Association Rule feature Minin (ARM) in Network Intrusion Detection System

April 15, 2020

Association Rule feature Mining (ARM) in Network Intrusion Detection System

1 1. Business Problem

1.1 1.1. Description

Source: https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets

Data: CISCO Networking Dataset - The UNSW-NB15 Dataset

Download UNSW-NB15 - csv file.

Problem statement :

Classify the given network is intrusion or normal based on evidence from The raw network packets of the UNSW-NB 15 dataset. it was created by the IXIA PerfectStorm tool in the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS).

2 2. Machine Learning Problem Formulation

2.1 2.1. Data

2.1.1 2.1.1. Data Overview

- Source: https://cloudstor.aarnet.edu.au/plus/index.php/s/2DhnLGDdEECo4ys
- We have multiple data files: download the UNSW-NB15 - csv file this file contains a following structure

```
a part of training and testing set - folder contains train and test data csv files
NUSW-NB15_features.csv - Feature description
NUSW-NB15_GT.csv
The UNSW-NB15 description.pdf
UNSW-NB15_1.csv
UNSW-NB15_2.csv
UNSW-NB15_3.csv
UNSW-NB15_4.csv
UNSW-NB15_LIST_EVENTS.csv
```

- These features are described in **UNSW-NB15_features.csv** file.

- The total number of records is two million and 540,044 which are stored in the four CSV files, namely, UNSW-NB15_1.csv, UNSW-NB15_2.csv, UNSW-NB15_3.csv and UNSW-NB15_4.csv.
- The ground truth table is named **UNSW-NB15_GT.csv** and the list of event file is called UNSW-NB15_LIST_EVENTS.csv.
- A partition from this dataset is configured as a training set and testing set, namely, **UNSW_NB15_training-set.csv** and **UNSW_NB15_testing-set.csv** respectively.
- The number of records in the training set is 82,332 records and the testing set is 175,341 records from the different types, attack and normal.Figure 1 and 2 show the testbed configuration dataset and the method of the feature creation of the UNSW-NB15, respectively.
- Data file's information:

   both train and test files contains 45 columns ['dur', 'spkts', 'dpkts', 'sb

  'sload', 'dload', 'sloss', 'dloss', 'sinpkt', 'dinpkt', 'sjit', 'djit', 'swin', 'stcpb', 'dtcpb', 'dwin', 'tcprtt', 'synack', 'ackdat', 'smean', 'dmean', 'trans_depth', 'response_body_len', 'ct_srv_src', 'ct_state_ttl', 'ct_dst_ltm', 'ct_src_dport_ltm', 'ct_dst_sport_ltm', 'ct_dst_src_ltm', 'is_ftp_login', 'ct_ftp_cmd', 'ct_flw_http_mthd', 'ct_src_ltm', 'ct_srv_dst', 'is_sm_ips_ports']

  Categorical (4) columns

  Numerical (41) columns

2.2 2.2. Mapping the real-world problem to an ML problem

2.2.1 2.2.1. Type of Machine Learning Problem

There are two different class normal or attack. Find the network is normal or intrusio

2.2.2 2.2.2. Performance Metric

metric used to identify the performance of the model

Metric(s): * AUC and f1-score * Confusion matrix * FAR - false alarm rate should be as minimum as possible

2.2.3 2.2.3. Machine Learing Objectives and Constraints

Objective: Predict the probability of each data-point whether the network is normal or

[1]:
```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
```

```
%matplotlib inline
from tqdm import tqdm
from mlxtend.frequent_patterns import apriori, association_rules, fpgrowth
```

[2]:
```
train_data =pd.read_csv("data/UNSW_NB15_training-set.csv")
print(train_data.shape)
train_data.head()
```

```
(82332, 45)
```

[2]:

	id	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	\
0	1	0.000011	udp	-	INT	2	0	496	0	
1	2	0.000008	udp	-	INT	2	0	1762	0	
2	3	0.000005	udp	-	INT	2	0	1068	0	
3	4	0.000006	udp	-	INT	2	0	900	0	
4	5	0.000010	udp	-	INT	2	0	2126	0	

	rate	…	ct_dst_sport_ltm	ct_dst_src_ltm	is_ftp_login	\
0	90909.0902	…	1	2	0	
1	125000.0003	…	1	2	0	
2	200000.0051	…	1	3	0	
3	166666.6608	…	1	3	0	
4	100000.0025	…	1	3	0	

	ct_ftp_cmd	ct_flw_http_mthd	ct_src_ltm	ct_srv_dst	is_sm_ips_ports	\
0	0	0	1	2	0	
1	0	0	1	2	0	
2	0	0	1	3	0	
3	0	0	2	3	0	
4	0	0	2	3	0	

	attack_cat	label
0	Normal	0
1	Normal	0
2	Normal	0
3	Normal	0
4	Normal	0

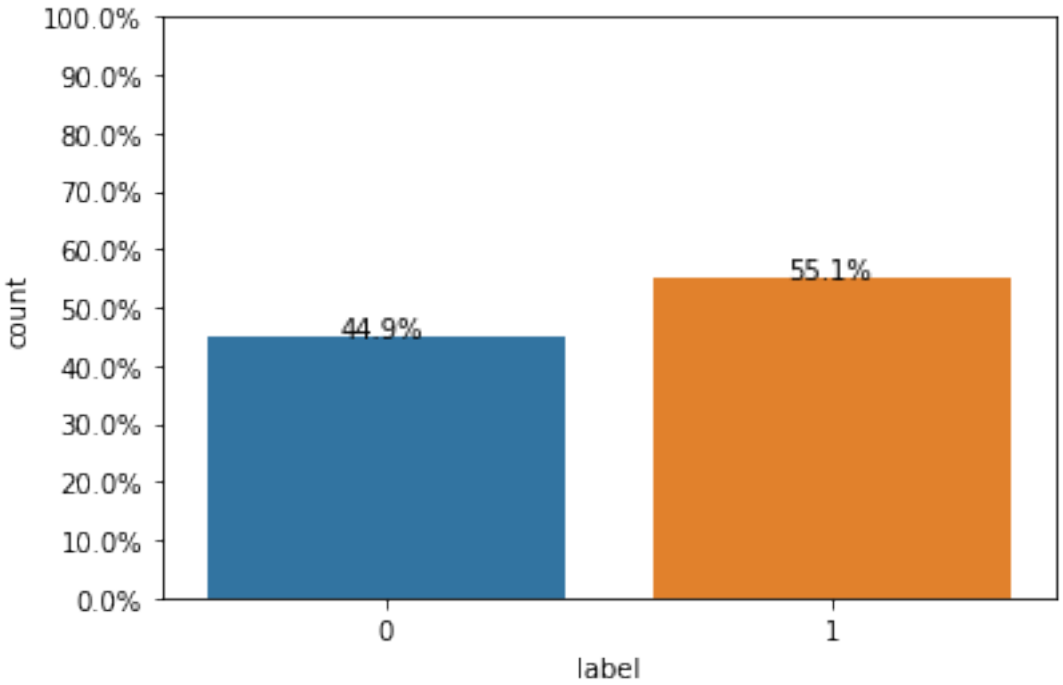
[5 rows x 45 columns]

3 3. Exploratory Data Analysis

3.1 3.1 visualizing class label

[6]:
```
total = len(train_data)*1.
ax=sns.countplot(x="label", data=train_data)
for p in ax.patches:
    print(p)
    ax.annotate('{:.1f}%'.format(100*p.get_height()/total), (p.get_x()+0.3, p.
 ↪get_height()+5))

#put 11 ticks (therefore 10 steps), from 0 to the total number of rows in the␣
 ↪dataframe
ax.yaxis.set_ticks(np.linspace(0, total, 11))
print(ax.yaxis.get_majorticklocs())
#adjust the ticklabel to the desired format, without changing the position of␣
 ↪the ticks.
ax.set_yticklabels(map('{:.1f}%'.format, 100*ax.yaxis.get_majorticklocs()/
 ↪total))
plt.savefig('class_label.png')
plt.show()
```

```
Rectangle(xy=(-0.4, 0), width=0.8, height=37000, angle=0)
Rectangle(xy=(0.6, 0), width=0.8, height=45332, angle=0)
[    0.   8233.2 16466.4 24699.6 32932.8 41166.  49399.2 57632.4 65865.6
 74098.8 82332. ]
```



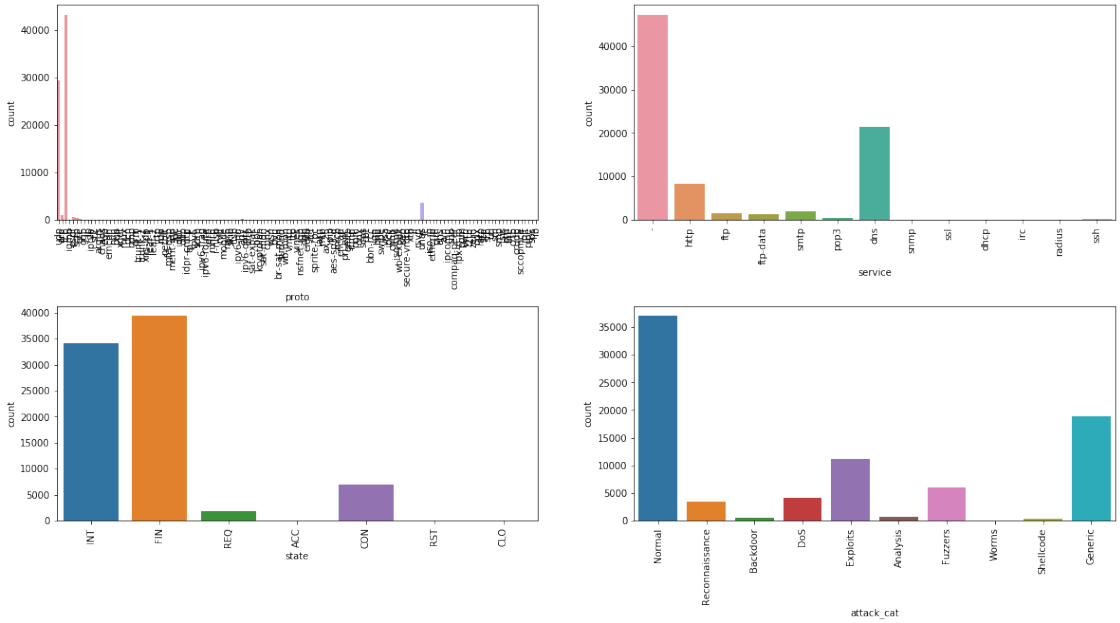
Above plot show that the dataset is not an imbalanced dataset

3.2 3.2 visualizing categorical data

[8]:
```
cat_feature = train_data.select_dtypes(include=['category', object]).columns
cat_feature
```

[8]:
```
Index(['proto', 'service', 'state', 'attack_cat'], dtype='object')
```

[9]:
```
fig, ax = plt.subplots(2, 2, figsize=(20, 10))
plt.subplots_adjust(hspace = 0.4)
for col, subplot in zip(cat_feature, ax.flatten()):
    sns.countplot(train_data[col], ax=subplot)
    for label in subplot.get_xticklabels():
        label.set_rotation(90)
plt.savefig('cate_f.png')
plt.show()
```



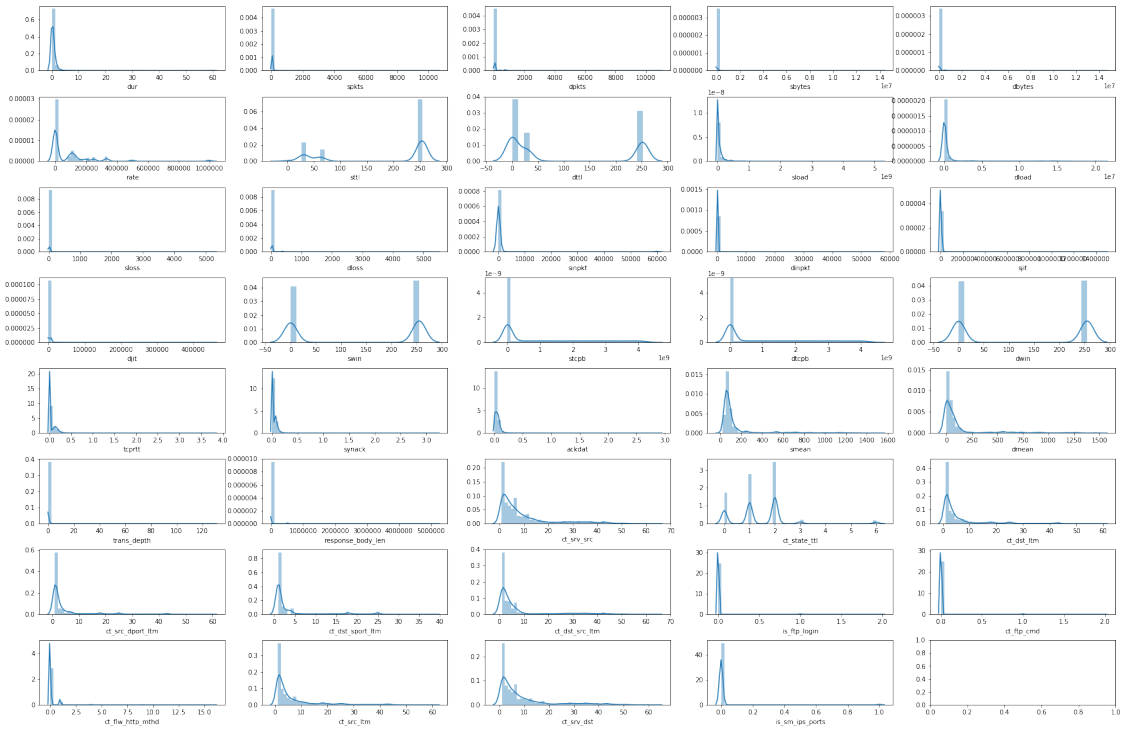
From the categorical data we can see the data imbalance. Also “proto” category has more than 200 categories. Other columns have less than or equal 13 columns

3.3 3.2 visualizing numerical data and its distribution

[10]:
```
numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
num_features = train_data.drop(['id','label'], axis=1).
 ↪select_dtypes(include=numerics).columns
num_features
```

[10]:
```
Index(['dur', 'spkts', 'dpkts', 'sbytes', 'dbytes', 'rate', 'sttl', 'dttl',
       'sload', 'dload', 'sloss', 'dloss', 'sinpkt', 'dinpkt', 'sjit', 'djit',
       'swin', 'stcpb', 'dtcpb', 'dwin', 'tcprtt', 'synack', 'ackdat', 'smean',
       'dmean', 'trans_depth', 'response_body_len', 'ct_srv_src',
       'ct_state_ttl', 'ct_dst_ltm', 'ct_src_dport_ltm', 'ct_dst_sport_ltm',
       'ct_dst_src_ltm', 'is_ftp_login', 'ct_ftp_cmd', 'ct_flw_http_mthd',
       'ct_src_ltm', 'ct_srv_dst', 'is_sm_ips_ports'],
      dtype='object')
```

[11]:
```
fig, ax = plt.subplots(8, 5, figsize=(30, 20))
plt.subplots_adjust(hspace = 0.4)
for col, splot in zip(num_features, ax.flatten()):
    sns.distplot(train_data[col], ax=splot)
plt.savefig('num_f.png')
plt.show()
```



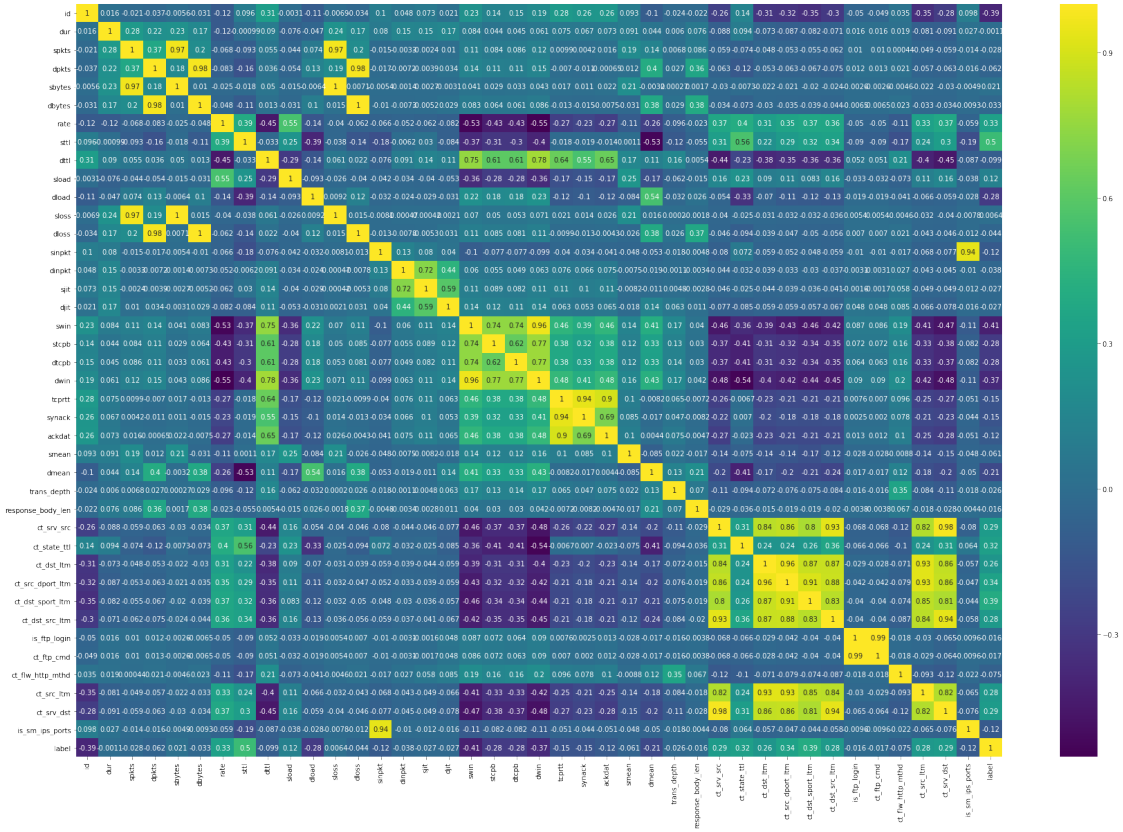
from the distribution plot each feature have one value which is occurring more number of times

than other values As we can see from numerical data distribution there few features which are highly correlated with each other.

3.4 3.3 Correlation of data

[12]:
```
df_corr = train_data.corr()
```

[13]:
```
plt.figure(figsize=(30,20))
sns.heatmap(df_corr, annot=True, cmap=plt.cm.viridis)
plt.savefig('corr_mat.png')
plt.show()
```



- From above visualization we can clearly see that there are few columns which are having high correlation with one another.
- we will find correlation and distribution of those columns and eliminate the necessary one

[263]:
```
high_corr_var=np.where(df_corr>0.95)
```

[264]:
```
high_corr_var
```

[264]:
```
(array([ 0,  1,  2,  2,  2,  3,  3,  3,  4,  4,  4,  5,  5,  5,  6,  7,  8,
         9, 10, 11, 11, 11, 12, 12, 12, 13, 14, 15, 16, 17, 17, 18, 19, 20,
        20, 21, 22, 23, 24, 25, 26, 27, 28, 28, 29, 30, 30, 31, 31, 32, 33,
        34, 34, 35, 35, 36, 37, 38, 38, 39, 40], dtype=int64),
 array([ 0,  1,  2,  4, 11,  3,  5, 12,  2,  4, 11,  3,  5, 12,  6,  7,  8,
         9, 10,  2,  4, 11,  3,  5, 12, 13, 14, 15, 16, 17, 20, 18, 19, 17,
        20, 21, 22, 23, 24, 25, 26, 27, 28, 38, 29, 30, 31, 30, 31, 32, 33,
        34, 35, 34, 35, 36, 37, 28, 38, 39, 40], dtype=int64))
```

[265]:
```
#ref: https://stackoverflow.com/questions/29294983/
 ↪how-to-calculate-correlation-between-all-columns-and-remove-highly-correlated-on
high_corr_var=[(df_corr.columns[x],df_corr.columns[y]) for x,y in␣
 ↪zip(*high_corr_var) if x!=y and x<y]
```

[266]:
```
high_corr_var
```

[266]:
```
[('spkts', 'sbytes'),
 ('spkts', 'sloss'),
 ('dpkts', 'dbytes'),
 ('dpkts', 'dloss'),
 ('sbytes', 'sloss'),
 ('dbytes', 'dloss'),
 ('swin', 'dwin'),
 ('ct_srv_src', 'ct_srv_dst'),
 ('ct_dst_ltm', 'ct_src_dport_ltm'),
 ('is_ftp_login', 'ct_ftp_cmd')]
```

3.5 3.4 Feature Description

[15]:
```
data_features =pd.read_csv("data/NUSW-NB15_features.csv", engine='python')
print(data_features.shape)
```

```
(49, 4)
```

[185]:
```
data_features.head(49)
```

[185]:

	No.	Name	Type	\
0	1	srcip	nominal	
1	2	sport	integer	
2	3	dstip	nominal	
3	4	dsport	integer	
4	5	proto	nominal	
5	6	state	nominal	
6	7	dur	Float	
7	8	sbytes	Integer	
8	9	dbytes	Integer	
9	10	sttl	Integer	

10	11	dttl	Integer
11	12	sloss	Integer
12	13	dloss	Integer
13	14	service	nominal
14	15	Sload	Float
15	16	Dload	Float
16	17	Spkts	integer
17	18	Dpkts	integer
18	19	swin	integer
19	20	dwin	integer
20	21	stcpb	integer
21	22	dtcpb	integer
22	23	smeansz	integer
23	24	dmeansz	integer
24	25	trans_depth	integer
25	26	res_bdy_len	integer
26	27	Sjit	Float
27	28	Djit	Float
28	29	Stime	Timestamp
29	30	Ltime	Timestamp
30	31	Sintpkt	Float
31	32	Dintpkt	Float
32	33	tcprtt	Float
33	34	synack	Float
34	35	ackdat	Float
35	36	is_sm_ips_ports	Binary
36	37	ct_state_ttl	Integer
37	38	ct_flw_http_mthd	Integer
38	39	is_ftp_login	Binary
39	40	ct_ftp_cmd	integer
40	41	ct_srv_src	integer
41	42	ct_srv_dst	integer
42	43	ct_dst_ltm	integer
43	44	ct_src_ ltm	integer
44	45	ct_src_dport_ltm	integer
45	46	ct_dst_sport_ltm	integer
46	47	ct_dst_src_ltm	integer
47	48	attack_cat	nominal
48	49	Label	binary

