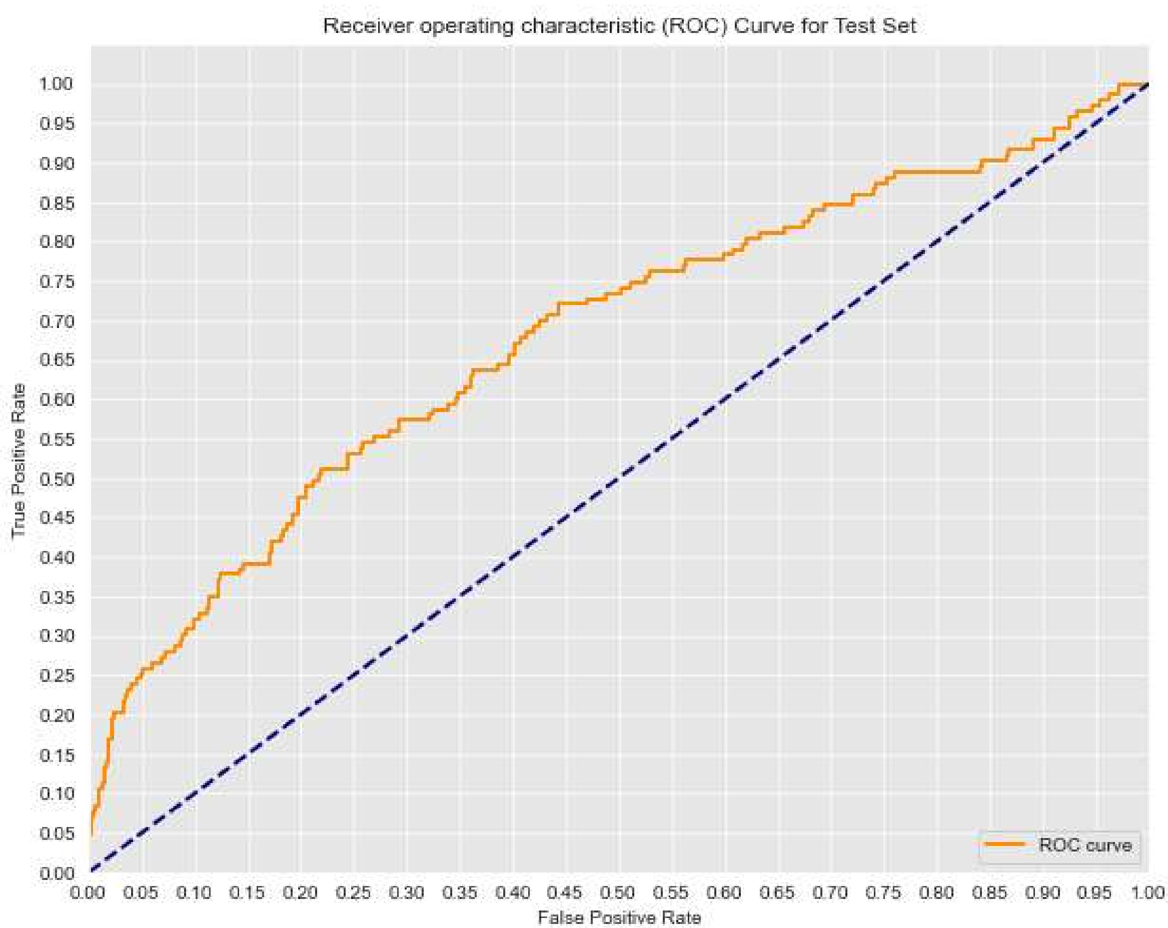
```
Training AUC: 0.7069552256426905
```



ROC curve and AUC score for class weighted logistic regression model for test data set.

In [219]:
```
plt.figure(figsize=(10, 8))
lw = 2
plt.plot(test_fpr, test_tpr, color='darkorange',
         lw=lw, label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC) Curve for Test Set')
plt.legend(loc='lower right')
print('Test AUC: {}'.format(auc(test_fpr, test_tpr)))
print('')
plt.show()
```

```
Test AUC: 0.6784603960799994
```



