SEP 721 – Data Analytics, Machine Learning and AI on Cloud Platforms

Assignment 1: Qwiklabs- 1 and 2

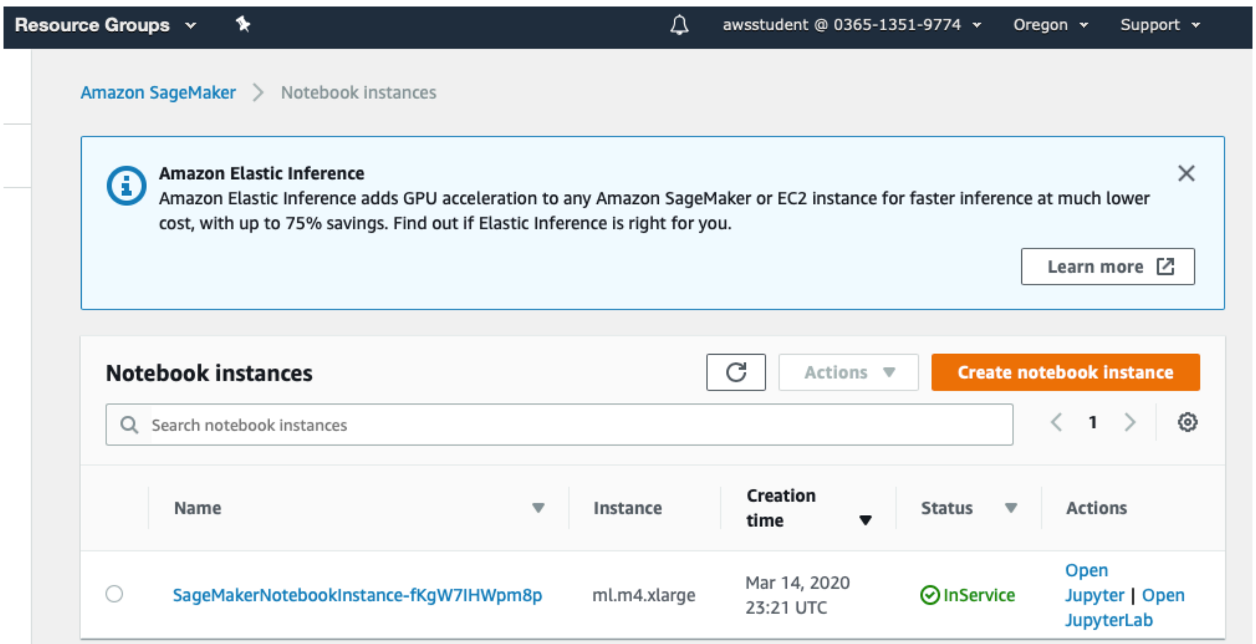
Submitted by,

Greeshma Gopal(gopalg)

ID- 400245291

Lab 1: Creating Models with Amazon SageMaker

- I have used qwiklab credits to access the jupyter file and the dataset.



- The dataset here is about mobile operator customers. The prediction we will be doing is the churning of the customers or in short finding the less satisfied customers and offering them better plans just so they do not switch to a different operator.

Background

Losing customers is costly for any business. Identifying unhappy customers early on gives you a chance to offer them incentives to stay. This notebook describes using machine learning (ML) for the automated identification of unhappy customers, also known as customer churn prediction. ML models rarely give perfect predictions though, so this notebook is also about how to incorporate the relative costs of prediction mistakes when determining the financial outcome of using ML.

We use an example of churn that is familiar to all of us–leaving a mobile phone operator. Seems like I can always find fault with my provider du jour! And if my provider knows that I'm thinking of leaving, it can offer timely incentives–I can always use a phone upgrade or perhaps have a new feature activated–and I might just stick around. Incentives are often much more cost effective than losing and reacquiring a customer.

Setup

This notebook was created and tested on an ml.m4.xlarge notebook instance.

Let's start by specifying:

- The S3 bucket and prefix that you want to use for training and model data. The lab account has created this for you. The name can be found on the left of the Qwiklabs console.
- Replace `<your_s3_bucket_name_here>` with the bucket name below.

- Qwiklab creates a bucket which would help us store the files

In [1]:
```
bucket = 'qls-12010048-f8254cebdf5dd1ab-labbucket-63n3xow7xm8c'
prefix = 'sagemaker/DEMO-xgboost-churn'

# Define IAM role
import boto3
import re
from sagemaker import get_execution_role

role = get_execution_role()
```

- The required libraries for modelling and training are imported initially.

In [2]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import io
import os
import sys
import time
import json
from IPython.display import display
from time import strftime, gmtime
import sagemaker
from sagemaker.predictor import csv_serializer
```

- In this step, we are unzipping the dataset file to extract the data

n [3]:

```
!unzip -o DKD2e_data_sets.zip
```

```
Archive:  DKD2e_data_sets.zip
 extracting: Data sets/adult.zip
  inflating: Data sets/cars.txt
  inflating: Data sets/cars2.txt
  inflating: Data sets/cereals.CSV
  inflating: Data sets/churn.txt
  inflating: Data sets/ClassifyRisk
  inflating: Data sets/ClassifyRisk - Missing.txt
 extracting: Data sets/DKD2e data sets.zip
  inflating: Data sets/nn1.txt
```

- Reading the dataset which is assigned as a data frame in the variable churn

In [4]:
```
churn = pd.read_csv('./Data sets/churn.txt')
pd.set_option('display.max_columns', 500)
churn
```

Out[4]:

	State	Account Length	Area Code	Phone	Int'l Plan	VMail Plan	VMail Message	Day Mins	Day Calls	Day Charge	Eve Mins	Eve Calls	Eve Charge	Night Mins	Night Calls	Night Charge	Intl Mins	Intl Calls	Intl Charge	CustServ Calls
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0	3	2.70	1
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7	3	3.70	1
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2	5	3.29	0
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6	7	1.78	2
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1	3	2.73	3
5	AL	118	510	391-8027	yes	no	0	223.4	98	37.98	220.6	101	18.75	203.9	118	9.18	6.3	6	1.70	0
6	MA	121	510	355-9993	no	yes	24	218.2	88	37.09	348.5	108	29.62	212.6	118	9.57	7.5	7	2.03	3

- Data visualization

In [5]:
```
# Frequency tables for each categorical feature
for column in churn.select_dtypes(include=['object']).columns:
    display(pd.crosstab(index=churn[column], columns='% observations', normalize='columns'))