	Description
0	Source IP address
1	Source port number
2	Destination IP address
3	Destination port number
4	Transaction protocol
5	Indicates to the state and its dependent proto…

```
6                                Record total duration
7             Source to destination transaction bytes
8              Destination to source transaction bytes
9            Source to destination time to live value
10            Destination to source time to live value
11            Source packets retransmitted or dropped
12       Destination packets retransmitted or dropped
13  http, ftp, smtp, ssh, dns, ftp-data ,irc  and …
14                               Source bits per second
15                          Destination bits per second
16                 Source to destination packet count
17                  Destination to source packet count
18               Source TCP window advertisement value
19          Destination TCP window advertisement value
20                     Source TCP base sequence number
21                  Destination TCP base sequence number
22  Mean of the ?ow packet size transmitted by the…
23  Mean of the ?ow packet size transmitted by the…
24  Represents the pipelined depth into the connec…
25  Actual uncompressed content size of the data t…
26                                Source jitter (mSec)
27                           Destination jitter (mSec)
28                                   record start time
29                                    record last time
30              Source interpacket arrival time (mSec)
31         Destination interpacket arrival time (mSec)
32  TCP connection setup round-trip time, the sum …
33  TCP connection setup time, the time between th…
34  TCP connection setup time, the time between th…
35  If source (1) and destination (3)IP addresses …
36  No. for each state (6) according to specific r…
37  No. of flows that has methods such as Get and …
38  If the ftp session is accessed by user and pas…
39     No of flows that has a command in ftp session.
40  No. of connections that contain the same servi…
41  No. of connections that contain the same servi…
42  No. of connections of the same destination add…
43  No. of connections of the same source address …
44  No of connections of the same source address (…
45  No of connections of the same destination addr…
46  No of connections of the same source (1) and t…
47  The name of each attack category. In this data…
48               0 for normal and 1 for attack records
```

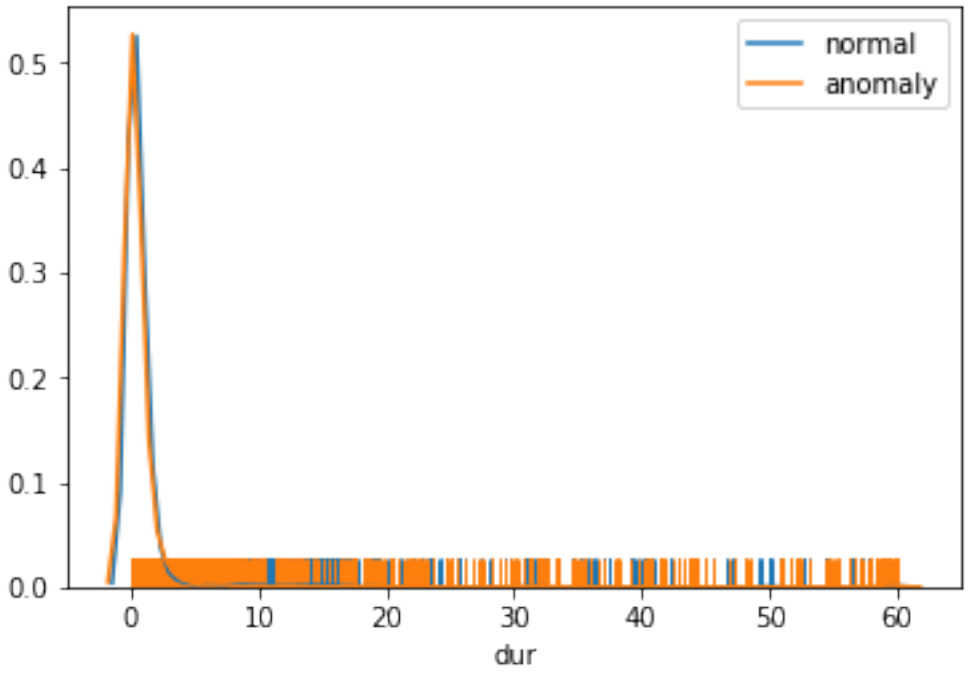
3.5.1 3.4.1 dur

[196]:
```
data_features[6:7]
```

[196]:
```
    No. Name  Type             Description
6    7  dur   Float  Record total duration
```

[181]:
```
sns.distplot(train_data[train_data['label']==0]['dur'], label='normal',␣
↪hist=False, rug=True)
sns.distplot(train_data[train_data['label']==1]['dur'], label='anomaly',␣
↪hist=False, rug=True)
plt.legend()
```

[181]:
```
<matplotlib.legend.Legend at 0x20677228a90>
```



dur is the total rocrd duration for both anomaly and normal.

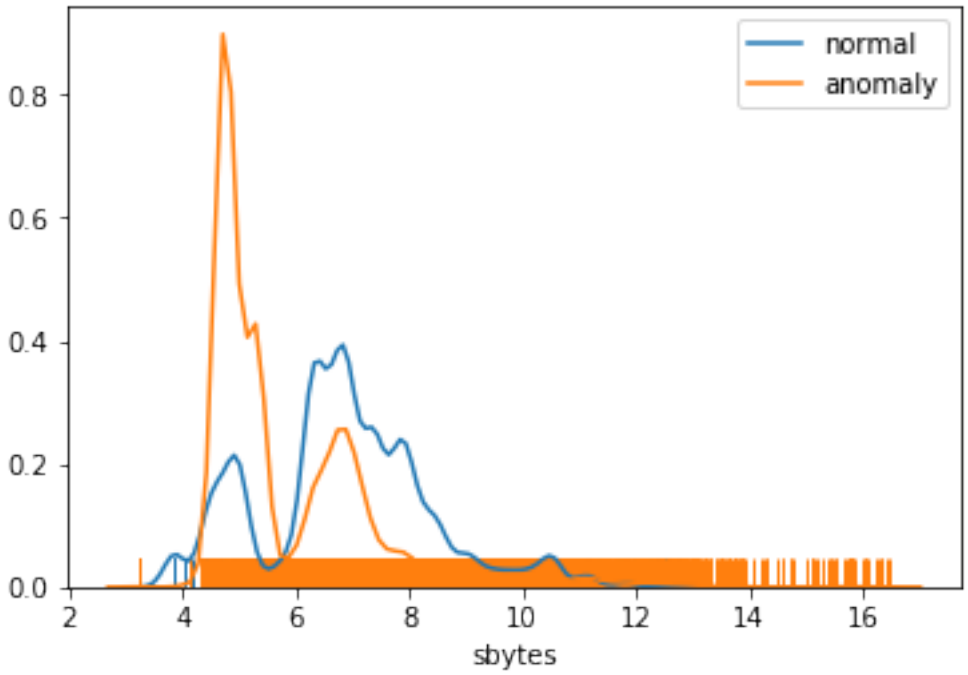
3.5.2 3.4.2 sbytes - dbytes

[197]:
```
data_features[7:9]
```

[197]:
```
    No.    Name    Type                                  Description
7    8  sbytes  Integer  Source to destination transaction bytes
8    9  dbytes  Integer   Destination to source transaction bytes
```

[255]:
```
sns.distplot(train_data[train_data['label']==0]['sbytes'].apply(np.log1p),␣
↪label='normal', hist=False, rug=True)
sns.distplot(train_data[train_data['label']==1]['sbytes'].apply(np.log1p),␣
↪label='anomaly', hist=False, rug=True)
plt.legend()
```

[255]:
```
<matplotlib.legend.Legend at 0x17db1ccd5f8>
```



- applied logarithm of x log(1+x) for some of the features to visualize properly

3.5.3 3.4.3 sloss - dloss

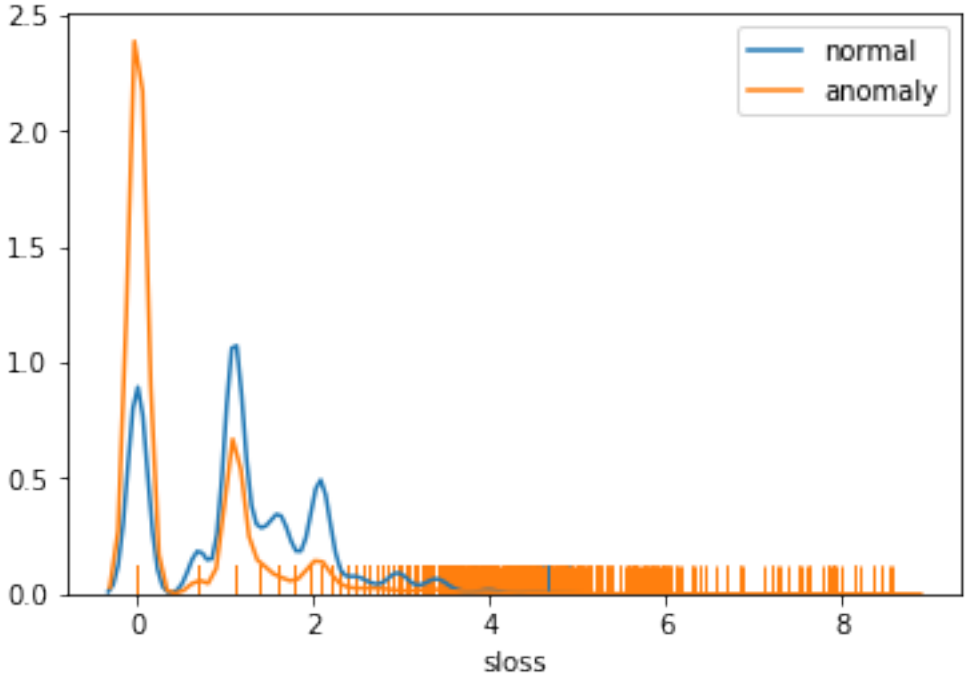
[217]:
```
data_features[11:13]
```

[217]:

	No.	Name	Type	Description
11	12	sloss	Integer	Source packets retransmitted or dropped
12	13	dloss	Integer	Destination packets retransmitted or dropped

[254]:
```
sns.distplot(train_data[train_data['label']==0]['sloss'].apply(np.log1p),␣
↪label='normal', hist=False, rug=True)
sns.distplot(train_data[train_data['label']==1]['sloss'].apply(np.log1p),␣
↪label='anomaly', hist=False, rug=True)
plt.legend()
```

[254]: <matplotlib.legend.Legend at 0x17da4084080>



- sloss and dloss have same disruption as we can see in overall numerical distribution plot.
-

3.5.4 3.4.4 spkts - dpkts

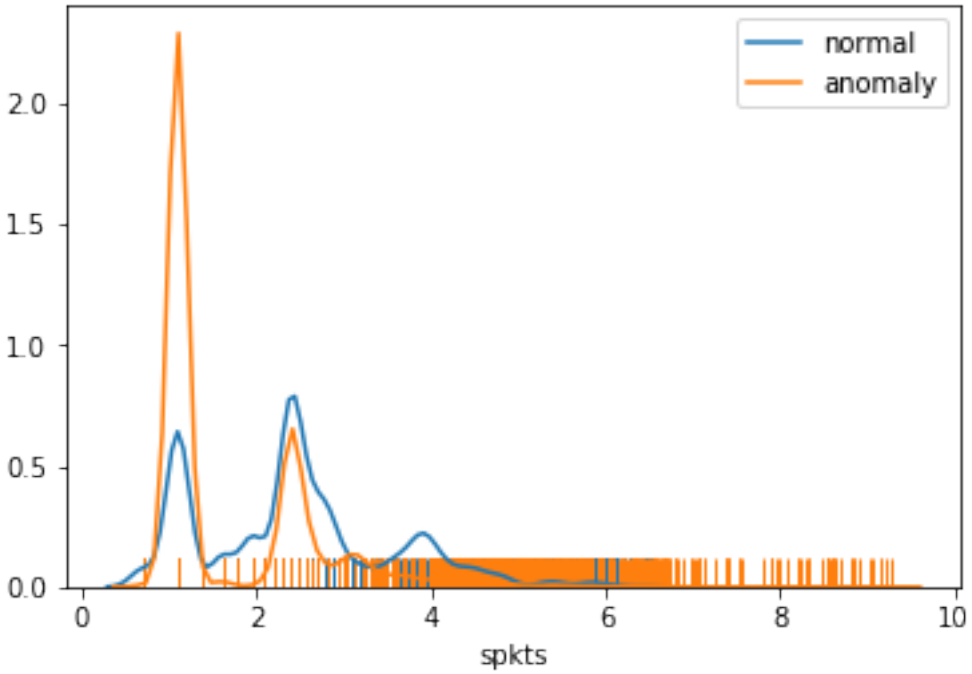
[208]:
```
data_features[16:18]
```

[208]:

	No.	Name	Type	Description
16	17	Spkts	integer	Source to destination packet count
17	18	Dpkts	integer	Destination to source packet count

[223]:
```
sns.distplot(train_data[train_data['label']==0]['spkts'].apply(np.log1p),␣
↪label='normal', hist=False, rug=True)
sns.distplot(train_data[train_data['label']==1]['spkts'].apply(np.log1p),␣
↪label='anomaly', hist=False, rug=True)
plt.legend()
```

[223]: <matplotlib.legend.Legend at 0x2059acbcc18>

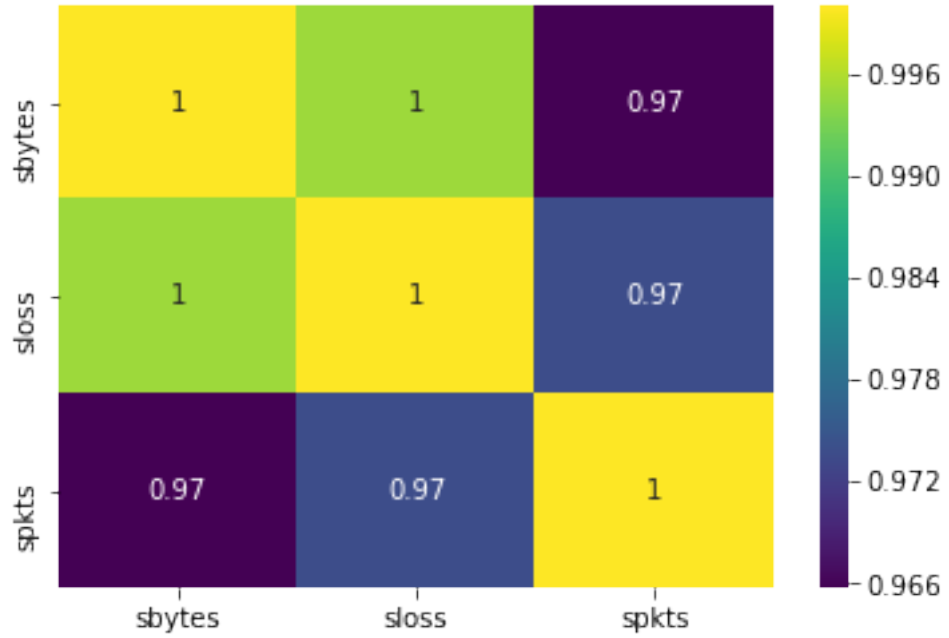


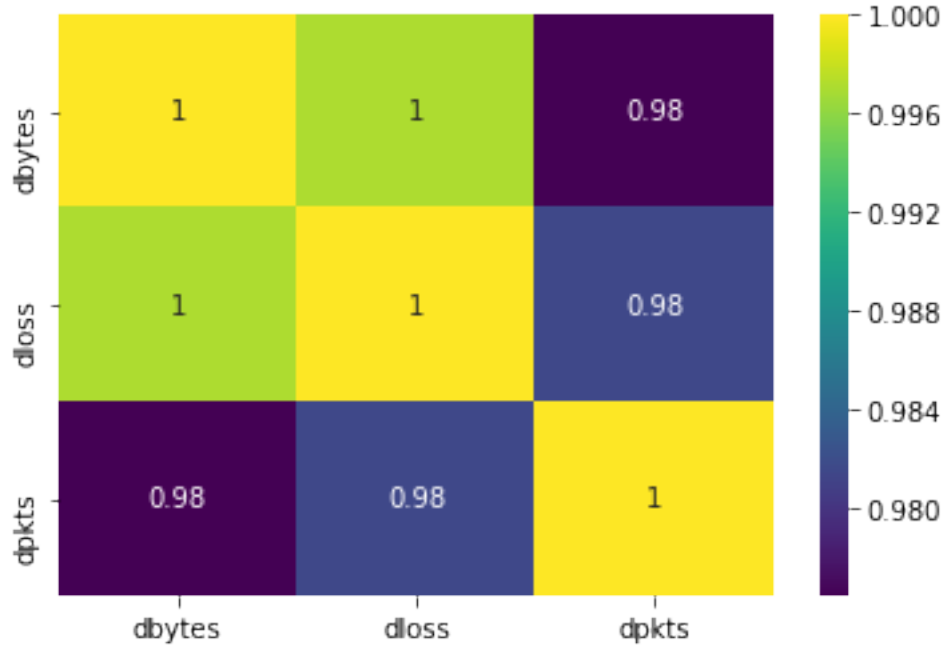
- these sbytes and dbytes have same distribution. anomaly we can see it peaks to 0.8 and above. normal is below 0.4

[243]:
```
print(train_data[['sbytes','sloss','spkts']].corr())
print(train_data[['dbytes','dloss','dpkts']].corr())
```

```
          sbytes     sloss     spkts
sbytes  1.000000  0.995027  0.965750
sloss   0.995027  1.000000  0.973644
spkts   0.965750  0.973644  1.000000
          dbytes     dloss     dpkts
dbytes  1.000000  0.997109  0.976419
dloss   0.997109  1.000000  0.981506
dpkts   0.976419  0.981506  1.000000
```

[244]:
```
sns.heatmap(train_data[['sbytes','sloss','spkts']].corr(), annot=True, cmap=plt.
 ↪cm.viridis)
plt.show()
sns.heatmap(train_data[['dbytes','dloss','dpkts']].corr(), annot=True, cmap=plt.
 ↪cm.viridis)
plt.show()
```





- so we can drop column sbyte and dbytes from above representation.
- From this visualization we can see that both have high correlation and same distribution with other columns

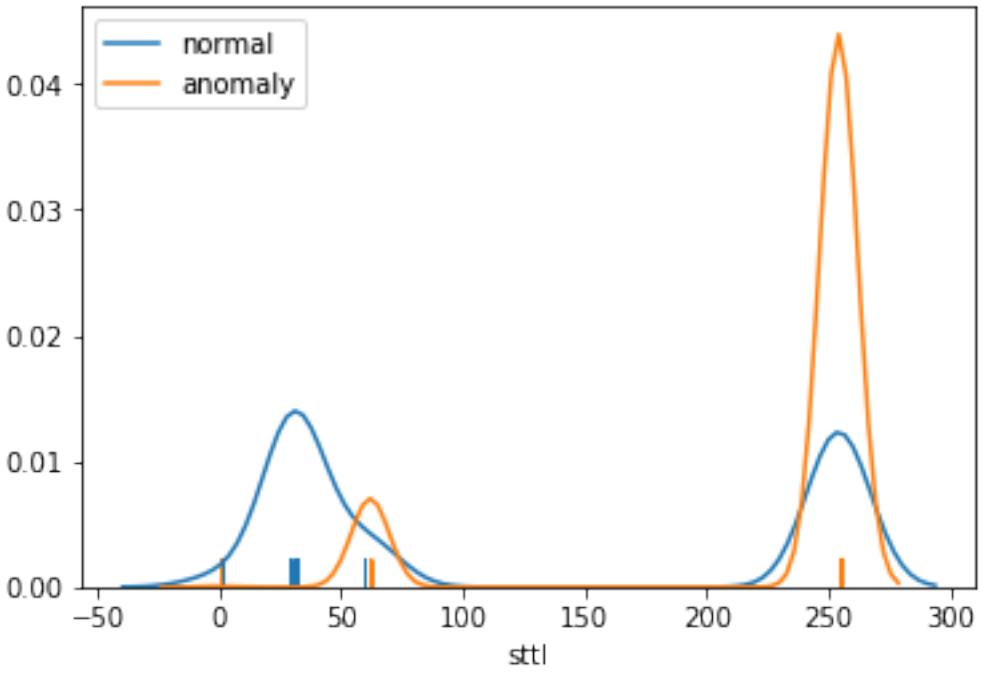
3.5.5 3.4.5 sttl - dttl

[211]:
```
data_features[9:11]
```

[211]:
```
     No.  Name    Type                                    Description
9     10  sttl   Integer  Source to destination time to live value
10    11  dttl   Integer   Destination to source time to live value
```

[216]:
```
sns.distplot(train_data[train_data['label']==0]['sttl'], label='normal',␣
 ↪hist=False, rug=True)
sns.distplot(train_data[train_data['label']==1]['sttl'], label='anomaly',␣
 ↪hist=False, rug=True)
plt.legend()
```

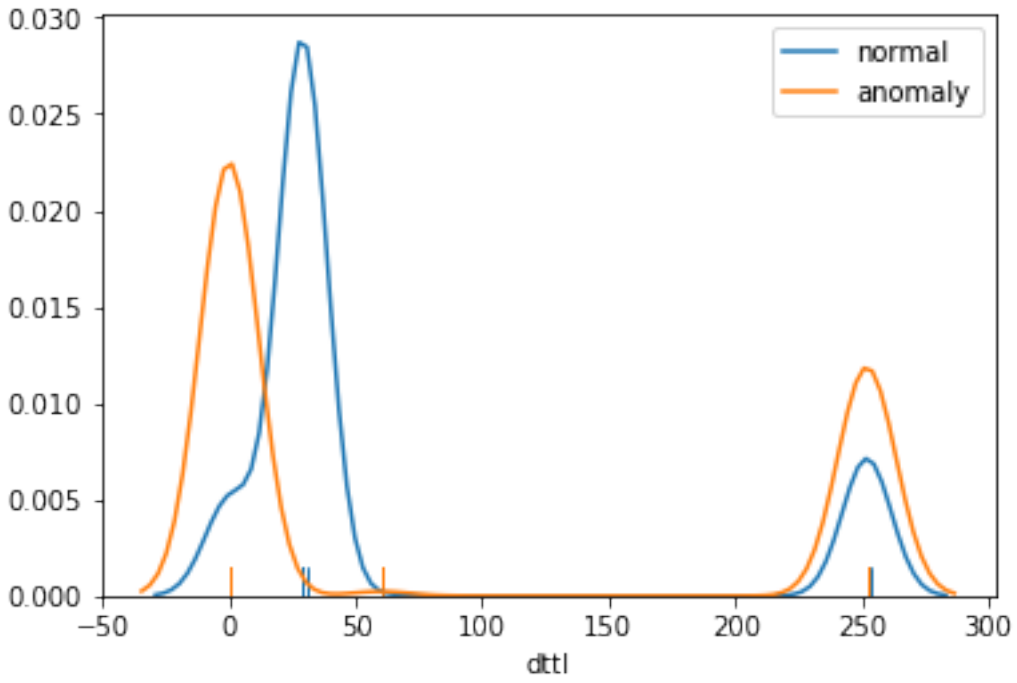
[216]:
```
<matplotlib.legend.Legend at 0x2058e5659b0>
```