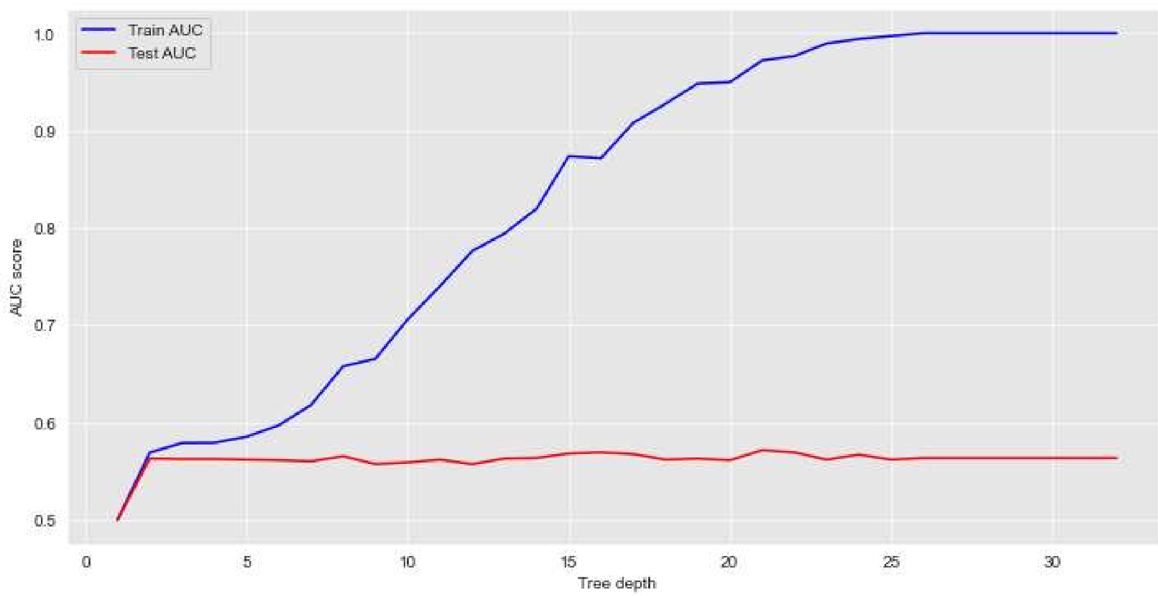
The initial train and test logistic regressions has an almost identical accuracy score of 0.86 and AUC train and test score of 0.707 and 0.678 respecitively. After class weighting to mitigate class imbalance the train and test accuracy scores reduced to 0.75 and 0.73 respectively.The train and test AUC scores , however, changed slightly to 0.706 and 0.678. Based on the data we are unable to conclude on whether the initial model or weighted model is more suitable because the accuracy scores have reduced but the AUC scores are unchanged.

Action: The performance of the logistic model has not improved hence we cannot adopt it for prediction.

Decision Tree Model Pruning.

Decision trees have a various advantages such as interpretability,ability to handle unbalanced data,variable selection among others. One of its main drawback is susceptability to overfitting.This evident under our training model where the accuracy score is 1 versus a test score of 0.78.There are various strategies to reduce overfitting such Maximum tree depth which has been used to prune the decision tree training model.

In [220]:
```
# Determining the maximum tree depth through use of graphical presentation of
from sklearn.metrics import roc_curve,auc

max_depths= list(range(1,33))
train_results_roc=[]
test_results_roc=[]
for max_depth in max_depths:
    dt= DecisionTreeClassifier(criterion='entropy',max_depth=max_depth,random_
    dt.fit(X_train_full,y_train_lb)
    train_pred=dt.predict(X_train_full)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train_l
    roc_auc = auc(false_positive_rate, true_positive_rate)
    train_results_roc.append(roc_auc)
    y_pred = dt.predict(X_test_full)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test_lb
    roc_auc = auc(false_positive_rate, true_positive_rate)
    test_results_roc.append(roc_auc)

plt.figure(figsize=(12,6))
plt.plot(max_depths, train_results_roc, 'b', label='Train AUC')
plt.plot(max_depths, test_results_roc, 'r', label='Test AUC')
plt.ylabel('AUC score')
plt.xlabel('Tree depth')
plt.legend()
plt.show()
```



The optimal tree depth is 2 because the test AUC does not increase beyond that.

In [221]:
```
# Decision tree train model pruning through use of maximum tree depth of 1.
clf2 = DecisionTreeClassifier(criterion='entropy',max_depth=3,random_state=1)
clf2.fit(X_train_full,y_train_lb)
```

Out[221]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=1)
```

In [222]:
```
y_train_lb_hat_dt_dp=clf2.predict(X_train_full)
```

In [223]:
```
accuracy_score(y_train_lb,y_train_lb_hat_dt_dp)
```

Out[223]:
```
0.876553793399057
```

After pruning our train accuracy score is more realistic at 0.876 versus an initial score of 1. Our model is not overfitting.

In [224]:
```
# Test data with the pruned Decision Tree model.
y_test_lb_hat_dt_dp=clf2.predict(X_test_full)
```

In [225]:
```
accuracy_score(y_test_lb_hat_dt_dp,y_test_lb)
```

Out[225]:
```
0.874
```

We now have a trained accuracy score of 0.876 and a test accuracy score of 0.874 . This shows the model is not overfitted nor is it underfitted.It is at an optimal level to make predictions.

Recommendations

1. The model to be considered to predict customer churn would be the Pruned decision tree which has a minimal difference between its train and test accuracy scores.
2. The pruned decision tree model's accuracy scores are higher than the logistic regression model.