# Histograms for each numeric features
display(churn.describe())
%matplotlib inline
hist = churn.hist(bins=30, sharey=True, figsize=(10, 10))
```

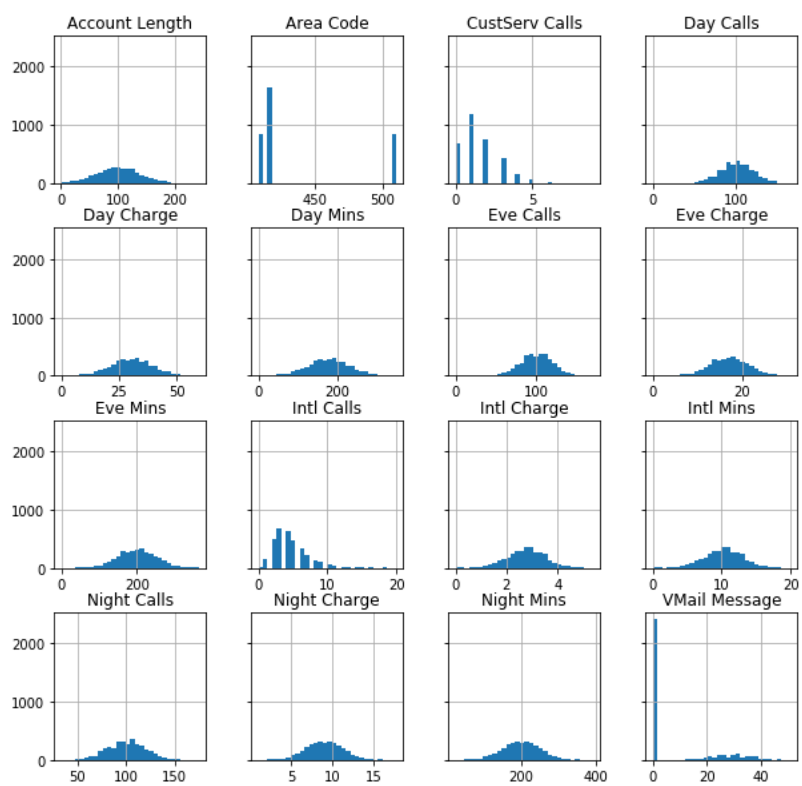
col_0	% observations
State	
AK	0.015602
AL	0.024002
AR	0.016502
AZ	0.019202
CA	0.010201
CO	0.019802
CT	0.022202
DC	0.016202
DE	0.018302
FL	0.018902

col_0	% observations
VMail Plan	
no	0.723372
yes	0.276628

col_0	% observations
Churn?	
False.	0.855086
True.	0.144914

	Account Length	Area Code	VMail Message	Day Mins	Day Calls	Day Charge	Eve Mins	Eve Calls	Eve Charge	Night Mins	Night Calls	
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.980348	100.114311	17.083540	200.872037	100.107711	9
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.713844	19.922625	4.310668	50.573847	19.568609	2
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	23.200000	33.000000	1
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.600000	87.000000	14.160000	167.000000	87.000000	7
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.400000	100.000000	17.120000	201.200000	100.000000	9
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.300000	114.000000	20.000000	235.300000	113.000000	10
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.700000	170.000000	30.910000	395.000000	175.000000	17

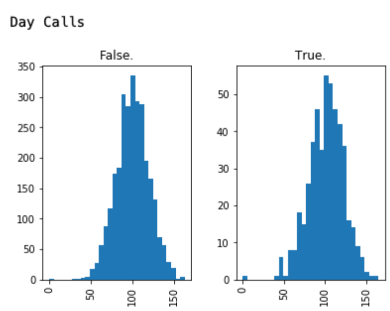
Account Length Area Code CustServ Calls Day Calls

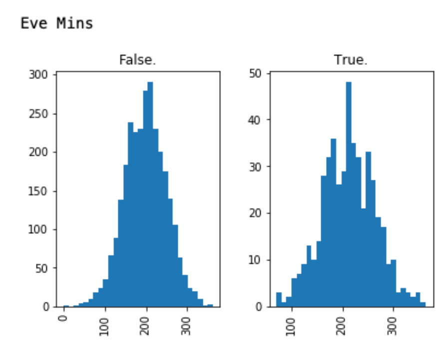


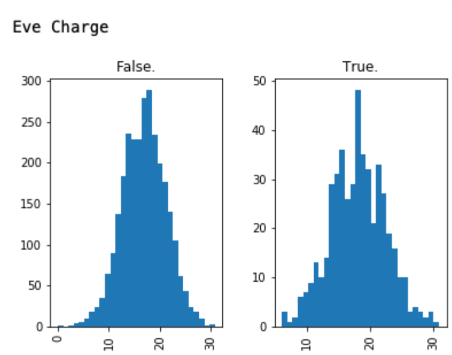
- Removing the target variable from the data frame churn

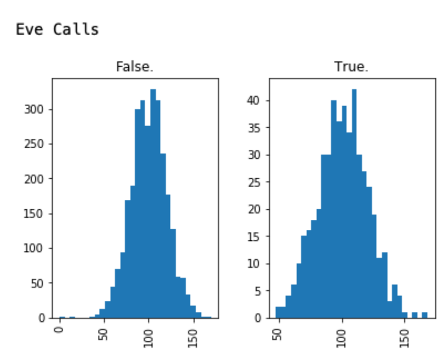
```
[6]: churn = churn.drop('Phone', axis=1)
     churn['Area Code'] = churn['Area Code'].astype(object)
```

- Visualizing the relationship between each of the features and our target variable

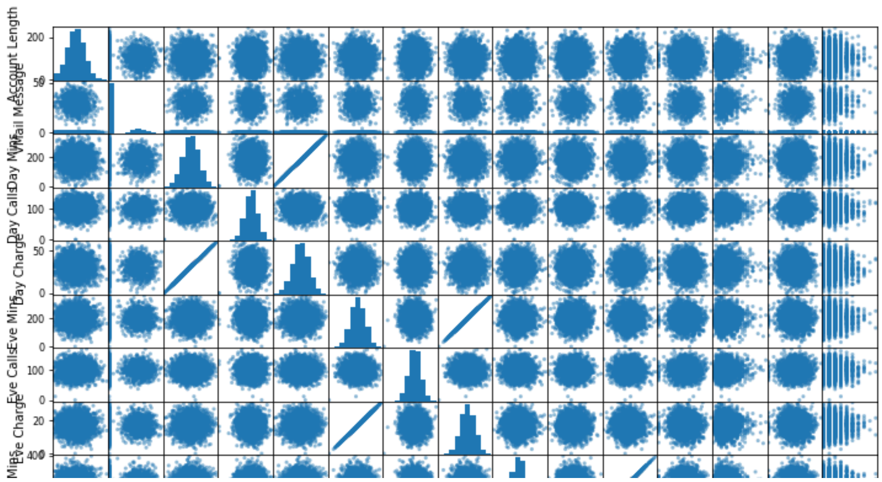








Night Charge	-0.008960	0.007663	0.004300	0.022927	0.004301	-0.012593	-0.002056	-0.012601	0.999999	0.011188	1.000000	-0.015214	-0.012329	-0.0151
Intl Mins	0.009514	0.002856	-0.010155	0.021565	-0.010157	-0.011035	0.008703	-0.011043	-0.015207	-0.013605	-0.015214	1.000000	0.032304	0.9999
Intl Calls	0.020661	0.013957	0.008033	0.004574	0.008032	0.002541	0.017434	0.002541	-0.012353	0.000305	-0.012329	0.032304	1.000000	0.0323
Intl Charge	0.009546	0.002884	-0.010092	0.021666	-0.010094	-0.011067	0.008674	-0.011074	-0.015180	-0.013630	-0.015186	0.999993	0.032372	1.0000
CustServ Calls	-0.003796	-0.013263	-0.013423	-0.018942	-0.013427	-0.012985	0.002423	-0.012987	-0.009288	-0.012802	-0.009277	-0.009640	-0.017561	-0.0096



- Converting categorical features into numeric features