- fewer amount of data have anomaly of higher rate. >0.2 are anomaly

[215]:
```
sns.distplot(train_data[train_data['label']==0]['dttl'], label='normal',␣
 ↪hist=False, rug=True)
sns.distplot(train_data[train_data['label']==1]['dttl'], label='anomaly',␣
 ↪hist=False, rug=True)
plt.legend()
```

[215]:
```
<matplotlib.legend.Legend at 0x21feb08c6d8>
```



- sttl and dttl have different distribution as we can see in overall numerical distribution plot. cant ignore column

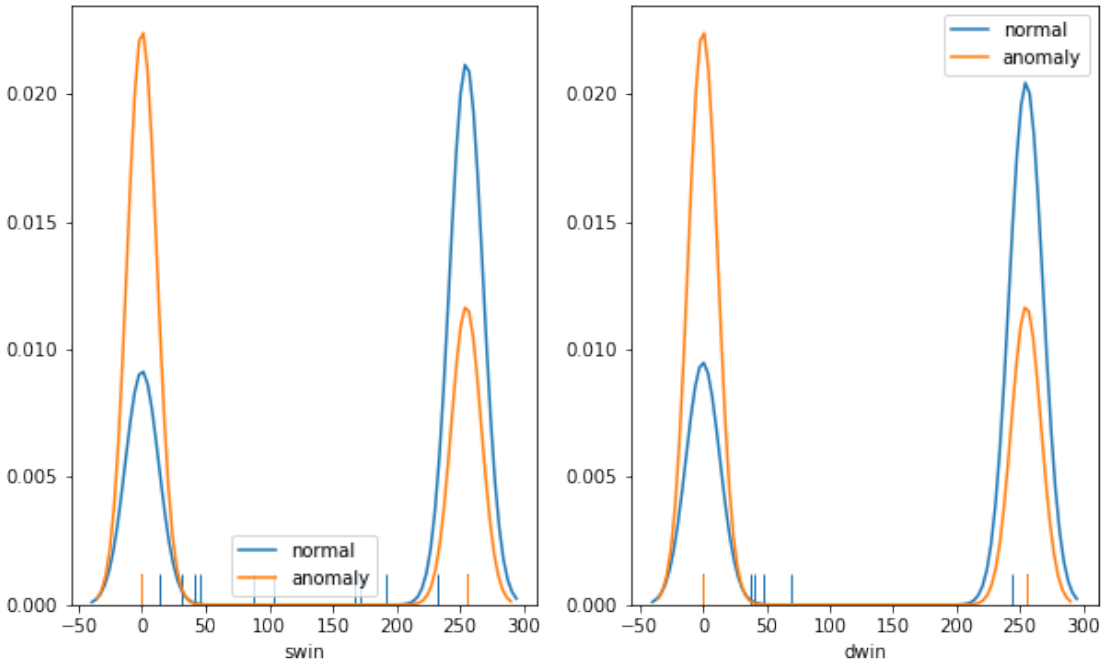
3.5.6 3.4.6 swin - dwin

[224]:
```
data_features[18:20]
```

[224]:

	No.	Name	Type	Description
18	19	swin	integer	Source TCP window advertisement value
19	20	dwin	integer	Destination TCP window advertisement value

[347]:
```
plt.figure(figsize=(10,6))
for i, col in enumerate(['swin', 'dwin']):
    plt.subplot(1,2,i+1)
    sns.distplot(train_data[train_data['label']==0][col], label='normal',␣
 ↪hist=False, rug=True)
    sns.distplot(train_data[train_data['label']==1][col], label='anomaly',␣
 ↪hist=False, rug=True)
    plt.legend()
plt.show()
```



- From above distribution plot we can distinguish between normal and anomaly.
- window rate of -50 to 50 less than 0.0075 is normal and 200 to 300 less than 0.0075 is anomaly

3.5.7 3.4.7 ct_dst_src_ltm, ct_srv_src, ct_srv_dst

[252]:
```
data_features[40:42]
```

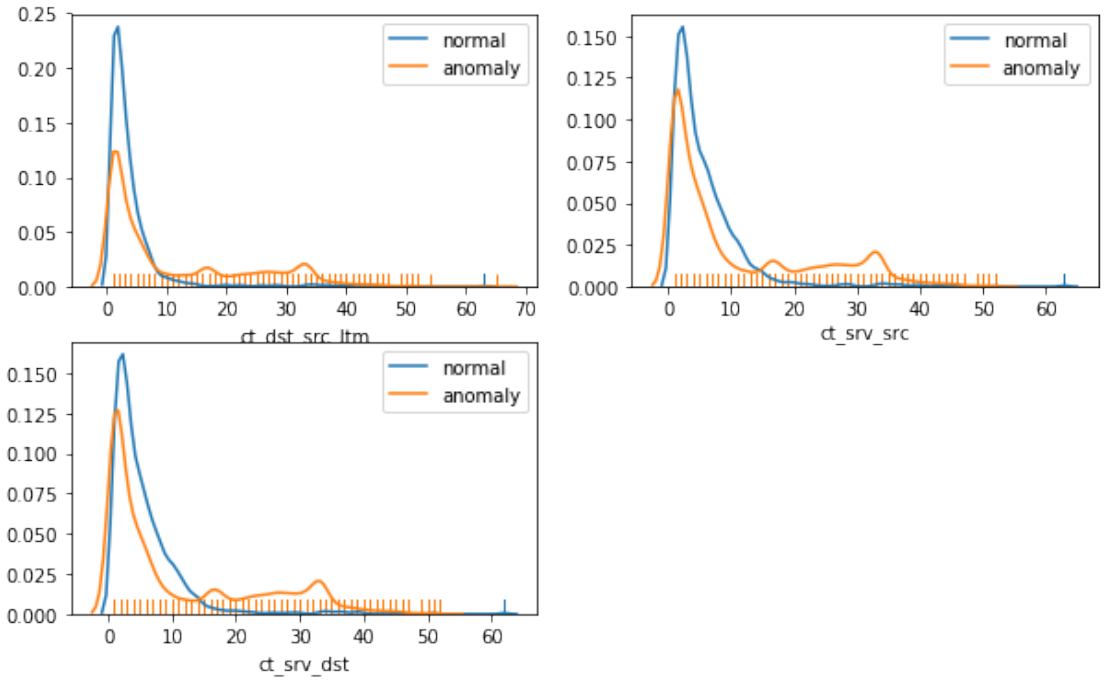
[252]:
```
    No.        Name    Type   \
40   41  ct_srv_src  integer
41   42  ct_srv_dst  integer

                                          Description
40  No. of connections that contain the same servi…
41  No. of connections that contain the same servi…
```

[254]:
```
list(data_features[40:42]['Description'])
```

[254]:
```
['No. of connections that contain the same service (14) and source address (1)
in 100 connections according to the last time (26).',
 'No. of connections that contain the same service (14) and destination address
(3) in 100 connections according to the last time (26).']
```

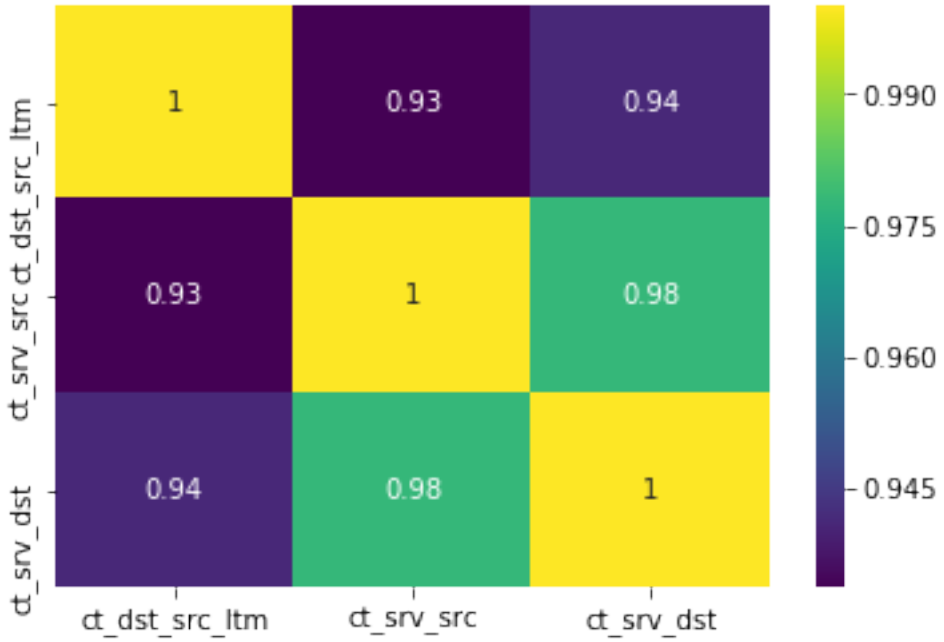
[342]:
```
plt.figure(figsize=(10,6))
for i, col in enumerate(['ct_dst_src_ltm', 'ct_srv_src', 'ct_srv_dst']):
    plt.subplot(2,2,i+1)
```

```
    sns.distplot(train_data[train_data['label']==0][col], label='normal',␣
 ↪hist=False, rug=True)
    sns.distplot(train_data[train_data['label']==1][col], label='anomaly',␣
 ↪hist=False, rug=True)
    plt.legend()
plt.show()
```



- As we can see the distribution are similar

[116]:
```
sns.heatmap(train_data[['ct_dst_src_ltm','ct_srv_src','ct_srv_dst']].corr(),␣
 ↪annot=True, cmap=plt.cm.viridis)
plt.show()
```



- we can drop column ct_srv_dst
- From this visualization we can see that three columns are having high correlation and same distribution

3.4.8 'is_ftp_login', 'ct_ftp_cmd'

[257]:
```
data_features[38:40]
```

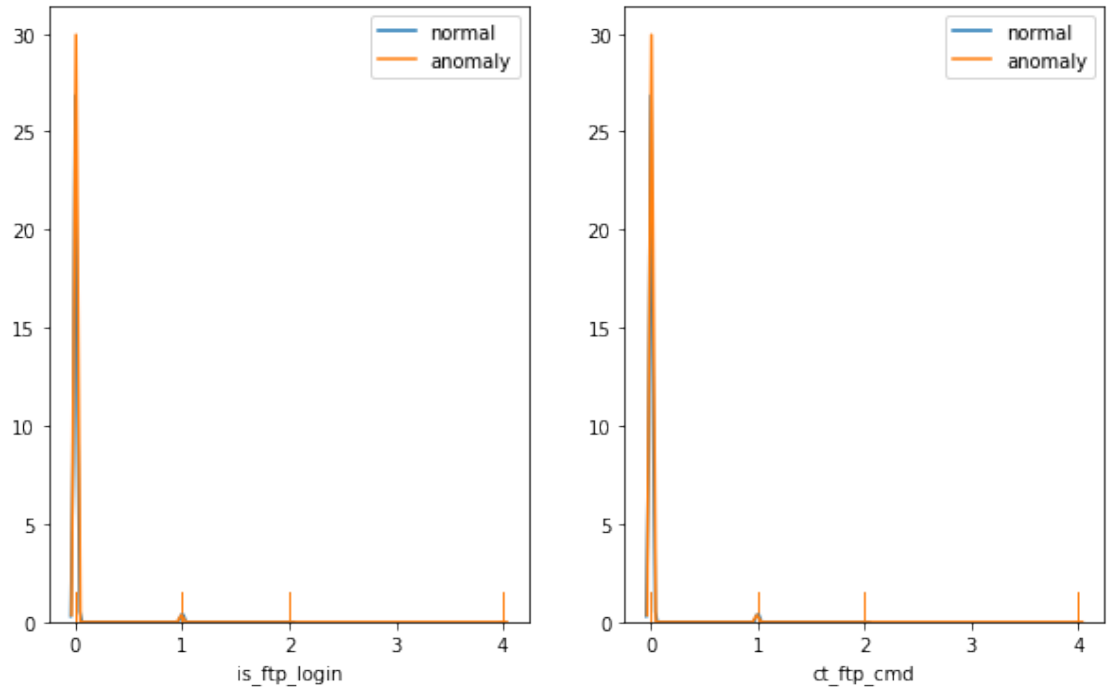
[257]:
```
    No.          Name     Type   \
38   39  is_ftp_login   Binary
39   40    ct_ftp_cmd  integer

                                          Description
38  If the ftp session is accessed by user and pas…
39     No of flows that has a command in ftp session.
```

[258]:
```
list(data_features[38:40]['Description'])
```

[258]:
```
['If the ftp session is accessed by user and password then 1 else 0. ',
 'No of flows that has a command in ftp session.']
```

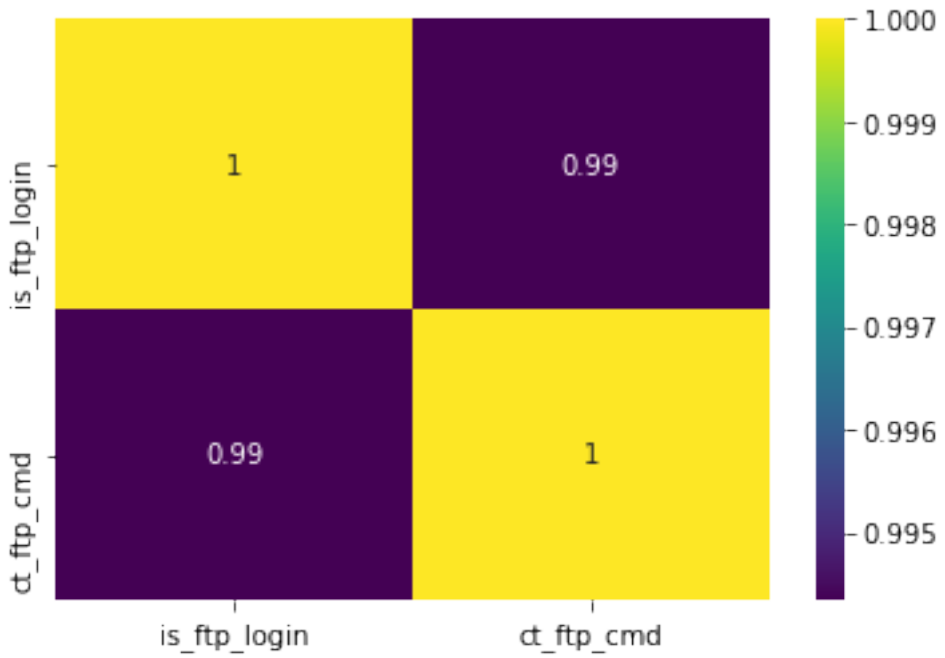
[341]:
```
plt.figure(figsize=(10,6))
for i, col in enumerate(['is_ftp_login', 'ct_ftp_cmd']):
    plt.subplot(1,2,i+1)
    sns.distplot(train_data[train_data['label']==0][col], label='normal',␣
 ↪hist=False, rug=True)
```

```
    sns.distplot(train_data[train_data['label']==1][col], label='anomaly',␣
 ↪hist=False, rug=True)
    plt.legend()
plt.show()
```



- As we can see the distribution are same and both the columns related to ftp session

[267]:
```
sns.heatmap(train_data[['is_ftp_login', 'ct_ftp_cmd']].corr(), annot=True,␣
 ↪cmap=plt.cm.viridis)
plt.show()
```



- we can drop ct_ftp_cmd.
- From this visualization we can see that both have high correlation and same distribution

3.6 3.5 EDA conclusion

we can drop the 5 columns mentioned from above analysis which are highly correlated

- sbyte and dbytes
- ct_srv_dst
- ct_ftp_cmd
- dwin

[3]:
```
train_data.drop(['sbytes', 'dbytes','ct_srv_dst', 'ct_ftp_cmd', 'dwin'],␣
↪axis=1, inplace=True)
train_data.shape
```

[3]:
```
(82332, 40)
```

4 4. ARM - Feature selection

4.1 4.1 Association Rule Mining

Association rule mining is a frequent pattern mining it is to determine how frequent an item is in total transaction. here Frequent item is a set of items which satisfies the minimum threshold value. threshold or metric for ARM. it is the support and confidence.

from our data set consider sbytes & dbytes two items. below is the rule.

sbytes => dbytes [support=3%, confidence- 70%]

The set of items sbytes & dbytes are called antecedent and consequent.

above state means that there is 3% that sbytes and dbytes are frequent together in total transaction. and there are 70% confidence level that sbytes and dbytes are occurred together.

4.1.1 4.1.1 Implementation Steps

4.2 |Steps|

Set the minimum threshold values

find all the subsets on the transaction using **apriori algorithm** having support of 30% or more.

Find all the item sets or rule of these subsets from step 2 which are having a higher confidence than minimum confidence and maximum rule length of 2.

get the columns from the rules using a set {} to eliminate the repeated columns.

use these columns as feature for machine learning models.

4.3 4.2 Data preprocessing

- For better understanding of dataset
- Identify the catogarical and numerical data and perform following encoding
  - Catogarical data (Label encoding)
  - Numerical data (StandardScalar)
- convert the data into numerical values so that it will input to machine learning models.

4.3.1 4.2.1 Catagorical Data

[4]:
```
cat_feature = train_data.select_dtypes(include=['category', object]).columns
cat_feature
```

[4]:
```
Index(['proto', 'service', 'state', 'attack_cat'], dtype='object')
```

[5]:
```
from sklearn.preprocessing import LabelEncoder
train_data[cat_feature] = train_data[cat_feature].apply(LabelEncoder().
 ↪fit_transform)
train_data.head()
```

[5]:

	id	dur	proto	service	state	spkts	dpkts	rate	sttl	dttl	\
0	1	0.000011	117	0	4	2	0	90909.0902	254	0	
1	2	0.000008	117	0	4	2	0	125000.0003	254	0	
2	3	0.000005	117	0	4	2	0	200000.0051	254	0	

```
3   4  0.000006    117        0      4      2      0  166666.6608   254     0
4   5  0.000010    117        0      4      2      0  100000.0025   254     0

   …  ct_dst_ltm  ct_src_dport_ltm  ct_dst_sport_ltm  ct_dst_src_ltm  \
0  …           1                 1                 1               2
1  …           1                 1                 1               2
2  …           1                 1                 1               3
3  …           2                 2                 1               3
4  …           2                 2                 1               3

   is_ftp_login  ct_flw_http_mthd  ct_src_ltm  is_sm_ips_ports  attack_cat  \
0             0                 0           1                0           6
1             0                 0           1                0           6
2             0                 0           1                0           6
3             0                 0           2                0           6
4             0                 0           2                0           6

   label
0      0
1      0
2      0
3      0
4      0

[5 rows x 40 columns]
```

[6]:
```
train_data.shape
```

[6]:
```
(82332, 40)
```

4.3.2 4.2.2 Split data into equal parts

- To reduce the time complexity, the data set are divided into equal parts ### Number dataset = Number / Number

[7]:
```
shuffled = train_data.sample(frac=1)
```

[8]:
```
shuffled.head()
```

[8]:
```
          id       dur  proto  service  state  spkts  dpkts           rate  \
2236    2237  0.000009    117        0      4      2      0  111111.107200
39952  39953  0.047272    111        0      3     72     74    3067.354852
36829  36830  0.001597    117        0      2      4      4    4383.218389
17530  17531  0.000005    117        2      4      2      0  200000.005100
78953  78954  1.168132    111        0      3     10      6      12.841015

       sttl  dttl  …  ct_dst_ltm  ct_src_dport_ltm  ct_dst_sport_ltm  \
```

```
2236    254      0  …          15                 15                  1
39952    31     29  …           5                  1                  1
36829    31     29  …           4                  1                  1
17530   254      0  …          18                 18                 18
78953   254    252  …           2                  3                  1

       ct_dst_src_ltm  is_ftp_login  ct_flw_http_mthd  ct_src_ltm  \
2236               39             0                 0          15
39952               1             0                 0           5
36829               3             0                 0           5
17530              34             0                 0          19
78953               3             0                 0           4

       is_sm_ips_ports  attack_cat  label
2236                 0           4      1
39952                0           6      0
36829                0           6      0
17530                0           5      1
78953                0           6      0

[5 rows x 40 columns]
```

[9]:
```
data_42 = np.array_split(shuffled, 42)
```

[10]:
```
len(data_42)
```

[10]: 42

4.3.3 4.2.3 Find Mode of the attribute

- Lets compute the mode for each attribute. it is the most frequent values of the attribute.
- For each data set attribute frequent values are identified, and most frequent value is set to true and remaining are false. task will be performed in both numerical and categorical data like below.
- It will accomplish the reliability of model output adopting to relevant attributes. ### Example:
  1. Numeric

  X = {1,2,3,1,3,1} => {1} => {1,0,0,1,0,1}
  2. Categorical

  X = {'INT', 'FIN', 'REQ', 'ACC', 'INT', 'REQ', 'INT'} => {'INT'} => {1, 0, 0, 0, 1, 0, 1}

[11]:
```
def create_arm_data(data):
    """ Create the binary mode for the data
    Find the most frequent data point in an attribute"""
    columns = data.columns
    for col in columns:
        #find mode of a attribute and make the model value 1 and others 0
```

```
        data[col] = np.where(data[col] == data[col].mode().values[0], 1, 0)
    return data
```

4.3.4 4.2.4 Create ARM rule based on Apriori Algorithm

[12]:
```
def create_arm_rule(result):
    """Create association rule for the given apriori data set """
    rules = association_rules(result, metric ="confidence", min_threshold = 1)
    # sort in order of confidence and lift
    rules = rules.sort_values(['confidence', 'lift'], ascending =[False, False])
    # find the length of antecedents & consequents
    rules["antecedent_len"] = rules["antecedents"].apply(lambda x: len(x))
    rules["consequents_len"] = rules["consequents"].apply(lambda x: len(x))
    rules_list_sorted = []
    # iterate each row to add the both antecedents & consequents in single␣
↪column set
    for x,y in rules.iterrows():
        rules_list_sorted.append(sorted(set(y.antecedents) | set(y.
↪consequents)))
    rules['rules_set_sorted'] = rules_list_sorted
    rules["rules_len"] = rules["rules_set_sorted"].apply(lambda x: len(x))
    #sort the set and make it list
    rules['rules_sorted'] = rules.rules_set_sorted.apply(lambda x: ','.
↪join(map(str, x)))
    return rules
```

4.4 4.2.5 Feature selction

- After the creation of rules in each set the antecedents, consequents are combined to generate the frequent itemsets on each dataset
- Each data set frequent items are again combined to make the final attributes for all dataset.
- Maximum rule length sets to two to eliminate the attribute which are not a frequent item. Least one item/attribute comes together with other attribute
- considering the items/attributes which are present more than 30% of total dataset/transaction.
- min_support = 0.3 that is 30% of items present in total transaction.

[13]:
```
col_ruled_sets = []
i=1
for part in data_42:
    """find columns of frequent transaction for all the dataset"""
    print("===Started dataset "+ str(i) +"====")
    #drop id and label
    part = part.drop(['id', 'label'], axis=1)
    print(part.shape)
    #create the binary mode data
    part_binary = create_arm_data(part)
```

```
    #Use apriori algorithm to find the sunsets of frequent item
    result = apriori(part_binary, min_support=0.3, use_colnames=True, max_len=2)
    #Create the rule from subsets
    arm_rules = create_arm_rule(result)
    final_columns = arm_rules['rules_sorted'].unique()
    col_final = set()
    #add each frequent columns to set
    for row in final_columns:
        for col in row.split(","):
            col_final.add(col)
    print(col_final)
    col_ruled_sets.append(col_final)
    print("===Completed dataset "+ str(i) +"====")
    i+=1
```

```
===Started dataset 1====
(1961, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 1====
===Started dataset 2====
(1961, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 2====
===Started dataset 3====
(1961, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 3====
===Started dataset 4====
(1961, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 4====
```

```
===Started dataset 5====
(1961, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 5====
===Started dataset 6====
(1961, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 6====
===Started dataset 7====
(1961, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 7====
===Started dataset 8====
(1961, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 8====
===Started dataset 9====
(1961, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 9====
===Started dataset 10====
(1961, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 10====
```

```
===Started dataset 11====
(1961, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 11====
===Started dataset 12====
(1961, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 12====
===Started dataset 13====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 13====
===Started dataset 14====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 14====
===Started dataset 15====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 15====
===Started dataset 16====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 16====
```

```
===Started dataset 17====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 17====
===Started dataset 18====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 18====
===Started dataset 19====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 19====
===Started dataset 20====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 20====
===Started dataset 21====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 21====
===Started dataset 22====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 22====
```

```
===Started dataset 23====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 23====
===Started dataset 24====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 24====
===Started dataset 25====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 25====
===Started dataset 26====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 26====
===Started dataset 27====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 27====
===Started dataset 28====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 28====
```

```
===Started dataset 29====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'dloss', 'proto', 'sloss',
'dmean', 'stcpb', 'dttl', 'dtcpb', 'ct_flw_http_mthd', 'is_sm_ips_ports',
'swin', 'dload', 'sjit', 'ct_dst_ltm', 'service', 'tcprtt', 'sttl', 'dinpkt',
'synack', 'djit', 'ct_dst_sport_ltm', 'response_body_len', 'ct_state_ttl',
'state', 'dpkts'}
===Completed dataset 29====
===Started dataset 30====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 30====
===Started dataset 31====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 31====
===Started dataset 32====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 32====
===Started dataset 33====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 33====
===Started dataset 34====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'dloss', 'proto', 'sloss',
'dmean', 'stcpb', 'dttl', 'dtcpb', 'ct_flw_http_mthd', 'is_sm_ips_ports',
'swin', 'dload', 'sjit', 'ct_dst_ltm', 'service', 'tcprtt', 'sttl', 'dinpkt',
'synack', 'djit', 'ct_dst_sport_ltm', 'response_body_len', 'ct_state_ttl',
'state', 'dpkts'}
===Completed dataset 34====
```

```
===Started dataset 35====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 35====
===Started dataset 36====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 36====
===Started dataset 37====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 37====
===Started dataset 38====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 38====
===Started dataset 39====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm',
'service', 'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 39====
===Started dataset 40====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'ct_src_dport_ltm', 'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb',
'ct_flw_http_mthd', 'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'service',
'tcprtt', 'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm',
'response_body_len', 'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 40====
```

```
===Started dataset 41====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'ct_dst_src_ltm', 'dloss',
'proto', 'sloss', 'dmean', 'stcpb', 'dttl', 'dtcpb', 'ct_flw_http_mthd',
'is_sm_ips_ports', 'swin', 'dload', 'sjit', 'ct_dst_ltm', 'service', 'tcprtt',
'sttl', 'dinpkt', 'synack', 'djit', 'ct_dst_sport_ltm', 'response_body_len',
'ct_state_ttl', 'state', 'dpkts'}
===Completed dataset 41====
===Started dataset 42====
(1960, 38)
{'trans_depth', 'is_ftp_login', 'spkts', 'ackdat', 'dloss', 'proto', 'sloss',
'dmean', 'stcpb', 'dttl', 'dtcpb', 'ct_flw_http_mthd', 'is_sm_ips_ports',
'swin', 'dload', 'sjit', 'ct_dst_ltm', 'service', 'tcprtt', 'sttl', 'dinpkt',
'synack', 'djit', 'ct_dst_sport_ltm', 'response_body_len', 'ct_state_ttl',
'state', 'dpkts'}
===Completed dataset 42====
```

iterate over all the 42 data set to find all possible columns

[14]:
```
#iterate over all the 42 data set to find all possibel columns
col_set = set()
for set_i in col_ruled_sets:
    for col in set_i:
        col_set.add(col)
print(len(col_set))
```

30

[15]:
```
col_set
```

[15]:
```
{'ackdat',
 'ct_dst_ltm',
 'ct_dst_sport_ltm',
 'ct_dst_src_ltm',
 'ct_flw_http_mthd',
 'ct_src_dport_ltm',
 'ct_state_ttl',
 'dinpkt',
 'djit',
 'dload',
 'dloss',
 'dmean',
 'dpkts',
 'dtcpb',
 'dttl',
 'is_ftp_login',
 'is_sm_ips_ports',
 'proto',
```

```
 'response_body_len',
 'service',
 'sjit',
 'sloss',
 'spkts',
 'state',
 'stcpb',
 'sttl',
 'swin',
 'synack',
 'tcprtt',
 'trans_depth'}
```

5 5. Machine Learning Models (Response Coding)

[16]:
```
from sklearn.calibration import CalibratedClassifierCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.metrics import log_loss, accuracy_score, f1_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn import preprocessing
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, Normalizer
from sklearn.ensemble import StackingClassifier
import warnings
warnings.filterwarnings('ignore')
```

5.1 Response Coding:

[17]:
```
def create_prime_df(x_data, y):
    d = {'state' : pd.Series(x_data), 'class' : pd.Series(y)}
    return pd.DataFrame(d)
```

[88]:
```
#generating response table
def get_response_df(s_u,p_df):
    data = []
    #iterate over unique values in state columns
    for u in tqdm(range(len(s_u))):
        class_0=0
        class_1=0
        #iterate over primary table
        for i in range(len(p_df)):
```

```
            s =p_df.loc[i, "state"]
            c =p_df.loc[i, "class"]
            #if state = unique value and class = 0 add 1 to class_0
            #else add 1 to class_1
            #summing all the values in primary table
            if s == s_u[u] and c == 0:
                class_0 += 1
            elif s == s_u[u] and c == 1:
                class_1 += 1
        #append [state,class0,class1] and return as dataframe
        data.append([s_u[u],class_0,class_1])
    return pd.DataFrame(data, columns=['state', 'class_0', 'class_1'])
```

[117]:
```
def encoded_data(input_df,res_df):
    data_e = []
    #iterate over response table
    #if state is present in input table the get the row
    #else 1/2 for class 0 and class 1
    for i in tqdm(range(len(input_df))):
        if input_df.loc[i, "state"] in res_df['state'].values:
            select_r = res_df.loc[res_df['state'] == input_df.loc[i, "state"]]
            c0 = select_r['class_0'].values[0]
            c1 = select_r['class_1'].values[0]
            #append the column in row as encoding rule
            data_e.append([int(c0)/int(c0+c1), int(c1)/int(c0+c1)])
        else:
            #append the column in row as encoding rule
            data_e.append([1/2, 1/2])
    return data_e
```

5.2 5.1 Reading Train and Test data

[20]:
```
train_data = pd.read_csv("data/UNSW_NB15_training-set.csv")
print(train_data.shape)
train_data.head()
```

```
(82332, 45)
```

[20]:

	id	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	\
0	1	0.000011	udp	-	INT	2	0	496	0	
1	2	0.000008	udp	-	INT	2	0	1762	0	
2	3	0.000005	udp	-	INT	2	0	1068	0	
3	4	0.000006	udp	-	INT	2	0	900	0	
4	5	0.000010	udp	-	INT	2	0	2126	0	

	rate	…	ct_dst_sport_ltm	ct_dst_src_ltm	is_ftp_login	\
0	90909.0902	…	1	2	0	

```
1  125000.0003  …                1               2             0
2  200000.0051  …                1               3             0
3  166666.6608  …                1               3             0
4  100000.0025  …                1               3             0

   ct_ftp_cmd  ct_flw_http_mthd  ct_src_ltm  ct_srv_dst  is_sm_ips_ports  \
0           0                 0           1           2                0
1           0                 0           1           2                0
2           0                 0           1           3                0
3           0                 0           2           3                0
4           0                 0           2           3                0

   attack_cat  label
0      Normal      0
1      Normal      0
2      Normal      0
3      Normal      0
4      Normal      0

[5 rows x 45 columns]
```

[21]:
```
df_train = train_data[list(col_set)]
df_train.head()
```

[21]:
```
   trans_depth  is_ftp_login  spkts  ct_dst_src_ltm  ackdat  dloss  \
0            0             0      2               2     0.0      0
1            0             0      2               2     0.0      0
2            0             0      2               3     0.0      0
3            0             0      2               3     0.0      0
4            0             0      2               3     0.0      0

   ct_src_dport_ltm proto  sloss  dmean  …  tcprtt  sttl  dinpkt  synack  \
0                 1   udp      0      0  …     0.0   254     0.0     0.0
1                 1   udp      0      0  …     0.0   254     0.0     0.0
2                 1   udp      0      0  …     0.0   254     0.0     0.0
3                 2   udp      0      0  …     0.0   254     0.0     0.0
4                 2   udp      0      0  …     0.0   254     0.0     0.0

   djit  ct_dst_sport_ltm  response_body_len  ct_state_ttl  state dpkts
0   0.0                 1                  0             2    INT     0
1   0.0                 1                  0             2    INT     0
2   0.0                 1                  0             2    INT     0
3   0.0                 1                  0             2    INT     0
4   0.0                 1                  0             2    INT     0

[5 rows x 30 columns]
```

[22]:
```
cat_features = df_train.select_dtypes(include=['category', object]).columns
cat_features
```

[22]:
```
Index(['proto', 'service', 'state'], dtype='object')
```

[23]:
```
test_data = pd.read_csv("data/UNSW_NB15_testing-set.csv")
print(test_data.shape)
test_data.head()
```

```
(175341, 45)
```

[23]:

	id	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	rate	\
0	1	0.121478	tcp	-	FIN	6	4	258	172	74.087490	
1	2	0.649902	tcp	-	FIN	14	38	734	42014	78.473372	
2	3	1.623129	tcp	-	FIN	8	16	364	13186	14.170161	
3	4	1.681642	tcp	ftp	FIN	12	12	628	770	13.677108	
4	5	0.449454	tcp	-	FIN	10	6	534	268	33.373826	

	…	ct_dst_sport_ltm	ct_dst_src_ltm	is_ftp_login	ct_ftp_cmd	\
0	…	1	1	0	0	
1	…	1	2	0	0	
2	…	1	3	0	0	
3	…	1	3	1	1	
4	…	1	40	0	0	

	ct_flw_http_mthd	ct_src_ltm	ct_srv_dst	is_sm_ips_ports	attack_cat	\
0	0	1	1	0	Normal	
1	0	1	6	0	Normal	
2	0	2	6	0	Normal	
3	0	2	1	0	Normal	
4	0	2	39	0	Normal	

	label
0	0
1	0
2	0
3	0
4	0

```
[5 rows x 45 columns]
```

[24]:
```
df_test = test_data[list(col_set)]
df_test.head()
```

[24]:

	trans_depth	is_ftp_login	spkts	ct_dst_src_ltm	ackdat	dloss	\
0	0	0	6	1	0.000000	0	
1	0	0	14	2	0.000000	17	

```
2            0            0      8              3  0.050439      6
3            0            1     12              3  0.000000      3
4            0            0     10             40  0.057234      1

   ct_src_dport_ltm proto  sloss  dmean  …    tcprtt  sttl      dinpkt  \
0                 1   tcp      0     43  …  0.000000   252    8.375000
1                 1   tcp      2   1106  …  0.000000    62   15.432865
2                 1   tcp      1    824  …  0.111897    62  102.737203
3                 1   tcp      1     64  …  0.000000    62   90.235726
4                 2   tcp      2     45  …  0.128381   254   75.659602

     synack          djit  ct_dst_sport_ltm  response_body_len  ct_state_ttl  \
0  0.000000     11.830604                 1                  0             0
1  0.000000   1387.778330                 1                  0             1
2  0.061458  11420.926230                 1                  0             1
3  0.000000   4991.784669                 1                  0             1
4  0.071147    115.807000                 1                  0             1

   state dpkts
0    FIN     4
1    FIN    38
2    FIN    16
3    FIN    12
4    FIN     6

[5 rows x 30 columns]
```

[25]:
```
cat_feature = df_test.select_dtypes(include=['category', object]).columns
cat_feature
```

[25]:
```
Index(['proto', 'service', 'state'], dtype='object')
```

5.2.1 Proto

[89]:
```
prime_train_s_df = create_prime_df(df_train['proto'].values,␣
 ↪train_data['label'])
prime_test_s_df = create_prime_df(df_test['proto'].values, test_data['label'])
response_df = get_response_df(df_train['proto'].unique(), prime_train_s_df)
response_df.head()
```

```
100%|
   | 131/131 [04:12<00:00,  1.93s/it]
```

[89]:
```
   state  class_0  class_1
0    udp     8097    21321
1    arp      987        0
2    tcp    27848    15247
```

```
3  igmp       30        0
4  ospf       38      638
```

[118]:
```
x_train_proto = pd.DataFrame(encoded_data(prime_train_s_df,response_df),␣
↪columns=['state_0', 'state_1'])
x_test_proto = pd.DataFrame(encoded_data(prime_test_s_df,response_df),␣
↪columns=['state_0', 'state_1'])
```

```
100%|
| 82332/82332 [01:39<00:00, 830.66it/s]
100%|                                          |
175341/175341 [03:30<00:00, 831.21it/s]
```

5.2.2 service

[119]:
```
prime_train_ser_df = create_prime_df(df_train['service'].values,␣
↪train_data['label'])
prime_test_ser_df = create_prime_df(df_test['service'].values,␣
↪test_data['label'])
response_df_ser = get_response_df(df_train['service'].unique(),␣
↪prime_train_ser_df)
response_df_ser.head()
```

```
100%|
    | 13/13 [00:24<00:00,  1.86s/it]
```

[119]:

	state	class_0	class_1
0	-	27375	19778
1	http	4013	4274
2	ftp	758	794
3	ftp-data	949	447
4	smtp	635	1216

[120]:
```
x_train_ser = pd.DataFrame(encoded_data(prime_train_ser_df,response_df_ser),␣
↪columns=['state_0', 'state_1'])
x_test_ser = pd.DataFrame(encoded_data(prime_test_ser_df,response_df_ser),␣
↪columns=['state_0', 'state_1'])
```

```
100%|
| 82332/82332 [01:46<00:00, 770.90it/s]
100%|                                          |
175341/175341 [03:45<00:00, 776.21it/s]
```

5.2.3 state

[84]:
```
prime_train_st_df = create_prime_df(df_train['state'].values,␣
↪train_data['label'])
prime_test_st_df = create_prime_df(df_test['state'].values, test_data['label'])
response_df_st = get_response_df(df_train['state'].unique(), prime_train_st_df)
response_df_st.head()
```

[84]:

	state	class_0	class_1
0	INT	4485	29678
1	FIN	24172	15167
2	REQ	1707	135
3	ACC	2	2
4	CON	6633	349

[85]:
```
x_train_state = pd.DataFrame(encoded_data(prime_train_st_df,response_df_st),␣
↪columns=['state_0', 'state_1'])
x_test_state = pd.DataFrame(encoded_data(prime_test_st_df,response_df_st),␣
↪columns=['state_0', 'state_1'])
```

[122]:
```
cat_df_train = pd.concat([x_train_proto, x_train_ser,x_train_state], axis=1,␣
↪sort=False)
cat_df_train.head()
```

[122]:

	state_0	state_1	state_0	state_1	state_0	state_1
0	0.27524	0.72476	0.580557	0.419443	0.131282	0.868718
1	0.27524	0.72476	0.580557	0.419443	0.131282	0.868718
2	0.27524	0.72476	0.580557	0.419443	0.131282	0.868718
3	0.27524	0.72476	0.580557	0.419443	0.131282	0.868718
4	0.27524	0.72476	0.580557	0.419443	0.131282	0.868718

[123]:
```
df_train = df_train.drop(cat_features, axis=1)
df_train.shape
```

[123]: (82332, 27)

[124]:
```
df_train = df_train.join(cat_df_train)
df_train.head()
```

[124]:

	trans_depth	is_ftp_login	spkts	ct_dst_src_ltm	ackdat	dloss	\
0	0	0	2	2	0.0	0	
1	0	0	2	2	0.0	0	
2	0	0	2	3	0.0	0	
3	0	0	2	3	0.0	0	
4	0	0	2	3	0.0	0	

	ct_src_dport_ltm	sloss	dmean	stcpb	…	ct_dst_sport_ltm	\

```
0                 1      0      0      0  …                 1
1                 1      0      0      0  …                 1
2                 1      0      0      0  …                 1
3                 2      0      0      0  …                 1
4                 2      0      0      0  …                 1

   response_body_len  ct_state_ttl  dpkts  state_0  state_1   state_0  \
0                  0             2      0  0.27524  0.72476  0.580557
1                  0             2      0  0.27524  0.72476  0.580557
2                  0             2      0  0.27524  0.72476  0.580557
3                  0             2      0  0.27524  0.72476  0.580557
4                  0             2      0  0.27524  0.72476  0.580557

    state_1   state_0   state_1
0  0.419443  0.131282  0.868718
1  0.419443  0.131282  0.868718
2  0.419443  0.131282  0.868718
3  0.419443  0.131282  0.868718
4  0.419443  0.131282  0.868718

[5 rows x 33 columns]
```

[125]:
```
cat_df_test = pd.concat([x_test_proto, x_test_ser,x_test_state], axis=1,␣
↪sort=False)
cat_df_test.head()
```

[125]:
```
   state_0  state_1   state_0   state_1   state_0   state_1
0   0.6462   0.3538  0.580557  0.419443  0.614454  0.385546
1   0.6462   0.3538  0.580557  0.419443  0.614454  0.385546
2   0.6462   0.3538  0.580557  0.419443  0.614454  0.385546
3   0.6462   0.3538  0.488402  0.511598  0.614454  0.385546
4   0.6462   0.3538  0.580557  0.419443  0.614454  0.385546
```

[126]:
```
df_test = df_test.drop(cat_feature, axis=1)
df_test.shape
```

[126]:
```
(175341, 27)
```

[127]:
```
df_test = df_test.join(cat_df_test)
df_test.head()
```

[127]:
```
   trans_depth  is_ftp_login  spkts  ct_dst_src_ltm    ackdat  dloss  \
0            0             0      6               1  0.000000      0
1            0             0     14               2  0.000000     17
2            0             0      8               3  0.050439      6
3            0             1     12               3  0.000000      3
4            0             0     10              40  0.057234      1
```

```
   ct_src_dport_ltm  sloss  dmean       stcpb  …  ct_dst_sport_ltm  \
0                 1      0     43   621772692  …                 1
1                 1      2   1106  1417884146  …                 1
2                 1      1    824  2116150707  …                 1
3                 1      1     64  1107119177  …                 1
4                 2      2     45  2436137549  …                 1

   response_body_len  ct_state_ttl  dpkts  state_0  state_1   state_0  \
0                  0             0      4   0.6462   0.3538  0.580557
1                  0             1     38   0.6462   0.3538  0.580557
2                  0             1     16   0.6462   0.3538  0.580557
3                  0             1     12   0.6462   0.3538  0.488402
4                  0             1      6   0.6462   0.3538  0.580557

    state_1   state_0   state_1
0  0.419443  0.614454  0.385546
1  0.419443  0.614454  0.385546
2  0.419443  0.614454  0.385546
3  0.511598  0.614454  0.385546
4  0.419443  0.614454  0.385546

[5 rows x 33 columns]
```

5.2.4 5.1.1 Standardize the data

[128]:
```
x = df_train.values
x_test = df_test.values
std_scaler = preprocessing.MinMaxScaler()
std_scaler.fit(x)
x_scaled = std_scaler.transform(x)
df_train = pd.DataFrame(x_scaled)
x_scaled_test = std_scaler.transform(x_test)
df_test = pd.DataFrame(x_scaled_test)
```