```
In [8]: model_data = pd.get_dummies(churn)
        model_data = pd.concat([model_data['Churn?_True.'], model_data.drop(['Churn?_False.', 'Churn?_True.'], axis=1)], axi
```

- Splitting Train and Test data

```
n [9]: train_data, validation_data, test_data = np.split(model_data.sample(frac=1, random_state=1729), [int(0.7 * len(model
       train_data.to_csv('train.csv', header=False, index=False)
       validation_data.to_csv('validation.csv', header=False, index=False)
```

- Uploading these files to S3 bucket which qwiklabs has created by default or the lab

```
In [10]: boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'train/train.csv')).upload_file('train.csv
         boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'validation/validation.csv')).upload_file(
```

- Specifying the locations of the XGBoost algorithm containers

```
[11]: from sagemaker.amazon.amazon_estimator import get_image_uri
      container = get_image_uri(boto3.Session().region_name, 'xgboost', repo_version='0.90-1')
```

```
In [12]: s3_input_train = sagemaker.s3_input(s3_data='s3://{}/{}/train'.format(bucket, prefix), content_type='csv')
         s3_input_validation = sagemaker.s3_input(s3_data='s3://{}/{}/validation/'.format(bucket, prefix), content_type='csv'
```

- Defining hyperparameters such as max_depth, subsample, num_round, eta and gamma after which the training data model fitting is done.

```
                                    role,
                                    train_instance_count=1,
                                    train_instance_type='ml.m4.xlarge',
                                    output_path='s3://{}/{}/output'.format(bucket, prefix),
                                    sagemaker_session=sess)
xgb.set_hyperparameters(max_depth=5,
                        eta=0.2,
                        gamma=4,
                        min_child_weight=6,
                        subsample=0.8,
                        silent=0,
                        objective='binary:logistic',
                        num_round=100)

xgb.fit({'train': s3_input_train, 'validation': s3_input_validation})
2020-03-14 23:38:40 Starting - Starting the training job...
2020-03-14 23:38:41 Starting - Launching requested ML instances...
2020-03-14 23:39:38 Starting - Preparing the instances for training......
2020-03-14 23:40:37 Downloading - Downloading input data...
2020-03-14 23:40:57 Training - Downloading the training image...
2020-03-14 23:41:39 Uploading - Uploading generated training model
2020-03-14 23:41:39 Completed - Training job completed
INFO:sagemaker-containers:Imported framework sagemaker_xgboost_container.training
INFO:sagemaker-containers:Failed to parse hyperparameter objective value binary:logistic to Json.
Returning the value itself
INFO:sagemaker-containers:No GPUs detected (normal if no gpus installed)
INFO:sagemaker_xgboost_container.training:Running XGBoost Sagemaker in algorithm mode
INFO:root:Determined delimiter of CSV input is ','
INFO:root:Determined delimiter of CSV input is ','
INFO:root:Determined delimiter of CSV input is ','
[23:41:27] 2333x73 matrix with 170309 entries loaded from /opt/ml/input/data/train?format=csv&label_column=0&delimi
ter=,
INFO:root:Determined delimiter of CSV input is ','
[23:41:27] 666x73 matrix with 48618 entries loaded from /opt/ml/input/data/validation?format=csv&label_column=0&del
```

- Creating the model and deploying it on the end point

```
In [23]: xgb_predictor = xgb.deploy(initial_instance_count=1,
                                     instance_type='ml.m4.xlarge')

         ----------------!
```

- Setting up serializers and de-serializers for passing the data in the model behind the end point

```
In [24]: xgb_predictor.content_type = 'text/csv'
         xgb_predictor.serializer = csv_serializer
         xgb_predictor.deserializer = None
```

- Writing the function to loop the test data. The final predictions are converted into CSV format

```
ı [25]: def predict(data, rows=500):
            split_array = np.array_split(data, int(data.shape[0] / float(rows) + 1))
            predictions = ''
            for array in split_array:
                predictions = ','.join([predictions, xgb_predictor.predict(array).decode('utf-8')])

            return np.fromstring(predictions[1:], sep=',')

        predictions = predict(test_data.values[:, 1:])
```

- Predicting whether the customer churned 1 or 0, which is a confusion matrix

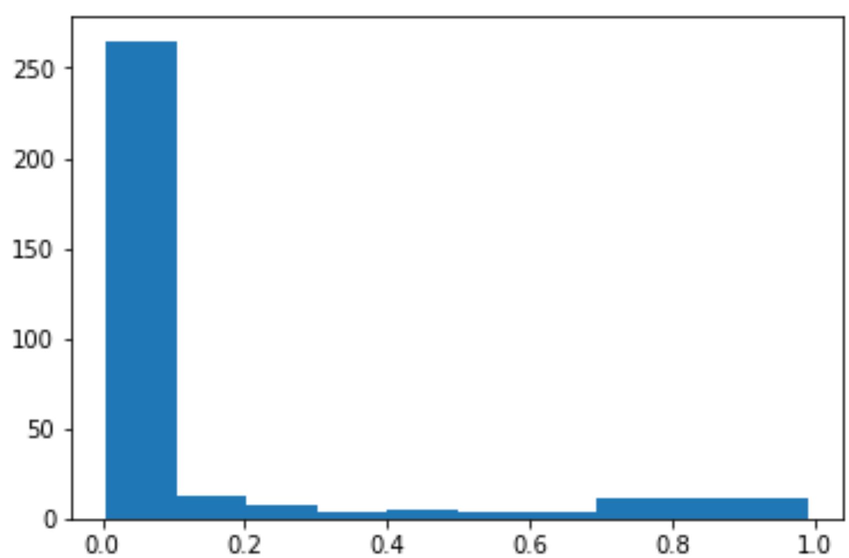
```
[26]: pd.crosstab(index=test_data.iloc[:, 0], columns=np.round(predictions), rownames=['actual'], colnames=['predictions'])
```

t[26]:

predictions	0.0	1.0
actual		
0	283	3
1	11	37

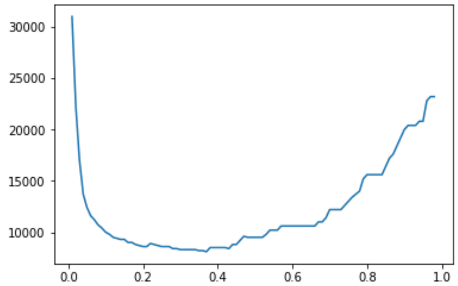
- Plotting the values of the predictions

```
[27]: plt.hist(predictions)
      plt.show()
```



- Minimizing the cost function. The chart indicates how picking a threshold too low results in costs skyrocketing as all customers are given a retention incentive. Meanwhile, setting the threshold too high results in too many lost customers, which ultimately grows to be nearly as costly.

[29]:
```
cutoffs = np.arange(0.01, 0.99, 0.01)
costs = []
for c in cutoffs:
    costs.append(np.sum(np.sum(np.array([[0, 100], [500, 100]]) *
                               pd.crosstab(index=test_data.iloc[:, 0],
                                           columns=np.where(predictions > c, 1, 0)))))

costs = np.array(costs)
plt.plot(cutoffs, costs)
plt.show()
print('Cost is minimized near a cutoff of:', cutoffs[np.argmin(costs)], 'for a cost of:', np.min(costs))
```




```
Cost is minimized near a cutoff of: 0.37 for a cost of: 8100
```

Lab 2: Applied Machine Learning: Building Models for an Amazon Use Case

- The data set here is related to IMDB movie ratings, and we are supposed to predict if a given movie will be nominated or winning on the upcoming award season.

- Importing the necessary libraries

```
In [1]: # Importing libs into the python environment. These functions will be referenced later in the notebook code.

        from __future__ import print_function
        import os
        import pandas as pd
        import pickle
        import matplotlib.pyplot as plt
        import gzip
        import numpy as np
        import seaborn as sns
        import itertools
        from IPython.display import Markdown, display
        from mpl_toolkits.mplot3d import axes3d, Axes3D  # <-- Note the capitalization!
        %matplotlib inline

        sns.set()
```

- Loading data into the bucket which was created

```
In [2]: import boto3
        import botocore
        bucket = 'mlu-data-834449297715-us-west-2-qls-12010358-e19d12fbaf244203' # Update this to the bucket that was create
        prefix = 'data/'

        s3 = boto3.resource('s3')
```

- Visualizing the data related to genres, ratings, awards, releases etc.

```
In [3]: def download_and_display_file(filename,names, title):
    s3.Bucket(bucket).download_file(filename, filename)
    user_info = pd.read_csv(filename, sep='\t', encoding= 'latin1', names = names)
    display(Markdown("**" + title +" Table** \n"))
    display(user_info.head(5))
    return user_info

user_info_genres = download_and_display_file('title_genres.tsv', ['titleId','genres'], 'Genres')
user_info_ratings = download_and_display_file('title_ratings.tsv', ["titleId","rating","ratingCount","topRank","bott
user_info_display = download_and_display_file('title_display.tsv', ["titleId","title","year","adult","runtimeMinutes
user_info_noms = download_and_display_file('award_noms.tsv', ["awardId","eventId","event","eventEditionId","award","
user_info_awards = download_and_display_file('title_awards.tsv', ["titleId","awardId","winner"], 'Awards')
user_info_releases = download_and_display_file('title_releases.tsv', ["titleId","ordering","date","region","premiere
```

Genres Table

	titleId	genres
0	tt0015724	DramaMysteryRomanceThriller
1	tt0035423	ComedyFantasyRomance
2	tt0059900	DramaFantasy
3	tt0064994	ComedyDramaRomance
4	tt0065188	Drama

Rating Table

	titleId	rating	ratingCount	topRank	bottomRank	topRankTV
0	tt0015724	6.2	19	\N	\N	\N
1	tt0035423	6.4	72032	3107	2579	\N

- Merging the data from three different data sets

```
In [4]: df_first_merge = pd.merge(user_info_genres, user_info_ratings, on='titleId', how='inner')
df_second_merge = pd.merge(df_first_merge, user_info_display, on='titleId', how='inner')
df_third_merge = pd.merge(df_second_merge, user_info_releases, on='titleId', how='inner')
```

- Dropping the target variable from the data frame

```
In [5]: df_third_merge = df_third_merge.drop_duplicates(['titleId'])
        df_fourth_merge = pd.merge(df_third_merge,user_info_awards,on='titleId', how='outer' )

        df = df_fourth_merge.drop_duplicates(['titleId'])
        df = df.drop(['imageUri','topRank','bottomRank','topRankTV','ordering','premiereType','festival' ], axis=1)
```

- Uploading the raw data into S3 bucket

```
In [6]: with open('df_pickle_nonoms_new.pkl', 'wb') as handle:
            pickle.dump(df, handle, protocol=pickle.HIGHEST_PROTOCOL)
        s3.Bucket(bucket).upload_file('df_pickle_nonoms_new.pkl','data/df_pickle_nonoms_new.pkl')
```

Review the top 30 rows of optimized table.

- Viewing the first 30 records of the data frame

```
[7]: df.head(30)
```

t[7]:

	titleId	genres	rating	ratingCount	title	year	adult	runtimeMinutes	imageId	type	originalTitle	date
0	tt0015724	DramaMysteryRomanceThriller	6.2	19.0	Dama de noche	1993	0.0	102	rm615620352	movie	Dama de noche	1993-03-18
1	tt0035423	ComedyFantasyRomance	6.4	72032.0	Kate & Leopold	2001	0.0	118	rm2171875072	movie	Kate & Leopold	2002-02-14
7	tt0059900	DramaFantasy	6.8	21.0	Wenn du groÃ bist, lieber Adam	1990	0.0	78	rm2847710208	movie	Wenn du groÃ bist, lieber Adam	1990-02-18
9	tt0064994	ComedyDramaRomance	7.6	1387.0	Larks on a String	1990	0.0	94	rm905747712	movie	SkrivÃ¡nci na niti	1990-11-16
12	tt0065188	Drama	6.7	19.0	Vojtech, receny sirotek	1990	0.0	80	\N	movie	Vojtech, receny sirotek	0000-00-00
14	tt0066498	DramaThriller	7.9	1995.0	The Ear	1990	0.0	94	rm3166657792	movie	Ucho	2003-04-30
16	tt0077432	ActionDrama	6.7	6.0	Dip huet kei bing	1991	0.0	\N	rm4261125120	movie	Dip huet kei bing	1991-10-19

- Data visualization using describe function

```
[9]: df.describe()
```

[9]:

	rating	ratingCount	adult	premiere	wide	winner
count	47781.000000	4.778100e+04	47781.0	47781.000000	47781.000000	21364.000000
mean	6.243308	5.153560e+03	0.0	0.106088	0.596388	0.543157
std	1.395214	4.094635e+04	0.0	0.307954	0.490627	0.498146
min	1.000000	5.000000e+00	0.0	0.000000	0.000000	0.000000
25%	5.400000	1.800000e+01	0.0	0.000000	0.000000	0.000000
50%	6.400000	7.000000e+01	0.0	0.000000	1.000000	1.000000
75%	7.200000	3.890000e+02	0.0	0.000000	1.000000	1.000000
max	10.000000	1.998757e+06	0.0	1.000000	1.000000	1.000000

- Loading the pickle file into padas data frame and dropping some features