[129]:
```
df_train.head()
```

[129]:
```
     0    1         2         3    4    5         6    7    8    9   …   23  \
0  0.0  0.0  0.000094  0.016129  0.0  0.0  0.000000  0.0  0.0  0.0  …  0.0
1  0.0  0.0  0.000094  0.016129  0.0  0.0  0.000000  0.0  0.0  0.0  …  0.0
2  0.0  0.0  0.000094  0.032258  0.0  0.0  0.000000  0.0  0.0  0.0  …  0.0
3  0.0  0.0  0.000094  0.032258  0.0  0.0  0.017241  0.0  0.0  0.0  …  0.0
4  0.0  0.0  0.000094  0.032258  0.0  0.0  0.017241  0.0  0.0  0.0  …  0.0

     24        25   26       27       28        29        30        31  \
0  0.0  0.333333  0.0  0.27524  0.72476  0.592168  0.407832  0.131282
1  0.0  0.333333  0.0  0.27524  0.72476  0.592168  0.407832  0.131282
```

```
2  0.0  0.333333  0.0  0.27524  0.72476  0.592168  0.407832  0.131282
3  0.0  0.333333  0.0  0.27524  0.72476  0.592168  0.407832  0.131282
4  0.0  0.333333  0.0  0.27524  0.72476  0.592168  0.407832  0.131282

         32
0  0.868718
1  0.868718
2  0.868718
3  0.868718
4  0.868718

[5 rows x 33 columns]
```

[130]:
```
y_train = train_data['label']
y_test = test_data['label']
print("train data shape", df_train.shape, y_train.shape)
print("test data shape", df_test.shape, y_test.shape)
```

```
train data shape (82332, 33) (82332,)
test data shape (175341, 33) (175341,)
```

5.3 5.2 Logistic Regression Model

[132]:
```
prams={
    'alpha':[10 ** x for x in range(-4, 1)],
     'max_iter':[5, 10, 20, 50, 100],
    'eta0': [10 ** x for x in range(-4, 1)]
}
lr_cfl=GridSearchCV(SGDClassifier(penalty='l2', loss='log', n_jobs = -1),␣
 ↪param_grid=prams,verbose=10,n_jobs=-1)
lr_cfl.fit(df_train,y_train)
```

```
Fitting 5 folds for each of 125 candidates, totalling 625 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    5.7s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:    6.7s
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:    7.6s
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    8.4s
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    9.4s
[Parallel(n_jobs=-1)]: Done  45 tasks      | elapsed:   10.3s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:   11.6s
[Parallel(n_jobs=-1)]: Done  69 tasks      | elapsed:   12.8s
[Parallel(n_jobs=-1)]: Done  82 tasks      | elapsed:   14.2s
[Parallel(n_jobs=-1)]: Done  97 tasks      | elapsed:   15.9s
[Parallel(n_jobs=-1)]: Done 112 tasks      | elapsed:   17.4s
[Parallel(n_jobs=-1)]: Done 129 tasks      | elapsed:   19.1s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:   20.8s
```

```
[Parallel(n_jobs=-1)]: Done 165 tasks      | elapsed:   22.5s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   24.4s
[Parallel(n_jobs=-1)]: Done 205 tasks      | elapsed:   26.2s
[Parallel(n_jobs=-1)]: Done 226 tasks      | elapsed:   27.9s
[Parallel(n_jobs=-1)]: Done 249 tasks      | elapsed:   30.1s
[Parallel(n_jobs=-1)]: Done 272 tasks      | elapsed:   31.9s
[Parallel(n_jobs=-1)]: Done 297 tasks      | elapsed:   33.9s
[Parallel(n_jobs=-1)]: Done 322 tasks      | elapsed:   36.1s
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:   38.5s
[Parallel(n_jobs=-1)]: Done 376 tasks      | elapsed:   40.6s
[Parallel(n_jobs=-1)]: Done 405 tasks      | elapsed:   43.2s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   45.2s
[Parallel(n_jobs=-1)]: Done 465 tasks      | elapsed:   47.6s
[Parallel(n_jobs=-1)]: Done 496 tasks      | elapsed:   50.1s
[Parallel(n_jobs=-1)]: Done 529 tasks      | elapsed:   53.1s
[Parallel(n_jobs=-1)]: Done 562 tasks      | elapsed:   55.9s
[Parallel(n_jobs=-1)]: Done 597 tasks      | elapsed:   59.6s
[Parallel(n_jobs=-1)]: Done 625 out of 625 | elapsed:  1.0min finished
```

[132]:
```
GridSearchCV(cv=None, error_score=nan,
             estimator=SGDClassifier(alpha=0.0001, average=False,
                                     class_weight=None, early_stopping=False,
                                     epsilon=0.1, eta0=0.0, fit_intercept=True,
                                     l1_ratio=0.15, learning_rate='optimal',
                                     loss='log', max_iter=1000,
                                     n_iter_no_change=5, n_jobs=-1,
                                     penalty='l2', power_t=0.5,
                                     random_state=None, shuffle=True, tol=0.001,
                                     validation_fraction=0.1, verbose=0,
                                     warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1],
                         'eta0': [0.0001, 0.001, 0.01, 0.1, 1],
                         'max_iter': [5, 10, 20, 50, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=10)
```

[133]:
```
results = pd.DataFrame.from_dict(lr_cfl.cv_results_)
results = results.sort_values(['rank_test_score'])
results.head()
```

[133]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	\
13	0.733896	0.125073	0.012087	0.001493	0.0001	
2	0.771722	0.084044	0.009397	0.001724	0.0001	
6	0.738520	0.055430	0.011809	0.003169	0.0001	
12	0.712953	0.036958	0.016193	0.010511	0.0001	
15	0.495005	0.021378	0.013800	0.004314	0.0001	

```
   param_eta0 param_max_iter  \
13       0.01             50
2      0.0001             20
6       0.001             10
12       0.01             20
15        0.1              5

                                              params  split0_test_score  \
13    {'alpha': 0.0001, 'eta0': 0.01, 'max_iter': 50}           0.939698
2   {'alpha': 0.0001, 'eta0': 0.0001, 'max_iter': 20}           0.901561
6    {'alpha': 0.0001, 'eta0': 0.001, 'max_iter': 10}           0.897431
12    {'alpha': 0.0001, 'eta0': 0.01, 'max_iter': 20}           0.918807
15      {'alpha': 0.0001, 'eta0': 0.1, 'max_iter': 5}           0.912613

    split1_test_score  split2_test_score  split3_test_score  \
13           0.979474           0.858253           0.828191
2            0.972308           0.869489           0.814345
6            0.963928           0.868213           0.819568
12           0.962956           0.859225           0.796793
15           0.944738           0.870339           0.809790

    split4_test_score  mean_test_score  std_test_score  rank_test_score
13           0.785437         0.878211        0.071473                1
2            0.800255         0.871591        0.062310                2
6            0.799648         0.869758        0.058431                3
12           0.806450         0.868846        0.064079                4
15           0.789445         0.865385        0.059008                5
```

[134]:
```
print(lr_cfl.best_params_)
```

```
{'alpha': 0.0001, 'eta0': 0.01, 'max_iter': 50}
```

[135]:
```
logisticR=SGDClassifier(alpha=lr_cfl.best_params_['alpha'],eta0=lr_cfl.
 ↪best_params_['eta0'], penalty='l2', loss='log', n_jobs = -1, max_iter=lr_cfl.
 ↪best_params_['max_iter'])
logisticR.fit(df_train,y_train)
sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
sig_clf.fit(df_train, y_train)
predict_y_tr_lr = sig_clf.predict(df_train)
predict_y_te_lr = sig_clf.predict(df_test)
lr_f1 = f1_score(y_test, predict_y_te_lr)
print(lr_f1)
```

```
0.9423032118488934
```

[136]:
```
cm_lr = confusion_matrix(y_test, predict_y_te_lr)
```

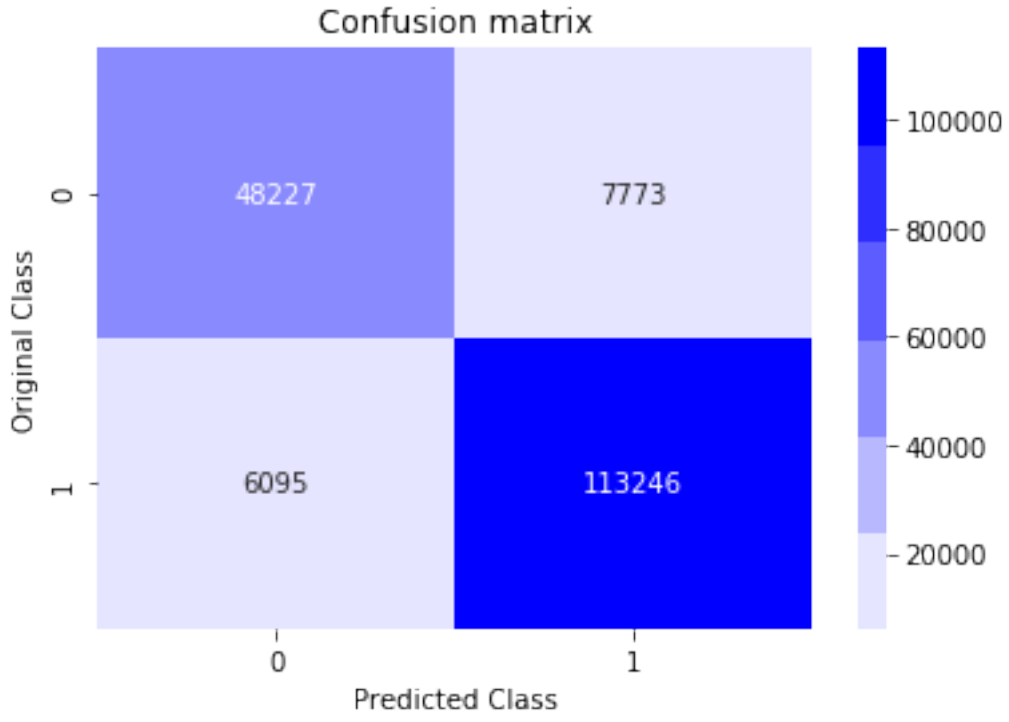
[137]:
```
tn, fp, fn, tp = cm_lr.ravel()
```

[138]:
```
fpr_lr = (fp/(fp+tn))*100
fnr_lr = (fn/(fn+tp))*100
far_lr = (fpr_lr+fnr_lr)/2
print("FAR:",far_lr)
```

```
FAR: 9.493785462605953
```

[139]:
```
def plot_cm(cm):
    sns.heatmap(cm, annot=True, cmap=sns.light_palette("blue"), fmt="g")
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")
    plt.show()
```

[140]:
```
plot_cm(cm_lr)
```

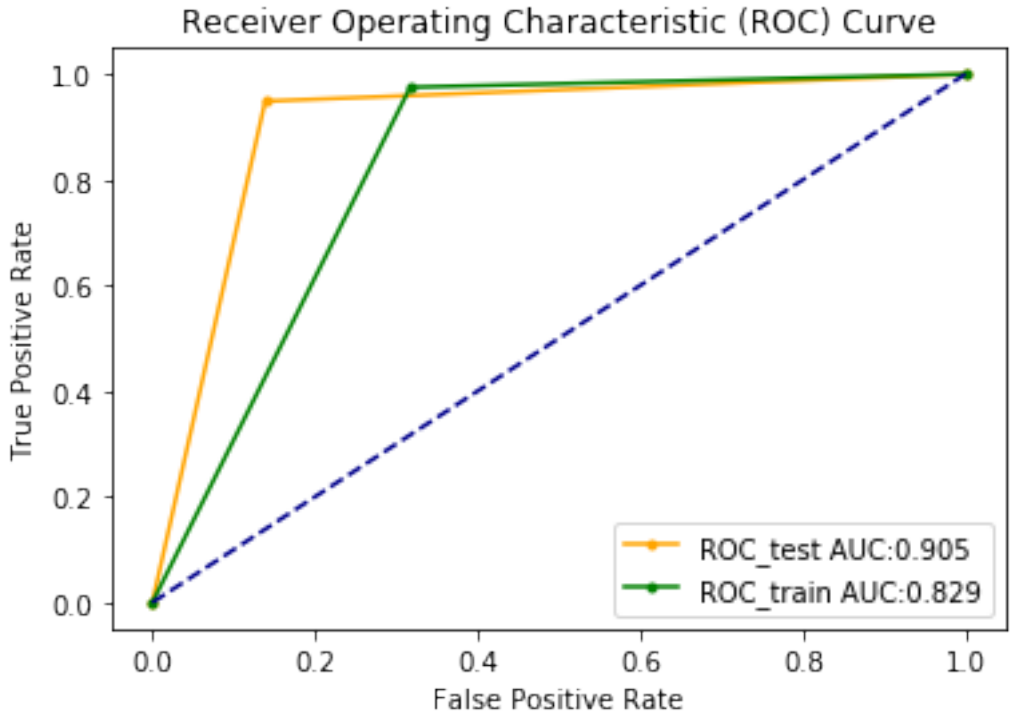


[141]:
```
from sklearn.metrics import roc_curve, auc
def plot_roc_curve(fpr_tr, tpr_tr,fpr_te, tpr_te):
    '''
    plot the ROC curve for the FPR and TPR value
    '''
```

```
    plt.plot(fpr_te, tpr_te, 'k.-', color='orange', label='ROC_test AUC:%.3f'%␣
↪auc(fpr_te, tpr_te))
    plt.plot(fpr_tr, tpr_tr, 'k.-', color='green', label='ROC_train AUC:%.3f'%␣
↪auc(fpr_tr, tpr_tr))
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()
```

[142]:
```
#finding the FPR and TPR for logistic reg model set
fpr_te_lr, tpr_te_lr, t_te_lr = roc_curve(y_test, predict_y_te_lr)
fpr_tr_lr, tpr_tr_lr, t_tr_lr = roc_curve(y_train, predict_y_tr_lr)
auc_te_lr = auc(fpr_te_lr, tpr_te_lr)
print("AUC_LR: ",auc_te_lr)
plot_roc_curve(fpr_tr_lr,tpr_tr_lr,fpr_te_lr, tpr_te_lr)
```

```
AUC_LR:  0.9050621453739406
```



5.4 5.3 Support Vector Machine Model

[143]:
```
prams={
    'alpha':[10 ** x for x in range(-4, 1)],
     'max_iter':[5, 10, 20, 50, 100],
    'eta0': [10 ** x for x in range(-4, 1)]
}
svm_cfl=GridSearchCV(SGDClassifier(penalty='l2', loss='hinge', n_jobs = -1),␣
 ↪param_grid=prams,verbose=10,n_jobs=-1)
svm_cfl.fit(df_train,y_train)
```

```
Fitting 5 folds for each of 125 candidates, totalling 625 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:    1.0s
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:    1.9s
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    2.7s
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    3.5s
[Parallel(n_jobs=-1)]: Done  45 tasks      | elapsed:    4.8s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:    5.8s
[Parallel(n_jobs=-1)]: Done  69 tasks      | elapsed:    7.1s
[Parallel(n_jobs=-1)]: Done  82 tasks      | elapsed:    8.3s
[Parallel(n_jobs=-1)]: Done  97 tasks      | elapsed:   10.0s
[Parallel(n_jobs=-1)]: Done 112 tasks      | elapsed:   11.4s
[Parallel(n_jobs=-1)]: Done 129 tasks      | elapsed:   13.0s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:   14.4s
[Parallel(n_jobs=-1)]: Done 165 tasks      | elapsed:   15.7s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   17.1s
[Parallel(n_jobs=-1)]: Done 205 tasks      | elapsed:   18.7s
[Parallel(n_jobs=-1)]: Done 226 tasks      | elapsed:   20.0s
[Parallel(n_jobs=-1)]: Done 249 tasks      | elapsed:   22.1s
[Parallel(n_jobs=-1)]: Done 272 tasks      | elapsed:   23.9s
[Parallel(n_jobs=-1)]: Done 297 tasks      | elapsed:   26.2s
[Parallel(n_jobs=-1)]: Done 322 tasks      | elapsed:   28.1s
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:   30.1s
[Parallel(n_jobs=-1)]: Done 376 tasks      | elapsed:   31.9s
[Parallel(n_jobs=-1)]: Done 405 tasks      | elapsed:   33.8s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   35.7s
[Parallel(n_jobs=-1)]: Done 465 tasks      | elapsed:   37.8s
[Parallel(n_jobs=-1)]: Done 496 tasks      | elapsed:   39.9s
[Parallel(n_jobs=-1)]: Done 529 tasks      | elapsed:   41.9s
[Parallel(n_jobs=-1)]: Done 562 tasks      | elapsed:   44.3s
[Parallel(n_jobs=-1)]: Done 597 tasks      | elapsed:   46.7s
[Parallel(n_jobs=-1)]: Done 625 out of 625 | elapsed:   48.5s finished
```

[143]:
```
GridSearchCV(cv=None, error_score=nan,
             estimator=SGDClassifier(alpha=0.0001, average=False,
```

```
                                   class_weight=None, early_stopping=False,
                                   epsilon=0.1, eta0=0.0, fit_intercept=True,
                                   l1_ratio=0.15, learning_rate='optimal',
                                   loss='hinge', max_iter=1000,
                                   n_iter_no_change=5, n_jobs=-1,
                                   penalty='l2', power_t=0.5,
                                   random_state=None, shuffle=True, tol=0.001,
                                   validation_fraction=0.1, verbose=0,
                                   warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1],
                         'eta0': [0.0001, 0.001, 0.01, 0.1, 1],
                         'max_iter': [5, 10, 20, 50, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=10)
```

[144]:
```
print(svm_cfl.best_params_)
```

```
{'alpha': 0.0001, 'eta0': 1, 'max_iter': 10}
```

[145]:
```
svm=SGDClassifier(alpha=svm_cfl.best_params_['alpha'],eta0=svm_cfl.
 ↪best_params_['eta0'], penalty='l2', loss='hinge', n_jobs = -1,␣
 ↪max_iter=svm_cfl.best_params_['max_iter'])
svm.fit(df_train,y_train)
sig_clf_svm = CalibratedClassifierCV(svm, method="sigmoid")
sig_clf_svm.fit(df_train, y_train)
predict_y_tr_svm = sig_clf.predict(df_train)
predict_y_te_svm = sig_clf_svm.predict(df_test)
svm_f1 = f1_score(y_test, predict_y_te_svm)
print("F1-Score", svm_f1)
```

```
F1-Score 0.9197150144204811
```

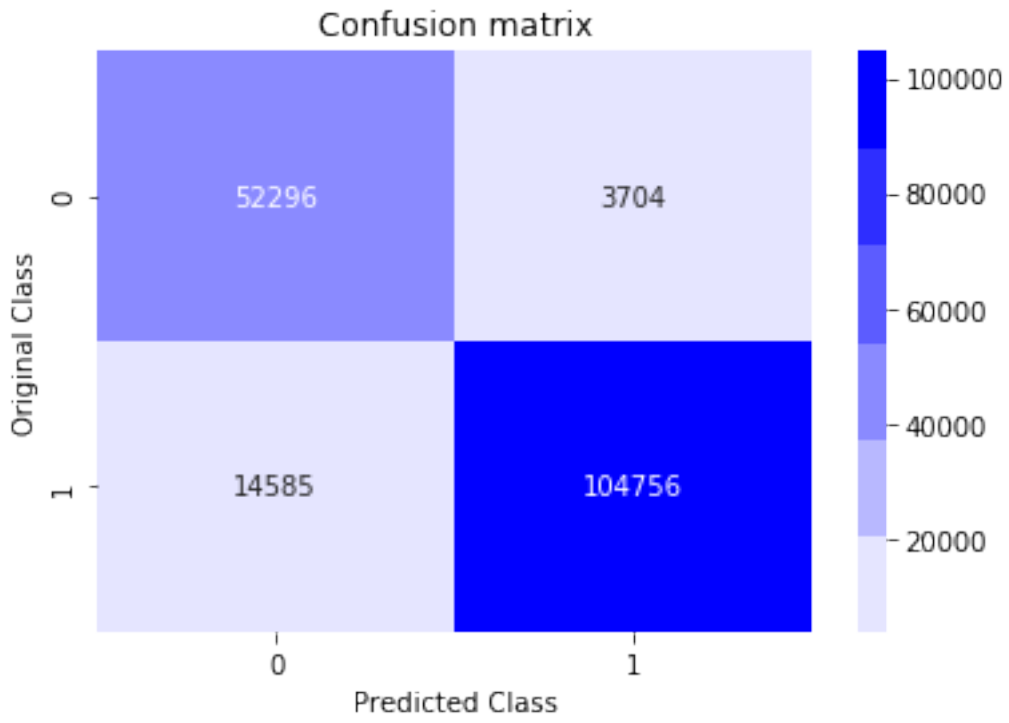
[146]:
```
cm_svm = confusion_matrix(y_test, predict_y_te_svm)
```

[147]:
```
tn, fp, fn, tp = cm_svm.ravel()
```

[148]:
```
fpr_svm = fp/(fp+tn)*100
fnr_svm = fn/(fn+tp)*100
far_svm = (fpr_svm+fnr_svm)/2
print("FAR:", far_svm)
```