```
[10]: s3.Bucket(bucket).download_file('data/df_pickle_nonoms_new.pkl', 'df_pickle_nonoms_new.pkl')
df = pickle.load(open('df_pickle_nonoms_new.pkl', 'rb'))
df = df[df.type == 'movie']
df = df.drop(['imageId', 'originalTitle', 'awardId', 'attributes' ], axis=1)
```

- Display tables with runtimes as \N and also display tables with year as \N

.1]:
```
df[df.runtimeMinutes == r'\N'].head()
```

.1]:

	titleId	genres	rating	ratingCount	title	year	adult	runtimeMinutes	type	date	region	premiere	wide	winner
16	tt0077432	ActionDrama	6.7	6.0	Dip huet kei bing	1991	0.0	\N	movie	1991-10-19	HK	0.0	1.0	NaN
61	tt0094004	ActionThrillerWar	5.7	60.0	Soldier of Fortune	1990	0.0	\N	movie	1990-04-28	JP	0.0	1.0	NaN
62	tt0094045	Documentary	7.0	8.0	Storme: Lady of the Jewel Box	1991	0.0	\N	movie	1991-05-07	US	0.0	0.0	NaN
88	tt0095335	Drama	5.5	6.0	The Vengeance	1995	0.0	\N	movie	1995-02-06	HK	0.0	1.0	NaN
103	tt0096174	Drama	6.9	70.0	Stela	1990	0.0	\N	movie	1990-05-29	XYU	0.0	1.0	NaN

Below will display tables with year `\N`.

.2]:
```
df[df.year == r'\N'].head()
```

.2]:

	titleId	genres	rating	ratingCount	title	year	adult	runtimeMinutes	type	date	region	premiere	wide	winner
130194	tt3329456	CrimeDramaRomance	5.6	58.0	Heartlock	\N	0.0	96	movie	2018-07-24	GB	1.0	0.0	NaN
130718	tt4630466	Horror	8.4	22.0	Virgin Cheerleaders in Chains	\N	0.0	\N	movie	2018-08-09	BR	0.0	1.0	NaN
130933	tt5130442	CrimeDrama	6.0	34.0	22 Chaser	\N	0.0	90	movie	2018-07-01	CA	0.0	1.0	NaN

- A separate column called nomination_winner added which would be either 0 or 1

[13]:
```
for i, mins in df['runtimeMinutes'].iteritems():
    if mins == r'\N':
        better_name = '0'
        df.loc[[i],['runtimeMinutes']] = better_name

for i, year in df['year'].iteritems():
    if year == r'\N':
        better_name = '0'
        df.loc[[i],['year']] = better_name
```

A separate column called `nomination_winner` is added to the DataFrame. If winner column has either `0.0` or `1.0` value, it is assumed that the title has been nominated. Else, the title has not been nominated.

[14]:
```
df['nomination_winner'] = 0
for i, winner in df['winner'].iteritems():
    if winner == (0.0):
        better_name = 1
        df.loc[[i],['nomination_winner']] = better_name
    if winner == (1.0):
        better_name = 1
        df.loc[[i],['nomination_winner']] = better_name
```

- Filling the missing values using **fillna**

In [15]:
```
df.runtimeMinutes = df.runtimeMinutes.astype(float).fillna(0.0)
df.year = df.year.astype(int).fillna(0.0)
```

- Creating data frame based on run time minutes

In [16]:
```
df[(df.runtimeMinutes) < 60].head()
```

Out[16]:

	titleId	genres	rating	ratingCount	title	year	adult	runtimeMinutes	type	date	region	premiere	wide	winner	nomination_winne
16	tt0077432	ActionDrama	6.7	6.0	Dip huet kei bing	1991	0.0	0.0	movie	1991-10-19	HK	0.0	1.0	NaN	(
61	tt0094004	ActionThrillerWar	5.7	60.0	Soldier of Fortune	1990	0.0	0.0	movie	1990-04-28	JP	0.0	1.0	NaN	(
62	tt0094045	Documentary	7.0	8.0	Storme: Lady of the Jewel Box	1991	0.0	0.0	movie	1991-05-07	US	0.0	0.0	NaN	(
88	tt0095335	Drama	5.5	6.0	The Vengeance	1995	0.0	0.0	movie	1995-02-06	HK	0.0	1.0	NaN	(
103	tt0096174	Drama	6.9	70.0	Stela	1990	0.0	0.0	movie	1990-05-29	XYU	0.0	1.0	NaN	(

Below, you will see sample data with `runtimesMinutes` of more than 720 minutes.

In [17]:
```
df[(df.runtimeMinutes) > 720].head()
```

Out[17]:

	titleId	genres	rating	ratingCount	title	year	adult	runtimeMinutes	type	date	region	premiere	wide	winner	nomination_win
127994	tt1447786	Documentary	8.2	8.0	Grandmother Martha	1996	0.0	1452.0	movie	0000-00-00	US	0.0	1.0	NaN	
129583	tt2355497	Documentary	7.1	9.0	Beijing 2003	2004	0.0	9000.0	movie	0000-00-00	CN	0.0	1.0	NaN	

In [18]:
```
df = df[(df.runtimeMinutes) > 60]
df = df[(df.runtimeMinutes) < 720]
df = df[(df.ratingCount) < 20000]
```

- Viewing the optimized data

```
[21]: df_2005 = df[(df.year) == 2005]
      df_2005.head()
```

t[21]:

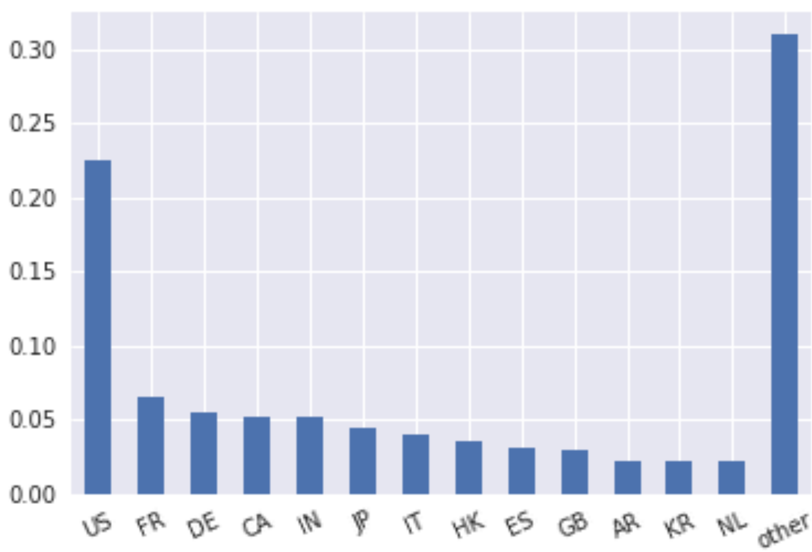
	titleId	genres	rating	ratingCount	title	year	adult	runtimeMinutes	type	date	region	premiere	wide	winner	nomination_win
39	tt0088751	ComedyHorrorSci-Fi	5.8	178.0	The Naked Monster	2005	0.0	100.0	movie	2005-04-22	US	1.0	0.0	NaN	
27448	tt0118141	Drama	5.9	806.0	What Is It?	2005	0.0	72.0	movie	2005-01-27	US	0.0	1.0	1.0	
42018	tt0148403	Documentary	4.9	50.0	Life, Love & Celluloid	2005	0.0	90.0	movie	2005-07-30	PL	0.0	0.0	NaN	
44097	tt0160706	CrimeDramaThriller	5.7	852.0	The Prodigy	2005	0.0	120.0	movie	2007-07-27	FI	1.0	0.0	NaN	
49561	tt0179803	Comedy	5.6	116.0	Angels with Angles	2005	0.0	87.0	movie	2005-12-16	US	0.0	1.0	NaN	

- Setting flags for feature selection