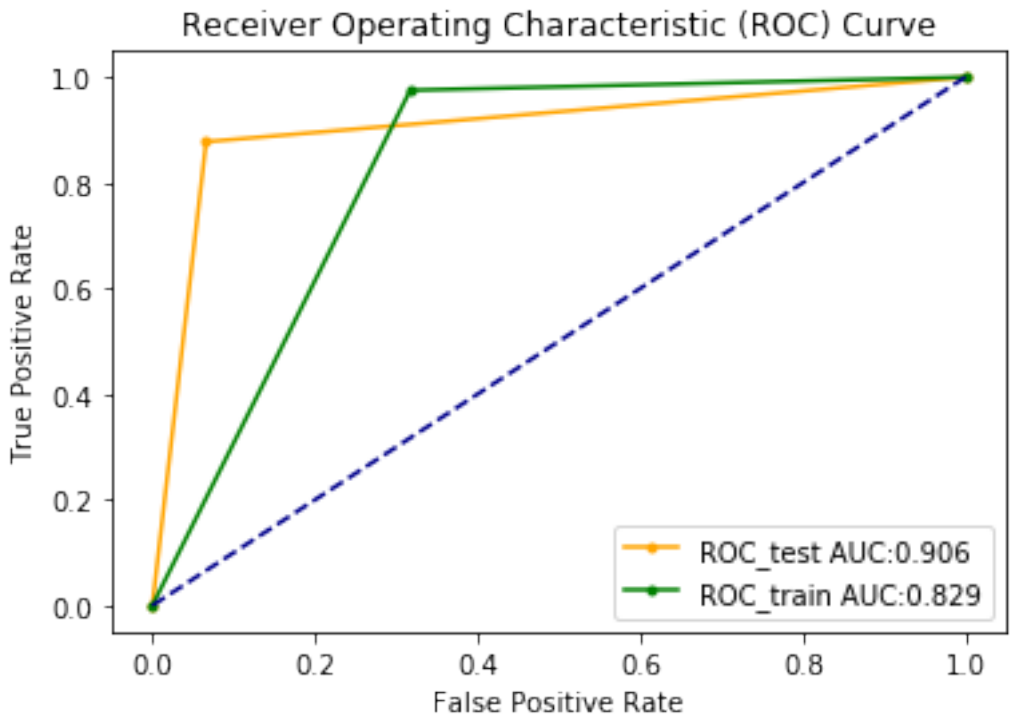
```
FAR: 9.417783793619005
```

[149]:
```
plot_cm(cm_svm)
```



[150]:
```
#finding the FPR and TPR for SVM set
fpr_te_svm, tpr_te_svm, t_te_svm = roc_curve(y_test, predict_y_te_svm)
fpr_tr_svm, tpr_tr_svm, t_tr_svm = roc_curve(y_train, predict_y_tr_svm)
auc_te_svm = auc(fpr_te_svm, tpr_te_svm)
print("AUC_SVM: ",auc_te_svm)
plot_roc_curve(fpr_tr_svm,tpr_tr_svm,fpr_te_svm, tpr_te_svm)
```

```
AUC_SVM:  0.9058221620638099
```



5.5 5.4 Random Forest Model

[151]:
```
param_grid = {"n_estimators": [10,100,500,1000, 2000],
    "min_samples_split": [50, 80, 120, 200],
              "max_depth": [3, 5, 10, 50, 100]}
rfc = RandomForestClassifier(criterion='gini', random_state=42, n_jobs=-1)
gridCV_rfc = GridSearchCV(rfc, param_grid, cv=3, verbose=10, n_jobs=-1)
gridCV_rfc.fit(df_train, y_train)
#grid Search cv results are stored in result for future use
results_rfc = pd.DataFrame.from_dict(gridCV_rfc.cv_results_)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    0.8s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:   22.6s
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:   49.9s
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:  1.9min
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  2.6min
[Parallel(n_jobs=-1)]: Done  45 tasks      | elapsed:  3.5min
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:  4.7min
[Parallel(n_jobs=-1)]: Done  69 tasks      | elapsed:  6.0min
[Parallel(n_jobs=-1)]: Done  82 tasks      | elapsed:  7.9min
[Parallel(n_jobs=-1)]: Done  97 tasks      | elapsed:  9.5min
```

```
[Parallel(n_jobs=-1)]: Done 112 tasks      | elapsed: 11.3min
[Parallel(n_jobs=-1)]: Done 129 tasks      | elapsed: 13.4min
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed: 16.1min
[Parallel(n_jobs=-1)]: Done 165 tasks      | elapsed: 18.6min
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed: 22.2min
[Parallel(n_jobs=-1)]: Done 205 tasks      | elapsed: 26.4min
[Parallel(n_jobs=-1)]: Done 226 tasks      | elapsed: 29.4min
[Parallel(n_jobs=-1)]: Done 249 tasks      | elapsed: 34.2min
[Parallel(n_jobs=-1)]: Done 272 tasks      | elapsed: 37.3min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 43.5min finished
```

[152]:
```
results_rfc = results_rfc.sort_values(['rank_test_score'])
results_rfc.head()
```

[152]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_min_samples_split	param_n_estimators	params	split0_test_score	split1_test_score	split2_test_score	mean_test_score	std_test_score	rank_test_score
62	63.188302	5.296170	8.351116	0.673394	50	50	500	{'max_depth': 50, 'min_samples_split': 50, 'n_…	0.950590	0.945453	0.902128	0.932724	0.021736	1
82	57.129693	3.058769	6.948288	0.296046	100	50	500	{'max_depth': 100, 'min_samples_split': 50, 'n…	0.950590	0.945453	0.902128	0.932724	0.021736	1
63	118.540384	13.901412	9.046493	0.692907	50	50	1000	{'max_depth': 50, 'min_samples_split': 50, 'n_…	0.950736	0.944979	0.902310	0.932675	0.021599	3
83	108.926263	12.328838	8.117628	0.368971	100	50	1000	{'max_depth': 100, 'min_samples_split': 50, 'n…	0.950736	0.944979	0.902310	0.932675	0.021599	3
64	224.453358	22.106221	11.170527	1.758222	50	50	2000	{'max_depth': 50, 'min_samples_split': 50, 'n_…	0.950772	0.945088	0.901399	0.932420	0.022057	5

```
64                  5
```

[153]:
```
print(gridCV_rfc.best_params_)
```

```
{'max_depth': 50, 'min_samples_split': 50, 'n_estimators': 500}
```

[154]:
```
rfc= RandomForestClassifier(criterion='gini', random_state=42, n_jobs=-1,␣
↪max_depth=gridCV_rfc.best_params_['max_depth'],min_samples_split=gridCV_rfc.
↪best_params_['min_samples_split'], n_estimators=gridCV_rfc.
↪best_params_['n_estimators'])
rfc.fit(df_train,y_train)
sig_clf_rfc = CalibratedClassifierCV(rfc, method="sigmoid")
sig_clf_rfc.fit(df_train, y_train)
predict_y_tr_rfc = sig_clf_rfc.predict(df_train)
predict_y_te_rfc = sig_clf_rfc.predict(df_test)
rfc_f1 = f1_score(y_test, predict_y_te_rfc)
print(rfc_f1)
```

```
0.929126145048273
```

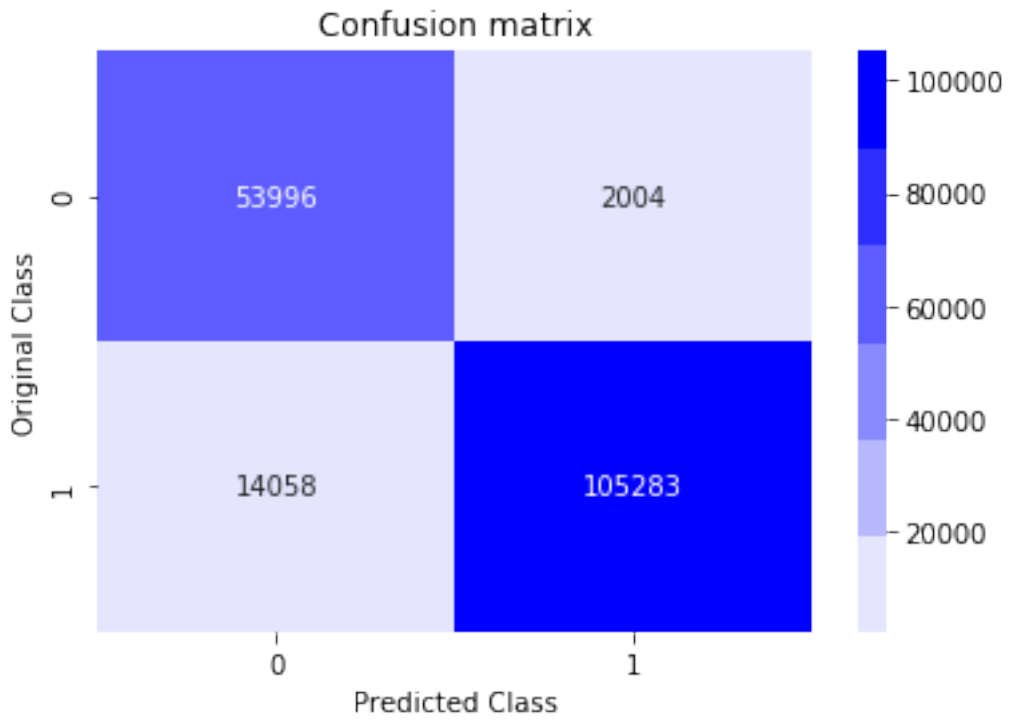
[155]:
```
cm_rfc = confusion_matrix(y_test, predict_y_te_rfc)
```

[156]:
```
tn, fp, fn, tp = cm_rfc.ravel()
```

[157]:
```
fpr_rfc = fp/(fp+tn)*100
fnr_rfc = fn/(fn+tp)*100
far_rfc = (fpr_rfc+fnr_rfc)/2
print("far:",far_rfc)
```

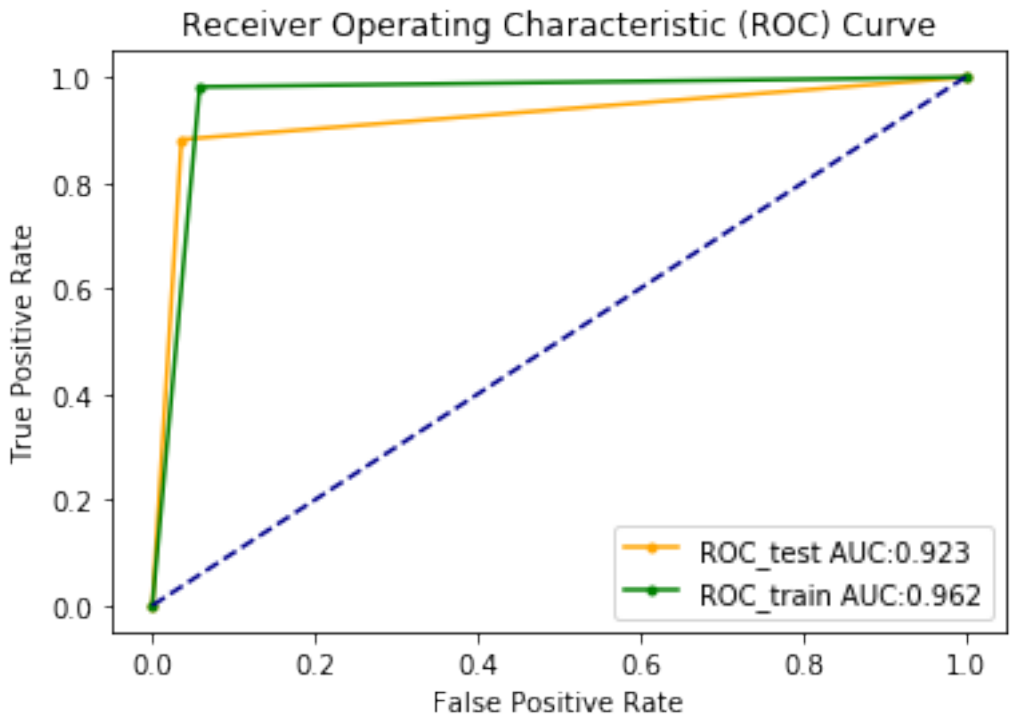
```
far: 7.679130780105509
```

[158]:
```
plot_cm(cm_rfc)
```



[159]:
```
#finding the FPR and TPR for RFC set
fpr_te_rfc, tpr_te_rfc, t_te_rfc = roc_curve(y_test, predict_y_te_rfc)
fpr_tr_rfc, tpr_tr_rfc, t_tr_rfc = roc_curve(y_train, predict_y_tr_rfc)
auc_te_rfc = auc(fpr_te_rfc, tpr_te_rfc)
print("AUC_RFC: ",auc_te_rfc)
plot_roc_curve(fpr_tr_rfc,tpr_tr_rfc,fpr_te_rfc, tpr_te_rfc)
```

```
AUC_RFC:  0.9232086921989449
```



5.6 5.5 Stacking classifier

[160]:
```
clf1 = SGDClassifier(alpha=0.0001,eta0=1, penalty='l2', loss='log', n_jobs =␣
↪-1, max_iter=10)
clf1.fit(df_train, y_train)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

clf2 = SGDClassifier(alpha=0.0001,eta0=0.0001, penalty='l2', loss='hinge',␣
↪n_jobs = -1, max_iter=5)
clf2.fit(df_train, y_train)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


clf3 = RandomForestClassifier(criterion='gini', random_state=42, n_jobs=-1,␣
↪max_depth=50,min_samples_split=50, n_estimators=10)
clf3.fit(df_train, y_train)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")
```

[81]:
```
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
```

```
    sclf = StackingClassifier(estimators=[("lr",sig_clf1), ("svm",␣
↪sig_clf2),("RF", sig_clf3)], final_estimator=lr, n_jobs=-1)
    sclf.fit(df_train, y_train)
    print("Stacking Classifer : for the value of alpha: %f Log loss: %0.3f␣
↪F1-score: %0.3f" % (i, log_loss(y_test, sclf.
↪predict_proba(df_test)),f1_score(y_test, sclf.predict(df_test))))
```

```
Stacking Classifer : for the value of alpha: 0.000100 Log loss: 0.422 F1-score:
0.946
Stacking Classifer : for the value of alpha: 0.001000 Log loss: 0.252 F1-score:
0.931
Stacking Classifer : for the value of alpha: 0.010000 Log loss: 0.221 F1-score:
0.930
Stacking Classifer : for the value of alpha: 0.100000 Log loss: 0.216 F1-score:
0.931
Stacking Classifer : for the value of alpha: 1.000000 Log loss: 0.215 F1-score:
0.931
Stacking Classifer : for the value of alpha: 10.000000 Log loss: 0.218 F1-score:
0.930
```

[101]:
```
lr = LogisticRegression(C=0.0001)
sig_clf_sc = StackingClassifier(estimators=[("lr",sig_clf1), ("svm",␣
 ↪sig_clf2),("RF", sig_clf3)], final_estimator=lr, n_jobs=-1)
sig_clf_sc.fit(df_train, y_train)
predict_y_tr_sc= sig_clf_sc.predict(df_train)
predict_y_te_sc = sig_clf_sc.predict(df_test)
sc_f1 = f1_score(y_test, predict_y_te_sc)
print(sc_f1)
```

```
0.9453929718455836
```

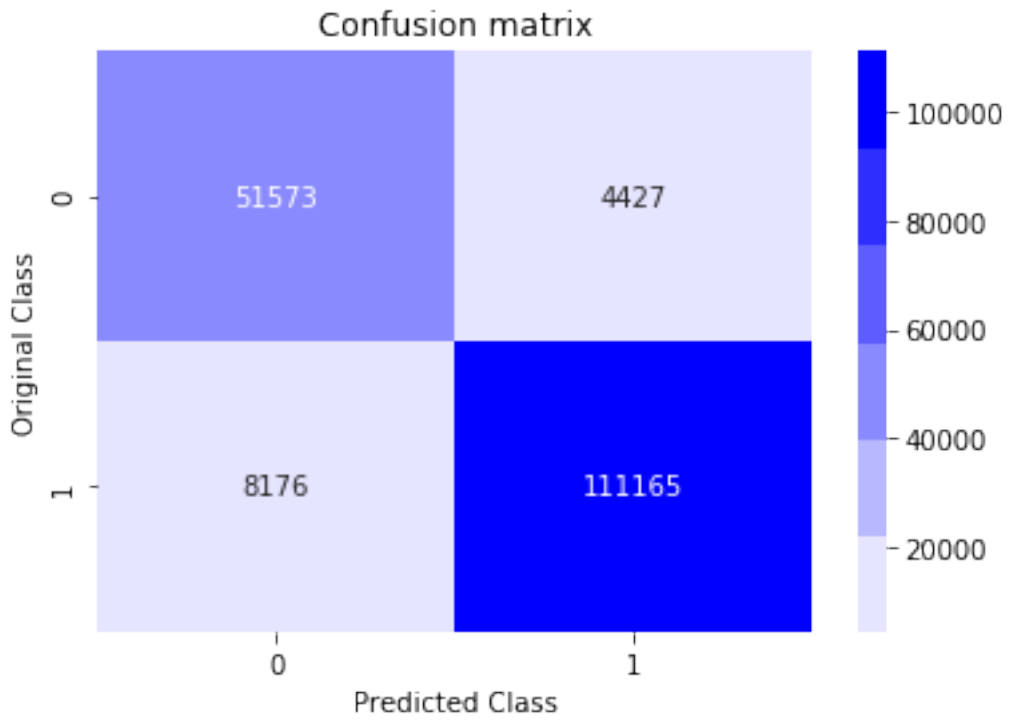
[102]:
```
cm_sc = confusion_matrix(y_test, predict_y_te_sc)
```

[103]:
```
tn, fp, fn, tp = cm_sc.ravel()
```

[104]:
```
fpr_sc = fp/(fp+tn)*100
fnr_sc = fn/(fn+tp)*100
far_sc = (fpr_sc+fnr_sc)/2
print("far:",far_sc)
```

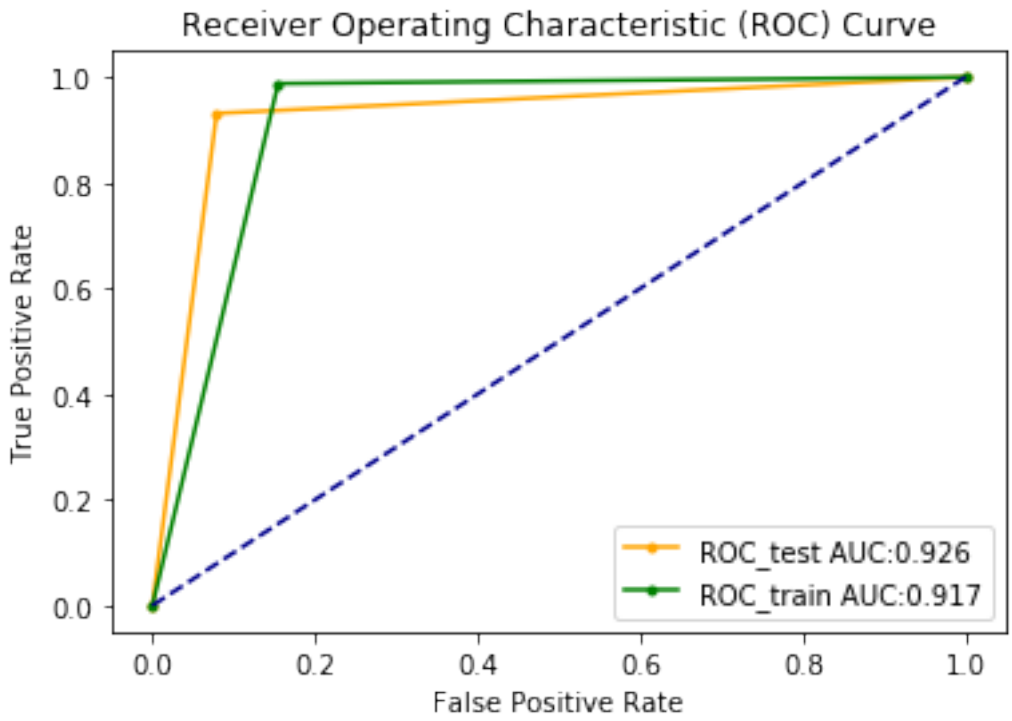
```
far: 7.396553550929091
```

[94]:
```
plot_cm(cm_sc)
```



[95]:
```
#finding the FPR and TPR for RFC set
fpr_te_sc, tpr_te_sc, t_te_sc = roc_curve(y_test, predict_y_te_sc)
fpr_tr_sc, tpr_tr_sc, t_tr_sc = roc_curve(y_train, predict_y_tr_sc)
auc_te_sc = auc(fpr_te_sc, tpr_te_sc)
print("AUC_SC: ",auc_te_sc)
plot_roc_curve(fpr_tr_sc,tpr_tr_sc,fpr_te_sc, tpr_te_sc)
```

```
AUC_SC:  0.9262184317717418
```



5.7 5.6. Model Evaluation

Measures	Equations
FPR	FP/(TN + FP)
FNR	FN/(FN + TP)
FAR	(FPR + FNR)/2

[178]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model", "F1 Score", "AUC","FPR %","FNR %","FAR %"]
x.add_row(["Logistic Regression", "{0:.4}".format(lr_f1), "{0:.4}".
 ↪format(auc_te_lr),"%.2f" % float(fpr_lr),"%.2f" % float(fnr_lr),"%.2f" %␣
 ↪float(far_lr)])
x.add_row(["Linear SVM", "{0:.4}".format(svm_f1), "{0:.4}".
 ↪format(auc_te_svm),"%.2f" % float(fpr_svm),"%.2f" % float(fnr_svm),"%.2f" %␣
 ↪float(far_svm)])
x.add_row(["Random Forest", "{0:.4}".format(rfc_f1), "{0:.4}".
 ↪format(auc_te_rfc),"%.2f" % float(fpr_rfc),"%.2f" % float(fnr_rfc),"%.2f" %␣
 ↪float(far_rfc)])
x.add_row(["Stacking Classifier", "{0:.4}".format(sc_f1), "{0:.4}".
 ↪format(auc_te_sc),"%.2f" % float(fpr_sc),"%.2f" % float(fnr_sc),"%.2f" %␣
 ↪float(far_sc)])
```

```
print(x)
```

```
+---------------------+----------+--------+-------+-------+-------+
|        Model        | F1 Score |  AUC   | FPR % | FNR % | FAR % |
+---------------------+----------+--------+-------+-------+-------+
| Logistic Regression |  0.9423  | 0.9051 | 13.88 |  5.11 |  9.49 |
|      Linear SVM     |  0.9197  | 0.9058 |  6.61 | 12.22 |  9.42 |
|    Random Forest    |  0.9291  | 0.9232 |  3.58 | 11.78 |  7.68 |
| Stacking Classifier |  0.9453  | 0.9262 |  7.91 |  6.85 |  7.39 |
+---------------------+----------+--------+-------+-------+-------+
```

6 6. Machine Learning Models (Label Encoder)

6.1 6.1 Reading Train and Test data

[17]:
```
train_data = pd.read_csv("data/UNSW_NB15_training-set.csv")
print(train_data.shape)
train_data.head()
```