```
In [22]: # Selection of different features

         feature_winner = 0          # Select this feature to make prediction on award winner.
                                     # Disable this feautre to make prediction on nomination winners.

         feature_pca_2D = 0          # Select this feature to perform Principal Component Analysis of 2 components.
         feature_pca_3D = 1          # Select this feature to perform Principal Component Analysis of 3 components.

         feature_premiere = 0        # Select this feature to limit analysis on limited premiered movies.
         feature_wide = 0            # Select this feature to limit analysis on world wide premiered movies.
         feature_premiere_wide = 1   # Select this feature to include analysis on both limited and wide premiered movies.


         # Normalize features
         normalize_flag = 0

         #Enable plotting
         plot_flag = 1

         #Feaure Selection
         US_flag = 1                 # Select this feature to limit analysis on US based movies.

         #Model Selection flags
         LR_flag = 1
         DT_flag = 1
         RF_flag = 1
         GB_flag = 1
         NN_flag = 1
         SVM_flag = 1
```

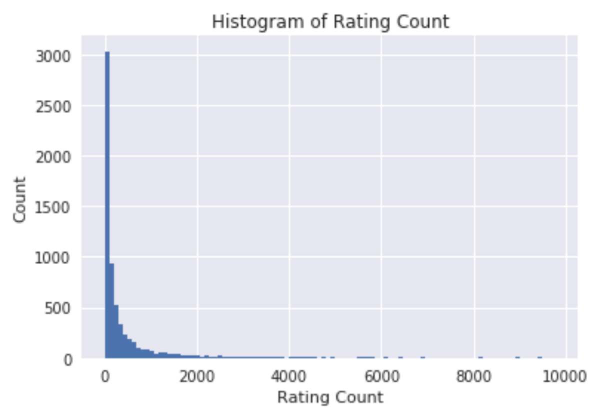
- Exploring the data by plotting

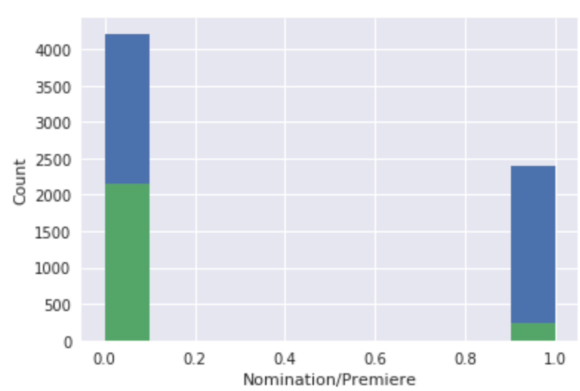
```
if feature_winner:
    df = df[(df.nomination_winner) == 1]

    for i, winner in df['winner'].iteritems():
        if winner == (0.0):
            better_name = 0
            df.loc[[i],['nomination_winner']] = better_name
        if winner == (1.0):
            better_name = 1
```



- Plot rating vs rating count & Histogram of Rating


nomination_winner
ratingCount
rating
timeMinutes


Histogram of Rating Count
Count
Rating Count

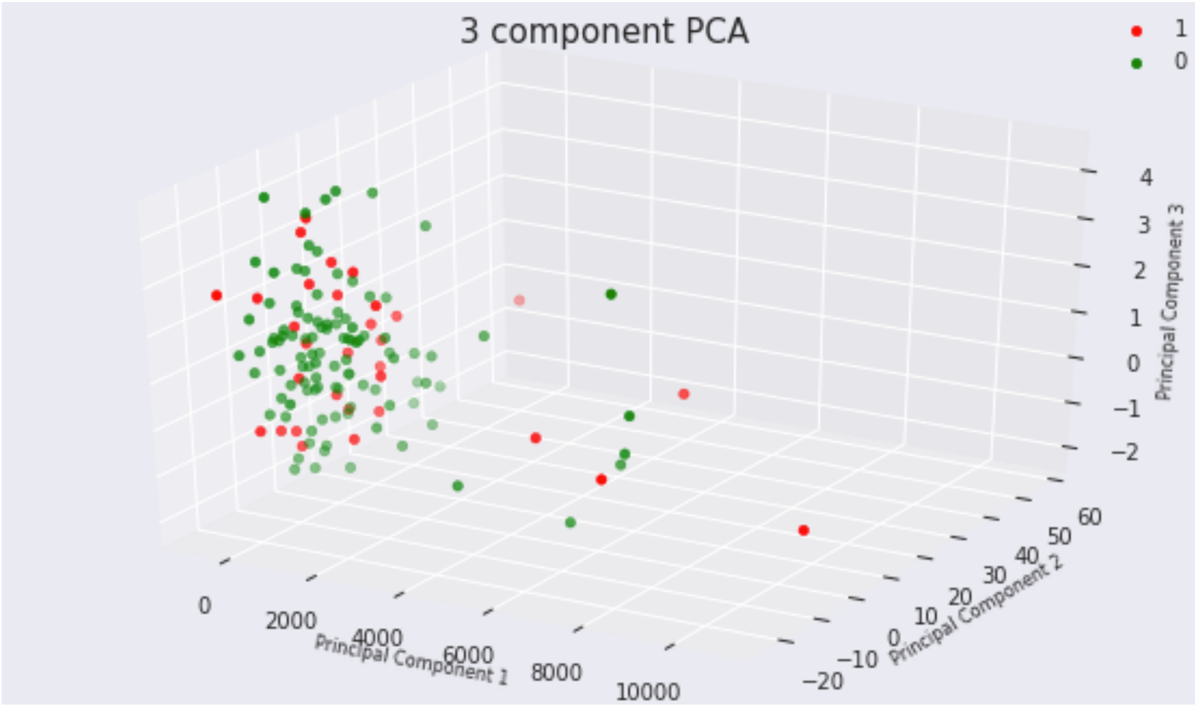

Count
Nomination/Premiere

- Feature selection

```
n [26]:
if feature_premiere_wide:
    X_train = df[['rating', 'ratingCount', 'runtimeMinutes','premiere','wide']]
elif feature_premiere:
    X_train = df[['rating', 'ratingCount', 'runtimeMinutes','premiere' ]]
elif feature_wide:
    X_train = df[['rating', 'ratingCount',  'runtimeMinutes' ,'wide']]
else :
    X_train = df[['rating', 'ratingCount',  'runtimeMinutes']]


Y_train = df['nomination_winner']

if normalize_flag:
    X_train=(StandardScaler().fit_transform(X_train ))
```

- PCA for reducing to 3 main features