```
(82332, 45)
```

[17]:

	id	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	\
0	1	0.000011	udp	-	INT	2	0	496	0	
1	2	0.000008	udp	-	INT	2	0	1762	0	
2	3	0.000005	udp	-	INT	2	0	1068	0	
3	4	0.000006	udp	-	INT	2	0	900	0	
4	5	0.000010	udp	-	INT	2	0	2126	0	

	rate	…	ct_dst_sport_ltm	ct_dst_src_ltm	is_ftp_login	\
0	90909.0902	…	1	2	0	
1	125000.0003	…	1	2	0	
2	200000.0051	…	1	3	0	
3	166666.6608	…	1	3	0	
4	100000.0025	…	1	3	0	

	ct_ftp_cmd	ct_flw_http_mthd	ct_src_ltm	ct_srv_dst	is_sm_ips_ports	\
0	0	0	1	2	0	
1	0	0	1	2	0	
2	0	0	1	3	0	
3	0	0	2	3	0	
4	0	0	2	3	0	

	attack_cat	label
0	Normal	0
1	Normal	0
2	Normal	0
3	Normal	0

```
4      Normal      0

[5 rows x 45 columns]
```

[18]:
```
cat_feature = train_data.select_dtypes(include=['category', object]).columns
```

[19]:
```
train_data[cat_feature] = train_data[cat_feature].apply(LabelEncoder().
↪fit_transform)
train_data.head()
```

[19]:

	id	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	\
0	1	0.000011	117	0	4	2	0	496	0	
1	2	0.000008	117	0	4	2	0	1762	0	
2	3	0.000005	117	0	4	2	0	1068	0	
3	4	0.000006	117	0	4	2	0	900	0	
4	5	0.000010	117	0	4	2	0	2126	0	

	rate	…	ct_dst_sport_ltm	ct_dst_src_ltm	is_ftp_login	\
0	90909.0902	…	1	2	0	
1	125000.0003	…	1	2	0	
2	200000.0051	…	1	3	0	
3	166666.6608	…	1	3	0	
4	100000.0025	…	1	3	0	

	ct_ftp_cmd	ct_flw_http_mthd	ct_src_ltm	ct_srv_dst	is_sm_ips_ports	\
0	0	0	1	2	0	
1	0	0	1	2	0	
2	0	0	1	3	0	
3	0	0	2	3	0	
4	0	0	2	3	0	

	attack_cat	label
0	6	0
1	6	0
2	6	0
3	6	0
4	6	0

[5 rows x 45 columns]

[20]:
```
df_train = train_data[list(col_set)]
df_train.head()
```

[20]:

	ackdat	dmean	proto	ct_state_ttl	ct_dst_ltm	trans_depth	service	\
0	0.0	0	117	2	1	0	0	
1	0.0	0	117	2	1	0	0	
2	0.0	0	117	2	1	0	0	

```
3     0.0      0    117              2           2            0        0
4     0.0      0    117              2           2            0        0

   ct_dst_src_ltm  response_body_len  dload  …  djit  synack  sjit  dloss  \
0               2                  0    0.0  …   0.0     0.0   0.0      0
1               2                  0    0.0  …   0.0     0.0   0.0      0
2               3                  0    0.0  …   0.0     0.0   0.0      0
3               3                  0    0.0  …   0.0     0.0   0.0      0
4               3                  0    0.0  …   0.0     0.0   0.0      0

   dtcpb  swin  stcpb  tcprtt  is_sm_ips_ports  sttl
0      0     0      0     0.0                0   254
1      0     0      0     0.0                0   254
2      0     0      0     0.0                0   254
3      0     0      0     0.0                0   254
4      0     0      0     0.0                0   254

[5 rows x 30 columns]
```

[21]:
```
test_data = pd.read_csv("data/UNSW_NB15_testing-set.csv")
print(test_data.shape)
test_data.head()
```

```
(175341, 45)
```

[21]:
```
   id       dur proto service state  spkts  dpkts  sbytes  dbytes       rate  \
0   1  0.121478   tcp       -   FIN      6      4     258     172  74.087490
1   2  0.649902   tcp       -   FIN     14     38     734   42014  78.473372
2   3  1.623129   tcp       -   FIN      8     16     364   13186  14.170161
3   4  1.681642   tcp     ftp   FIN     12     12     628     770  13.677108
4   5  0.449454   tcp       -   FIN     10      6     534     268  33.373826

   …  ct_dst_sport_ltm  ct_dst_src_ltm  is_ftp_login  ct_ftp_cmd  \
0  …                 1               1             0           0
1  …                 1               2             0           0
2  …                 1               3             0           0
3  …                 1               3             1           1
4  …                 1              40             0           0

   ct_flw_http_mthd  ct_src_ltm  ct_srv_dst  is_sm_ips_ports  attack_cat  \
0                 0           1           1                0      Normal
1                 0           1           6                0      Normal
2                 0           2           6                0      Normal
3                 0           2           1                0      Normal
4                 0           2          39                0      Normal

   label
```

```
0      0
1      0
2      0
3      0
4      0

[5 rows x 45 columns]
```

[22]:
```
cat_feature_test = test_data.select_dtypes(include=['category', object]).columns
```

[23]:
```
test_data[cat_feature_test] = test_data[cat_feature_test].apply(LabelEncoder().
↪fit_transform)
test_data.head()
```

[23]:
```
   id       dur  proto  service  state  spkts  dpkts  sbytes  dbytes  \
0   1  0.121478    113        0      2      6      4     258     172
1   2  0.649902    113        0      2     14     38     734   42014
2   3  1.623129    113        0      2      8     16     364   13186
3   4  1.681642    113        3      2     12     12     628     770
4   5  0.449454    113        0      2     10      6     534     268

         rate  …  ct_dst_sport_ltm  ct_dst_src_ltm  is_ftp_login  ct_ftp_cmd  \
0  74.087490  …                 1               1             0           0
1  78.473372  …                 1               2             0           0
2  14.170161  …                 1               3             0           0
3  13.677108  …                 1               3             1           1
4  33.373826  …                 1              40             0           0

   ct_flw_http_mthd  ct_src_ltm  ct_srv_dst  is_sm_ips_ports  attack_cat  \
0                 0           1           1                0           6
1                 0           1           6                0           6
2                 0           2           6                0           6
3                 0           2           1                0           6
4                 0           2          39                0           6

   label
0      0
1      0
2      0
3      0
4      0

[5 rows x 45 columns]
```

[24]:
```
df_test = test_data[list(col_set)]
df_test.head()
```

[24]:
```
      ackdat  dmean  proto  ct_state_ttl  ct_dst_ltm  trans_depth  service  \
0  0.000000     43    113             0           1            0        0
1  0.000000   1106    113             1           1            0        0
2  0.050439    824    113             1           2            0        0
3  0.000000     64    113             1           2            0        3
4  0.057234     45    113             1           2            0        0

   ct_dst_src_ltm  response_body_len          dload  …          djit  \
0               1                  0    8495.365234  …     11.830604
1               2                  0  503571.312500  …   1387.778330
2               3                  0   60929.230470  …  11420.926230
3               3                  0    3358.622070  …   4991.784669
4              40                  0    3987.059814  …    115.807000

     synack          sjit  dloss       dtcpb  swin       stcpb    tcprtt  \
0  0.000000     30.177547      0  2202533631   255   621772692  0.000000
1  0.000000     61.426934     17  3077387971   255  1417884146  0.000000
2  0.061458  17179.586860      6  2963114973   255  2116150707  0.111897
3  0.000000    259.080172      3  1047442890   255  1107119177  0.000000
4  0.071147   2415.837634      1  1977154190   255  2436137549  0.128381

   is_sm_ips_ports  sttl
0                0   252
1                0    62
2                0    62
3                0    62
4                0   254

[5 rows x 30 columns]
```

6.1.1 6.1.1 Standardize the data

[25]:
```
x = df_train.values
x_test = df_test.values
std_scaler = preprocessing.MinMaxScaler()
std_scaler.fit(x)
x_scaled = std_scaler.transform(x)
df_train = pd.DataFrame(x_scaled)
x_scaled_test = std_scaler.transform(x_test)
df_test = pd.DataFrame(x_scaled_test)
```

[26]:
```
df_train.head()
```

[26]:
```
     0    1    2         3         4    5    6         7    8    9   …   20  \
0  0.0  0.0  0.9  0.333333  0.000000  0.0  0.0  0.016129  0.0  0.0  …  0.0
1  0.0  0.0  0.9  0.333333  0.000000  0.0  0.0  0.016129  0.0  0.0  …  0.0
2  0.0  0.0  0.9  0.333333  0.000000  0.0  0.0  0.032258  0.0  0.0  …  0.0
```

```
3  0.0  0.0  0.9  0.333333  0.017241  0.0  0.0  0.032258  0.0  0.0  …  0.0
4  0.0  0.0  0.9  0.333333  0.017241  0.0  0.0  0.032258  0.0  0.0  …  0.0

    21   22   23   24   25   26   27   28        29
0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.996078
1  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.996078
2  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.996078
3  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.996078
4  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.996078

[5 rows x 30 columns]
```

[27]:
```
y_train = train_data['label']
y_test = test_data['label']
print("train data shape", df_train.shape, y_train.shape)
print("test data shape", df_test.shape, y_test.shape)
```

```
train data shape (82332, 30) (82332,)
test data shape (175341, 30) (175341,)
```

[28]:
```
[10 ** x for x in range(-5, 2)]
```

[28]:
```
[1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10]
```

6.2 6.2 Logistic Regression Model

[156]:
```
prams={
    'alpha':[10 ** x for x in range(-4, 1)],
     'max_iter':[5, 10, 20, 50, 100],
    'eta0': [10 ** x for x in range(-4, 1)]
}
lr_cfl=GridSearchCV(SGDClassifier(penalty='l2', loss='log', n_jobs = -1),␣
 ↪param_grid=prams,verbose=10,n_jobs=-1)
lr_cfl.fit(df_train,y_train)
```

```
Fitting 5 folds for each of 125 candidates, totalling 625 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    0.6s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:    1.3s
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:    1.9s
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    2.6s
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    3.4s
[Parallel(n_jobs=-1)]: Done  45 tasks      | elapsed:    4.4s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:    5.4s
[Parallel(n_jobs=-1)]: Done  69 tasks      | elapsed:    6.5s
[Parallel(n_jobs=-1)]: Done  82 tasks      | elapsed:    7.5s
[Parallel(n_jobs=-1)]: Done  97 tasks      | elapsed:    8.8s
```

```
[Parallel(n_jobs=-1)]: Done 112 tasks      | elapsed:    9.9s
[Parallel(n_jobs=-1)]: Done 129 tasks      | elapsed:   11.5s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:   13.1s
[Parallel(n_jobs=-1)]: Done 165 tasks      | elapsed:   14.5s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   16.0s
[Parallel(n_jobs=-1)]: Done 205 tasks      | elapsed:   17.7s
[Parallel(n_jobs=-1)]: Done 226 tasks      | elapsed:   19.2s
[Parallel(n_jobs=-1)]: Done 249 tasks      | elapsed:   21.1s
[Parallel(n_jobs=-1)]: Done 272 tasks      | elapsed:   22.9s
[Parallel(n_jobs=-1)]: Done 297 tasks      | elapsed:   24.8s
[Parallel(n_jobs=-1)]: Done 322 tasks      | elapsed:   26.8s
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:   28.9s
[Parallel(n_jobs=-1)]: Done 376 tasks      | elapsed:   30.9s
[Parallel(n_jobs=-1)]: Done 405 tasks      | elapsed:   33.4s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   35.4s
[Parallel(n_jobs=-1)]: Done 465 tasks      | elapsed:   37.9s
[Parallel(n_jobs=-1)]: Done 496 tasks      | elapsed:   40.4s
[Parallel(n_jobs=-1)]: Done 529 tasks      | elapsed:   42.7s
[Parallel(n_jobs=-1)]: Done 562 tasks      | elapsed:   45.2s
[Parallel(n_jobs=-1)]: Done 597 tasks      | elapsed:   48.2s
[Parallel(n_jobs=-1)]: Done 625 out of 625 | elapsed:   50.0s finished
```

[156]:
```
GridSearchCV(cv=None, error_score=nan,
             estimator=SGDClassifier(alpha=0.0001, average=False,
                                     class_weight=None, early_stopping=False,
                                     epsilon=0.1, eta0=0.0, fit_intercept=True,
                                     l1_ratio=0.15, learning_rate='optimal',
                                     loss='log', max_iter=1000,
                                     n_iter_no_change=5, n_jobs=-1,
                                     penalty='l2', power_t=0.5,
                                     random_state=None, shuffle=True, tol=0.001,
                                     validation_fraction=0.1, verbose=0,
                                     warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1],
                         'eta0': [0.0001, 0.001, 0.01, 0.1, 1],
                         'max_iter': [5, 10, 20, 50, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=10)
```

[157]:
```
results = pd.DataFrame.from_dict(lr_cfl.cv_results_)
results = results.sort_values(['rank_test_score'])
results.head()
```

[157]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	\
9	0.659036	0.014011	0.007781	0.000399	0.0001	
7	0.619651	0.027652	0.008979	0.001263	0.0001	

```
6        0.601999      0.037194         0.010476        0.001782      0.0001
21       0.626726      0.041205         0.011367        0.001019      0.0001
16       0.582242      0.020178         0.011183        0.005554      0.0001

   param_eta0 param_max_iter  \
9       0.001            100
7       0.001             20
6       0.001             10
21          1             10
16        0.1             10

                                                 params  split0_test_score  \
9   {'alpha': 0.0001, 'eta0': 0.001, 'max_iter': 100}           0.913463
7    {'alpha': 0.0001, 'eta0': 0.001, 'max_iter': 20}           0.905629
6    {'alpha': 0.0001, 'eta0': 0.001, 'max_iter': 10}           0.905872
21      {'alpha': 0.0001, 'eta0': 1, 'max_iter': 10}           0.904415
16    {'alpha': 0.0001, 'eta0': 0.1, 'max_iter': 10}           0.899435

    split1_test_score  split2_test_score  split3_test_score  \
9            0.961134           0.858071           0.795943
7            0.978928           0.845621           0.793878
6            0.944131           0.859104           0.801348
21           0.966721           0.839184           0.797279
16           0.978381           0.838759           0.776388

    split4_test_score  mean_test_score  std_test_score  rank_test_score
9            0.772015         0.860125        0.070618                1
7            0.765395         0.857890        0.077113                2
6            0.765820         0.855255        0.065392                3
21           0.760962         0.853712        0.073946                4
16           0.760962         0.850785        0.080493                5
```

[158]:
```
print(lr_cfl.best_params_)
```

```
{'alpha': 0.0001, 'eta0': 0.001, 'max_iter': 100}
```

[159]:
```
logisticR=SGDClassifier(alpha=lr_cfl.best_params_['alpha'],eta0=lr_cfl.
 ↪best_params_['eta0'], penalty='l2', loss='log', n_jobs = -1, max_iter=lr_cfl.
 ↪best_params_['max_iter'])
logisticR.fit(df_train,y_train)
sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
sig_clf.fit(df_train, y_train)
predict_y_tr_lr = sig_clf.predict(df_train)
predict_y_te_lr = sig_clf.predict(df_test)
lr_f1 = f1_score(y_test, predict_y_te_lr)
print(lr_f1)
```

```
0.8029353446987435
```

[160]:
```
cm_lr = confusion_matrix(y_test, predict_y_te_lr)
```

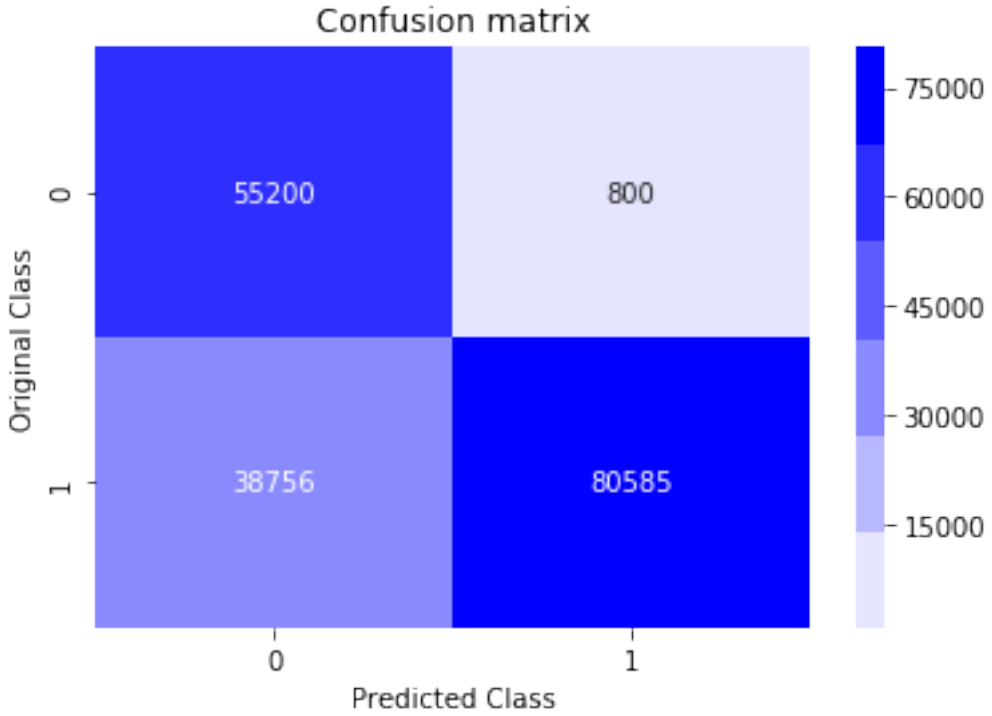
[161]:
```
tn, fp, fn, tp = cm_lr.ravel()
```

[162]:
```
fpr_lr = fp/(fp+tn)*100
fnr_lr = fn/(fn+tp)*100
far_lr = (fpr_lr+fnr_lr)/2
print("FAR: %0.2f" %far_lr)
```

```
FAR: 16.95
```

[40]:
```
def plot_cm(cm):
    sns.heatmap(cm, annot=True, cmap=sns.light_palette("blue"), fmt="g")
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")
    plt.show()
```

[164]:
```
plot_cm(cm_lr)
```

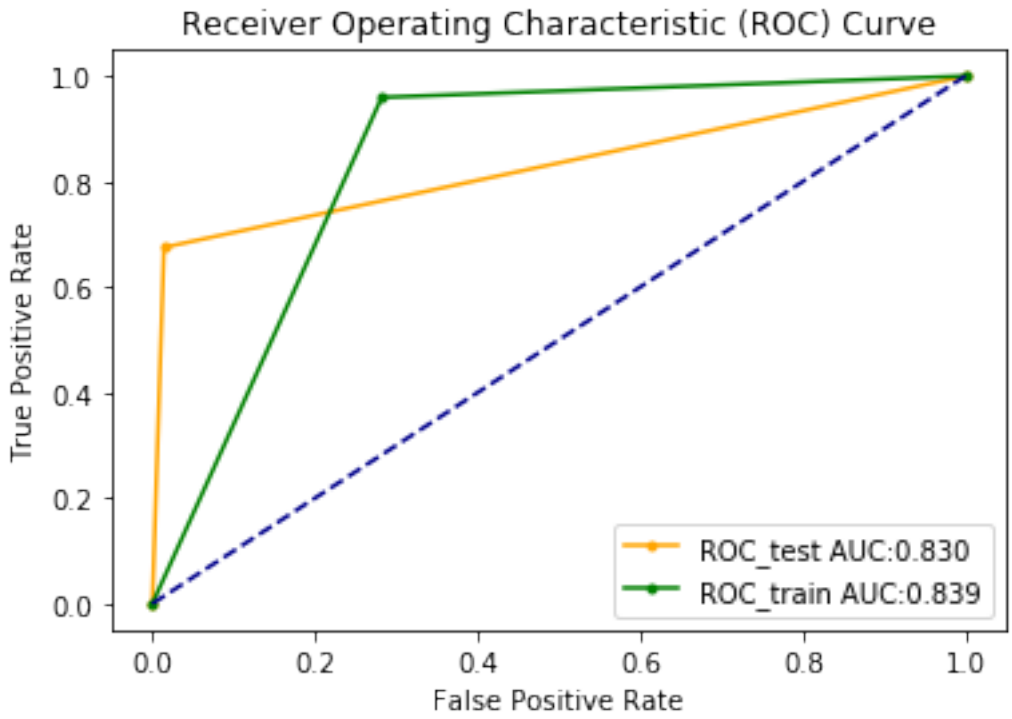


[43]:
```
from sklearn.metrics import roc_curve, auc
def plot_roc_curve(fpr_tr, tpr_tr,fpr_te, tpr_te):
```

```
    '''
    plot the ROC curve for the FPR and TPR value
    '''
    plt.plot(fpr_te, tpr_te, 'k.-', color='orange', label='ROC_test AUC:%.3f'%␣
↪auc(fpr_te, tpr_te))
    plt.plot(fpr_tr, tpr_tr, 'k.-', color='green', label='ROC_train AUC:%.3f'%␣
↪auc(fpr_tr, tpr_tr))
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()
```

[166]:
```
#finding the FPR and TPR for logistic reg model set
fpr_te_lr, tpr_te_lr, t_te_lr = roc_curve(y_test, predict_y_te_lr)
fpr_tr_lr, tpr_tr_lr, t_tr_lr = roc_curve(y_train, predict_y_tr_lr)
auc_te_lr = auc(fpr_te_lr, tpr_te_lr)
print("AUC_LR: ",auc_te_lr)
plot_roc_curve(fpr_tr_lr,tpr_tr_lr,fpr_te_lr, tpr_te_lr)
```

```
AUC_LR:  0.8304820999129745
```



6.3 6.3 Support Vector Machine Model

[167]:
```
prams={
    'alpha':[10 ** x for x in range(-4, 1)],
     'max_iter':[5, 10, 20, 50, 100],
    'eta0': [10 ** x for x in range(-4, 1)]
}
svm_cfl=GridSearchCV(SGDClassifier(penalty='l1', loss='hinge', n_jobs = -1),␣
 ↪param_grid=prams,verbose=10,n_jobs=-1)
svm_cfl.fit(df_train,y_train)
```