```
In [27]: #PCA

if feature_pca_3D:
    pca = decomposition.PCA(n_components=3)


    principalComponents = pca.fit_transform(X_train )

    principalDf = pd.DataFrame(data = principalComponents
                 , columns = ['principal component 1', 'principal component 2','principal component 3'])


    finalDf = pd.concat([principalDf, Y_train], axis = 1)
    finalDf = finalDf.head(2000)
    targets = [1, 0 ]
    colors = ['r', 'g' ]

    my_dpi = 96
```



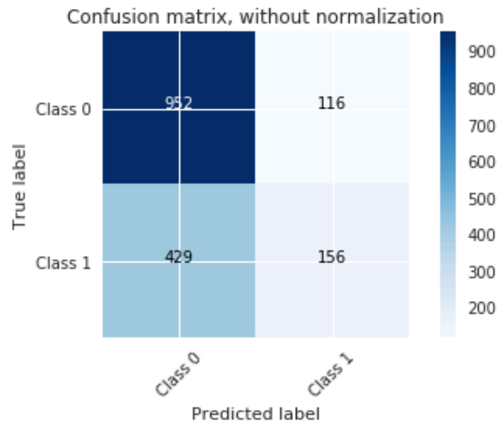
- Train and Test data split

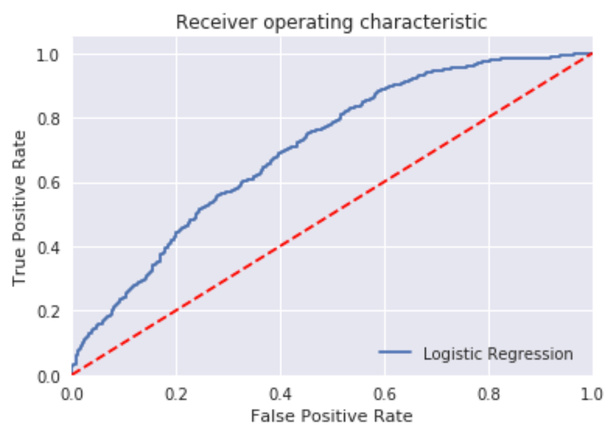
In [28]:

```
x_train, x_test, y_train, y_test = train_test_split(X_train, Y_train, test_size=0.25, random_state=0)
```

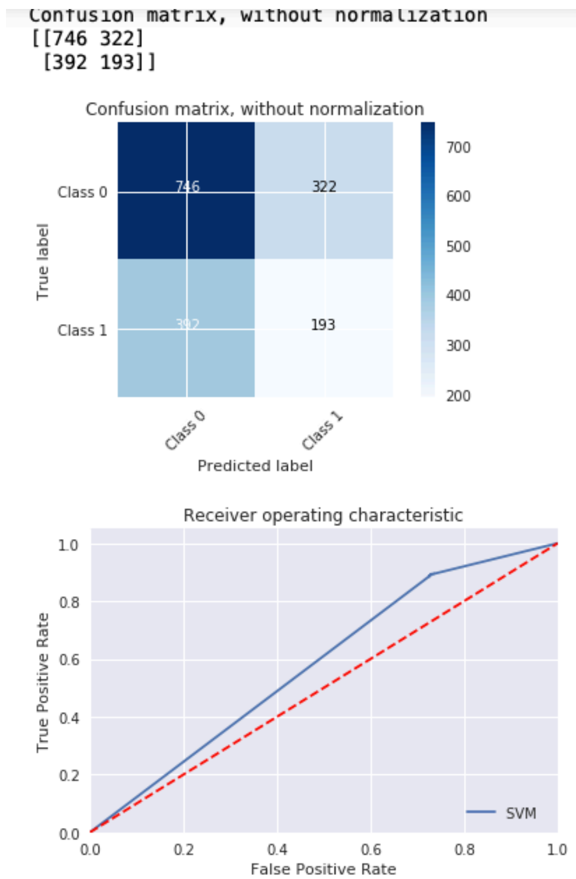
- Fitting Logistic regression model

```
Confusion matrix, without normalization
[[952 116]
 [429 156]]
```



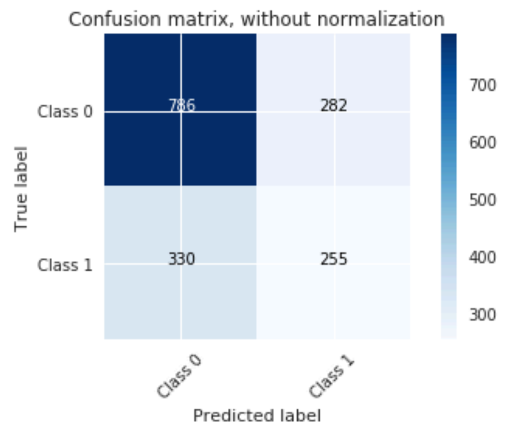


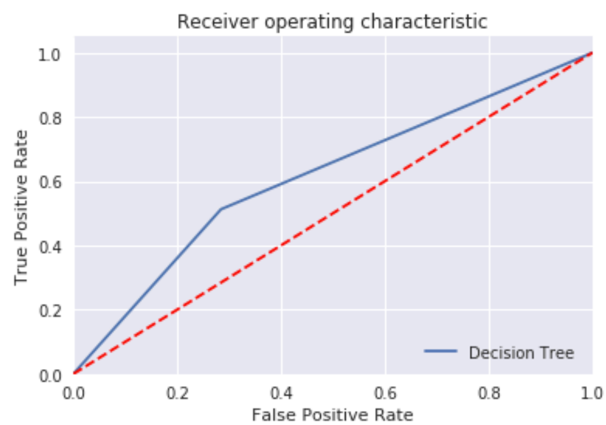
- Fitting Support vector machine



- Fitting Decision Tree model

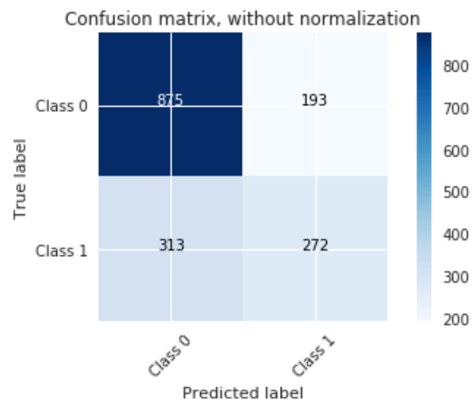
```
Confusion matrix, without normalization
[[786 282]
 [330 255]]
```



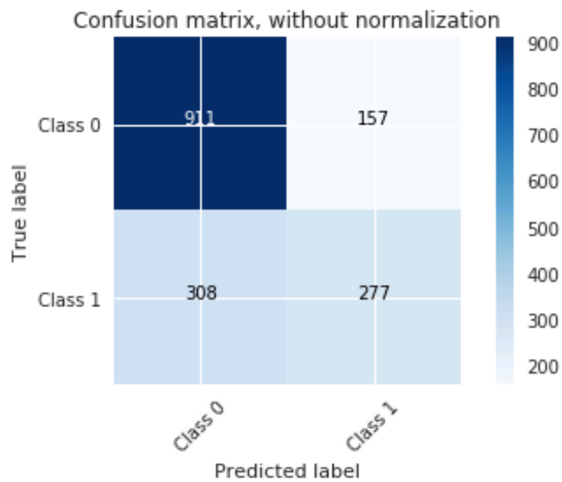


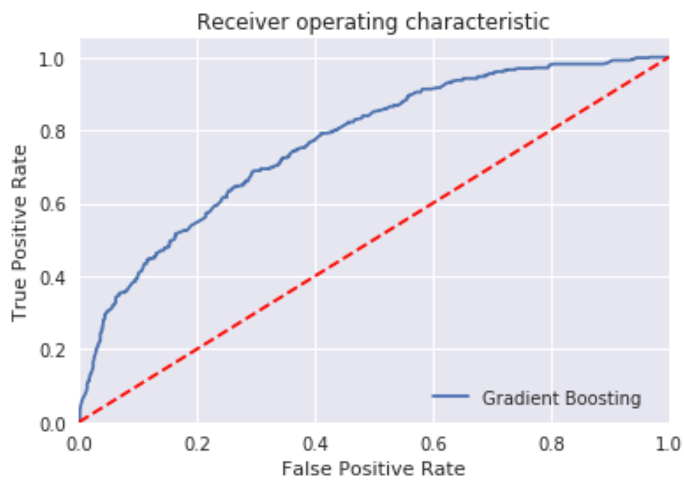
- Fitting Random forest model

```
[[875 193]
 [313 272]]
```



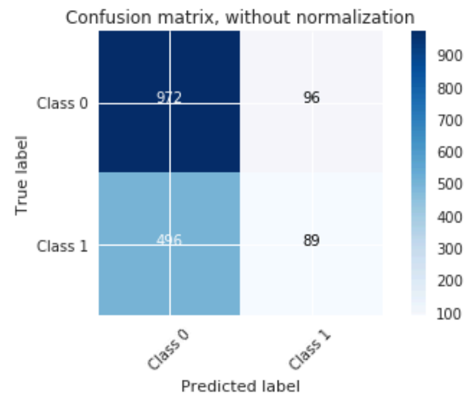
- Fitting Gaussian model

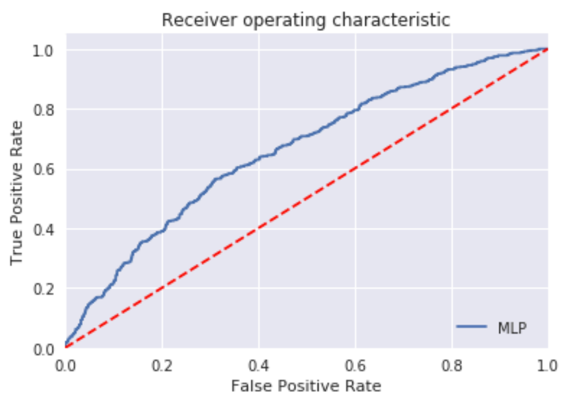




- Fitting neural network

```
[[972  96]
 [496  89]]
```





- Viewing the metrics of every model which was fitted to determine the right model

LR **Model Classification Report**

	precision	recall	f1-score	support
0	0.69	0.89	0.78	1068
1	0.57	0.27	0.36	585
micro avg	0.67	0.67	0.67	1653
macro avg	0.63	0.58	0.57	1653
weighted avg	0.65	0.67	0.63	1653

SVM **Model Classification Report**

	precision	recall	f1-score	support
0	0.66	0.70	0.68	1068
1	0.37	0.33	0.35	585
micro avg	0.57	0.57	0.57	1653
macro avg	0.52	0.51	0.51	1653
weighted avg	0.56	0.57	0.56	1653

GB **Model Classification Report**

	precision	recall	f1-score	support
0	0.75	0.85	0.80	1068
1	0.64	0.47	0.54	585
micro avg	0.72	0.72	0.72	1653
macro avg	0.69	0.66	0.67	1653
weighted avg	0.71	0.72	0.71	1653

NN **Model Classification Report**

	precision	recall	f1-score	support
0	0.66	0.91	0.77	1068
1	0.48	0.15	0.23	585
micro avg	0.64	0.64	0.64	1653
macro avg	0.57	0.53	0.50	1653
weighted avg	0.60	0.64	0.58	1653

DT **Model Classification Report**

	precision	recall	f1-score	support
0	0.70	0.74	0.72	1068
1	0.47	0.44	0.45	585
micro avg	0.63	0.63	0.63	1653
macro avg	0.59	0.59	0.59	1653
weighted avg	0.62	0.63	0.63	1653

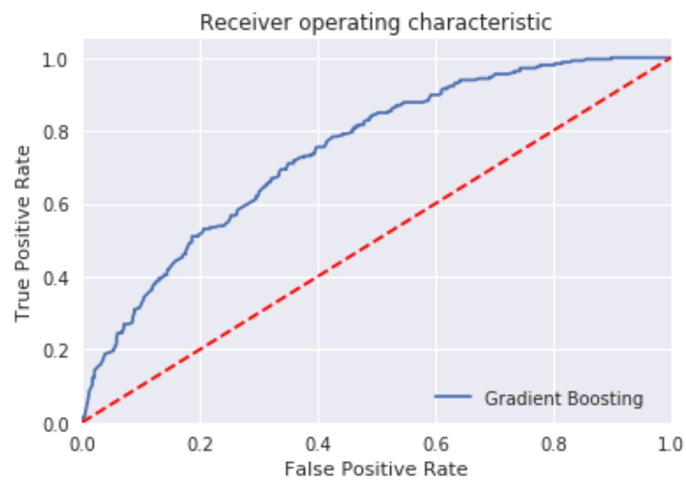
RF **Model Classification Report**

	precision	recall	f1-score	support
0	0.74	0.82	0.78	1068
1	0.58	0.46	0.52	585
micro avg	0.69	0.69	0.69	1653
macro avg	0.66	0.64	0.65	1653
weighted avg	0.68	0.69	0.68	1653

GB **Model Classification Report**

	precision	recall	f1-score	support
0	0.75	0.85	0.80	1068
1	0.64	0.47	0.54	585
micro avg	0.72	0.72	0.72	1653
macro avg	0.69	0.66	0.67	1653
weighted avg	0.71	0.72	0.71	1653

- By Comparing the different models, I found that gradient boosting classifier was showing a better accuracy and roc value. Hence, I ran the same model against the 2005 data and below were the results of the same.




```
GB Accuracy 0.6914498141263941
roc score 0.5942878930931803
Classification Report
====================================================
              precision    recall  f1-score   support

           0       0.75      0.84      0.79       562
           1       0.49      0.35      0.41       245

   micro avg       0.69      0.69      0.69       807
   macro avg       0.62      0.59      0.60       807
weighted avg       0.67      0.69      0.67       807

Confusion matrix, without normalization
[[473  89]
 [160  85]]
```