```
Fitting 5 folds for each of 125 candidates, totalling 625 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    0.7s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:    1.5s
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:    2.3s
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    3.3s
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    4.1s
[Parallel(n_jobs=-1)]: Done  45 tasks      | elapsed:    5.4s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:    6.5s
[Parallel(n_jobs=-1)]: Done  69 tasks      | elapsed:    7.9s
[Parallel(n_jobs=-1)]: Done  82 tasks      | elapsed:    9.2s
[Parallel(n_jobs=-1)]: Done  97 tasks      | elapsed:   11.0s
[Parallel(n_jobs=-1)]: Done 112 tasks      | elapsed:   12.4s
[Parallel(n_jobs=-1)]: Done 129 tasks      | elapsed:   14.1s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:   15.6s
[Parallel(n_jobs=-1)]: Done 165 tasks      | elapsed:   17.4s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   19.0s
[Parallel(n_jobs=-1)]: Done 205 tasks      | elapsed:   20.8s
[Parallel(n_jobs=-1)]: Done 226 tasks      | elapsed:   22.6s
[Parallel(n_jobs=-1)]: Done 249 tasks      | elapsed:   24.7s
[Parallel(n_jobs=-1)]: Done 272 tasks      | elapsed:   26.5s
[Parallel(n_jobs=-1)]: Done 297 tasks      | elapsed:   28.5s
[Parallel(n_jobs=-1)]: Done 322 tasks      | elapsed:   30.5s
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:   32.9s
[Parallel(n_jobs=-1)]: Done 376 tasks      | elapsed:   34.9s
[Parallel(n_jobs=-1)]: Done 405 tasks      | elapsed:   37.3s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   39.6s
[Parallel(n_jobs=-1)]: Done 465 tasks      | elapsed:   42.3s
[Parallel(n_jobs=-1)]: Done 496 tasks      | elapsed:   45.1s
[Parallel(n_jobs=-1)]: Done 529 tasks      | elapsed:   48.0s
[Parallel(n_jobs=-1)]: Done 562 tasks      | elapsed:   51.0s
[Parallel(n_jobs=-1)]: Done 597 tasks      | elapsed:   54.3s
[Parallel(n_jobs=-1)]: Done 625 out of 625 | elapsed:   56.9s finished
```

[167]:
```
GridSearchCV(cv=None, error_score=nan,
             estimator=SGDClassifier(alpha=0.0001, average=False,
```

```
                                   class_weight=None, early_stopping=False,
                                   epsilon=0.1, eta0=0.0, fit_intercept=True,
                                   l1_ratio=0.15, learning_rate='optimal',
                                   loss='hinge', max_iter=1000,
                                   n_iter_no_change=5, n_jobs=-1,
                                   penalty='l1', power_t=0.5,
                                   random_state=None, shuffle=True, tol=0.001,
                                   validation_fraction=0.1, verbose=0,
                                   warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1],
                         'eta0': [0.0001, 0.001, 0.01, 0.1, 1],
                         'max_iter': [5, 10, 20, 50, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=10)
```

[168]:
```
print(svm_cfl.best_params_)
```

```
{'alpha': 0.0001, 'eta0': 0.01, 'max_iter': 100}
```

[169]:
```
svm=SGDClassifier(alpha=svm_cfl.best_params_['alpha'],eta0=svm_cfl.
 ↪best_params_['eta0'], penalty='l2', loss='hinge', n_jobs = -1,␣
 ↪max_iter=svm_cfl.best_params_['max_iter'])
svm.fit(df_train,y_train)
sig_clf_svm = CalibratedClassifierCV(svm, method="sigmoid")
sig_clf_svm.fit(df_train, y_train)
predict_y_tr_svm = sig_clf.predict(df_train)
predict_y_te_svm = sig_clf_svm.predict(df_test)
svm_f1 = f1_score(y_test, predict_y_te_svm)
print("F1-Score", svm_f1)
```

```
F1-Score 0.8570189661023341
```

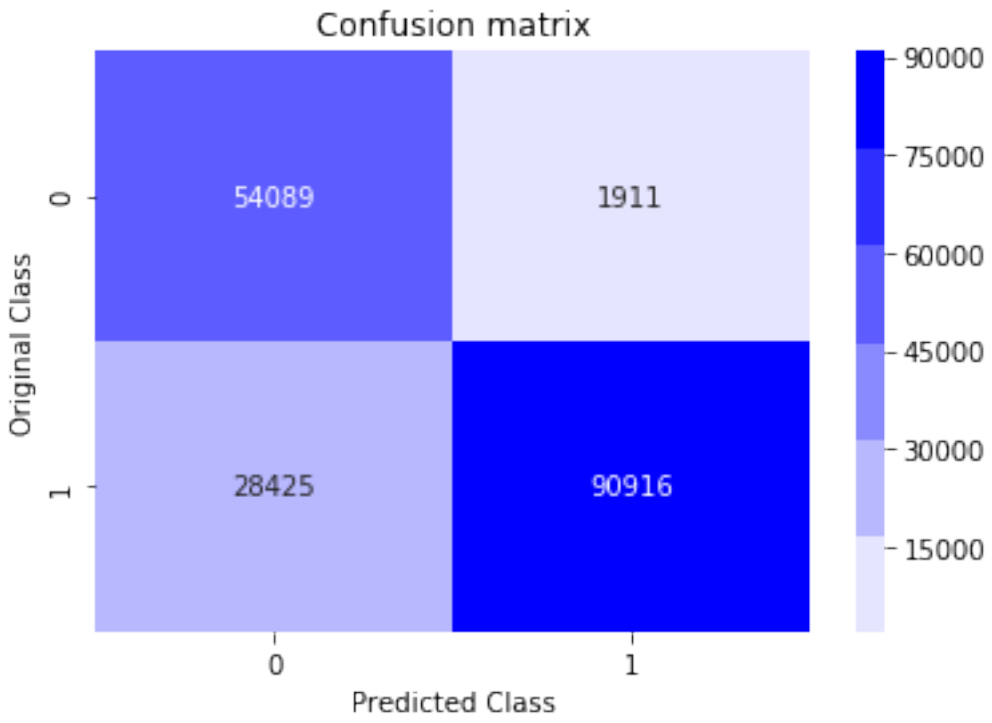
[170]:
```
cm_svm = confusion_matrix(y_test, predict_y_te_svm)
```

[171]:
```
tn, fp, fn, tp = cm_svm.ravel()
```

[172]:
```
fpr_svm = fp/(fp+tn)*100
fnr_svm = fn/(fn+tp)*100
far_svm = (fpr_svm+fnr_svm)/2
print("FAR: %0.2f" % far_svm)
```

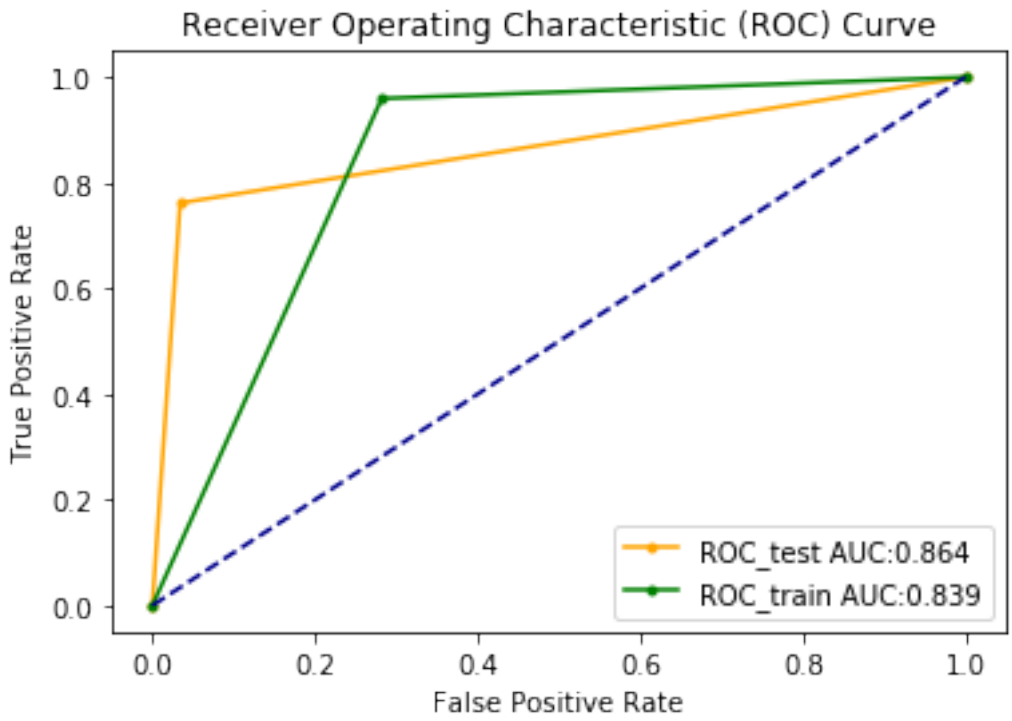
```
FAR: 13.62
```

[173]:
```
plot_cm(cm_svm)
```



[174]:
```
#finding the FPR and TPR for SVM set
fpr_te_svm, tpr_te_svm, t_te_svm = roc_curve(y_test, predict_y_te_svm)
fpr_tr_svm, tpr_tr_svm, t_tr_svm = roc_curve(y_train, predict_y_tr_svm)
auc_te_svm = auc(fpr_te_svm, tpr_te_svm)
print("AUC_SVM: ",auc_te_svm)
plot_roc_curve(fpr_tr_svm,tpr_tr_svm,fpr_te_svm, tpr_te_svm)
```

```
AUC_SVM:  0.8638459891194141
```



6.4 6.4 Random Forest Model

[175]:
```
param_grid = {"n_estimators": [10,100,500,1000, 2000],
    "min_samples_split": [50, 80, 120, 200],
                "max_depth": [3, 5, 10, 50, 100]}
rfc = RandomForestClassifier(criterion='gini', random_state=42, n_jobs=-1)
gridCV_rfc = GridSearchCV(rfc, param_grid, cv=3, return_train_score=True,␣
 ↪verbose=10, n_jobs=-1)
gridCV_rfc.fit(df_train, y_train)
#grid Search cv results are stored in result for future use
results_rfc = pd.DataFrame.from_dict(gridCV_rfc.cv_results_)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    4.4s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:   32.8s
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:   58.6s
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:  2.0min
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  2.9min
[Parallel(n_jobs=-1)]: Done  45 tasks      | elapsed:  3.8min
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:  5.2min
[Parallel(n_jobs=-1)]: Done  69 tasks      | elapsed:  6.8min
[Parallel(n_jobs=-1)]: Done  82 tasks      | elapsed:  8.5min
```

```
[Parallel(n_jobs=-1)]: Done  97 tasks      | elapsed: 10.1min
[Parallel(n_jobs=-1)]: Done 112 tasks      | elapsed: 11.7min
[Parallel(n_jobs=-1)]: Done 129 tasks      | elapsed: 13.7min
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed: 16.4min
[Parallel(n_jobs=-1)]: Done 165 tasks      | elapsed: 19.1min
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed: 22.8min
[Parallel(n_jobs=-1)]: Done 205 tasks      | elapsed: 27.2min
[Parallel(n_jobs=-1)]: Done 226 tasks      | elapsed: 30.5min
[Parallel(n_jobs=-1)]: Done 249 tasks      | elapsed: 35.6min
[Parallel(n_jobs=-1)]: Done 272 tasks      | elapsed: 39.2min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 45.8min finished
```

[176]:
```
results_rfc = results_rfc.sort_values(['rank_test_score'])
results_rfc.head()
```

[176]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_min_samples_split	param_n_estimators	params	split0_test_score	split1_test_score	split2_test_score	mean_test_score	std_test_score	rank_test_score	split0_train_score	split1_train_score
64	207.673127	17.483878	14.140464	1.567208	50	50	2000	{'max_depth': 50, 'min_samples_split': 50, 'n_…	0.874071	0.944651	0.898921	0.905881	0.029231	1	0.969374	0.967461
84	208.767452	17.792458	14.420839	1.246119	100	50	2000	{'max_depth': 100, 'min_samples_split': 50, 'n…	0.874071	0.944651	0.898921	0.905881	0.029231	1	0.969374	0.967461
63	109.719585	14.179158	7.163603	0.395994	50	50	1000	{'max_depth': 50, 'min_samples_split': 50, 'n_…	0.871192	0.944760	0.899177	0.905043	0.030319	3	0.969210	0.967643
83	108.777701	11.380208	10.336323	0.203533	100	50	1000	{'max_depth': 100, 'min_samples_split': 50, 'n…	0.871192	0.944760	0.899177	0.905043	0.030319	3	0.969210	0.967643
82	55.538241	2.770431	6.988311	0.162329	100	50	500	{'max_depth': 100, 'min_samples_split': 50, 'n…	0.871302	0.944724	0.898849	0.904958	0.030284	5	0.969210	0.967643

```
83                3           0.969210            0.967643
82                5           0.969101            0.967607

    split2_train_score  mean_train_score  std_train_score
64            0.974767          0.970534         0.003093
84            0.974767          0.970534         0.003093
63            0.974730          0.970528         0.003040
83            0.974730          0.970528         0.003040
82            0.974785          0.970497         0.003092
```

[177]:
```
print(gridCV_rfc.best_params_)
```

```
{'max_depth': 50, 'min_samples_split': 50, 'n_estimators': 2000}
```

[178]:
```
rfc= RandomForestClassifier(criterion='gini', random_state=42, n_jobs=-1,␣
 ↪max_depth=gridCV_rfc.best_params_['max_depth'],min_samples_split=gridCV_rfc.
 ↪best_params_['min_samples_split'], n_estimators=gridCV_rfc.
 ↪best_params_['n_estimators'])
rfc.fit(df_train,y_train)
sig_clf_rfc = CalibratedClassifierCV(rfc, method="sigmoid")
sig_clf_rfc.fit(df_train, y_train)
predict_y_tr_rfc = sig_clf_rfc.predict(df_train)
predict_y_te_rfc = sig_clf_rfc.predict(df_test)
rfc_f1 = f1_score(y_test, predict_y_te_rfc)
print(rfc_f1)
```

```
0.9405912523518647
```

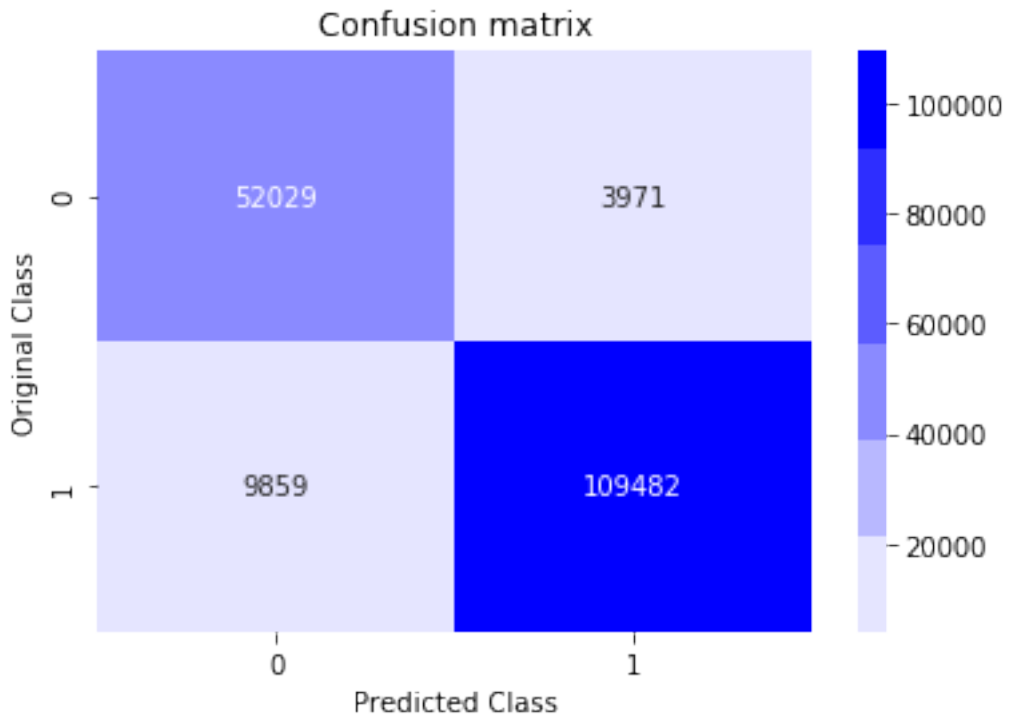
[179]:
```
cm_rfc = confusion_matrix(y_test, predict_y_te_rfc)
```

[180]:
```
tn, fp, fn, tp = cm_rfc.ravel()
```

[181]:
```
fpr_rfc = fp/(fp+tn)*100
fnr_rfc = fn/(fn+tp)*100
far_rfc = (fpr_rfc+fnr_rfc)/2
print("far:",far_rfc)
```

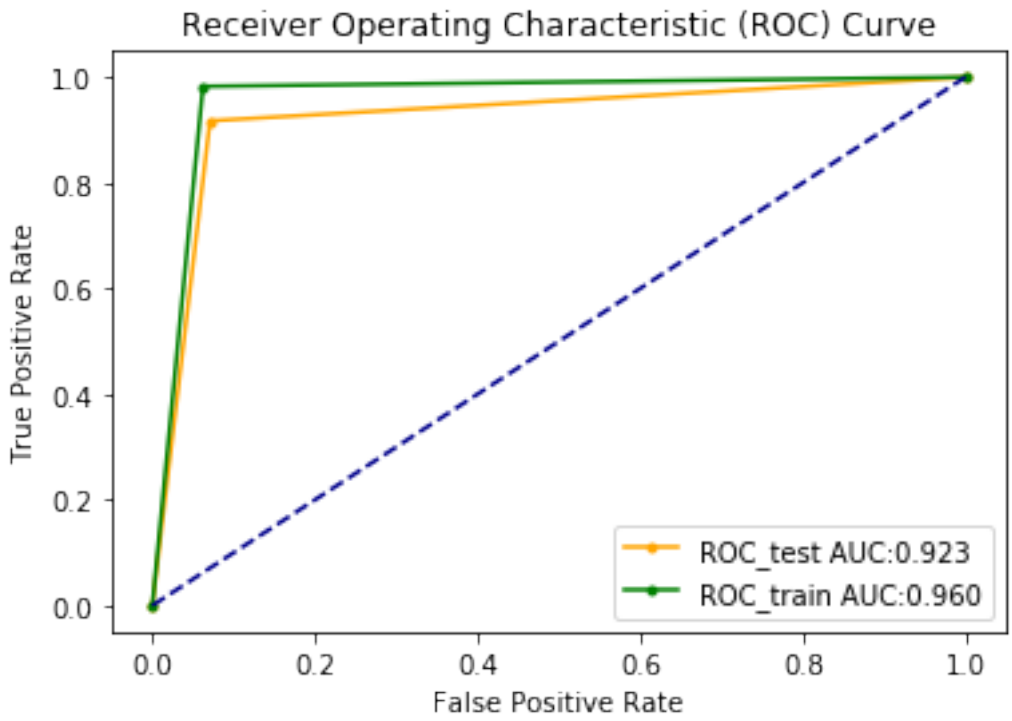
```
far: 7.676136262295199
```

[182]:
```
plot_cm(cm_rfc)
```



[183]:
```
#finding the FPR and TPR for RFC set
fpr_te_rfc, tpr_te_rfc, t_te_rfc = roc_curve(y_test, predict_y_te_rfc)
fpr_tr_rfc, tpr_tr_rfc, t_tr_rfc = roc_curve(y_train, predict_y_tr_rfc)
auc_te_rfc = auc(fpr_te_rfc, tpr_te_rfc)
print("AUC_RFC: ",auc_te_rfc)
plot_roc_curve(fpr_tr_rfc,tpr_tr_rfc,fpr_te_rfc, tpr_te_rfc)
```

```
AUC_RFC:  0.923238637377048
```



6.5 6.5 Stacking classifier

[32]:
```
clf1 = SGDClassifier(alpha=0.0001,eta0=0.001, penalty='l2', loss='log', n_jobs␣
 ↪= -1, max_iter=100)
clf1.fit(df_train, y_train)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

clf2 = SGDClassifier(alpha=0.0001,eta0=0.01, penalty='l2', loss='hinge', n_jobs␣
 ↪= -1, max_iter=100)
clf2.fit(df_train, y_train)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


clf3 = RandomForestClassifier(criterion='gini', random_state=42, n_jobs=-1,␣
 ↪max_depth=50,min_samples_split=50, n_estimators=2000)
clf3.fit(df_train, y_train)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")
```

[32]:
```
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
```

```
    sclf = StackingClassifier(estimators=[("lr",sig_clf1), ("svm",␣
↪sig_clf2),("RF", sig_clf3)], final_estimator=lr, n_jobs=-1)
    sclf.fit(df_train, y_train)
    print("Stacking Classifer : for the value of alpha: %f Log loss: %0.3f␣
↪F1-score: %0.3f" % (i, log_loss(y_test, sclf.
↪predict_proba(df_test)),f1_score(y_test, sclf.predict(df_test))))
    log_error =log_loss(y_test, sclf.predict_proba(df_test))
    if best_alpha > log_error:
        best_alpha = log_error
```

```
Stacking Classifer : for the value of alpha: 0.000100 Log loss: 0.431 F1-score:
0.918
Stacking Classifer : for the value of alpha: 0.001000 Log loss: 0.283 F1-score:
0.924
Stacking Classifer : for the value of alpha: 0.010000 Log loss: 0.245 F1-score:
0.930
Stacking Classifer : for the value of alpha: 0.100000 Log loss: 0.243 F1-score:
0.931
Stacking Classifer : for the value of alpha: 1.000000 Log loss: 0.238 F1-score:
0.932
Stacking Classifer : for the value of alpha: 10.000000 Log loss: 0.241 F1-score:
0.932
```

[35]:
```
lr = LogisticRegression(C=10)
sig_clf_sc = StackingClassifier(estimators=[("lr",sig_clf1), ("svm",␣
 ↪sig_clf2),("RF", sig_clf3)], final_estimator=lr, n_jobs=-1)
sig_clf_sc.fit(df_train, y_train)
predict_y_tr_sc= sig_clf_sc.predict(df_train)
predict_y_te_sc = sig_clf_sc.predict(df_test)
sc_f1 = f1_score(y_test, predict_y_te_sc)
print(sc_f1)
```

```
0.9318461928445934
```

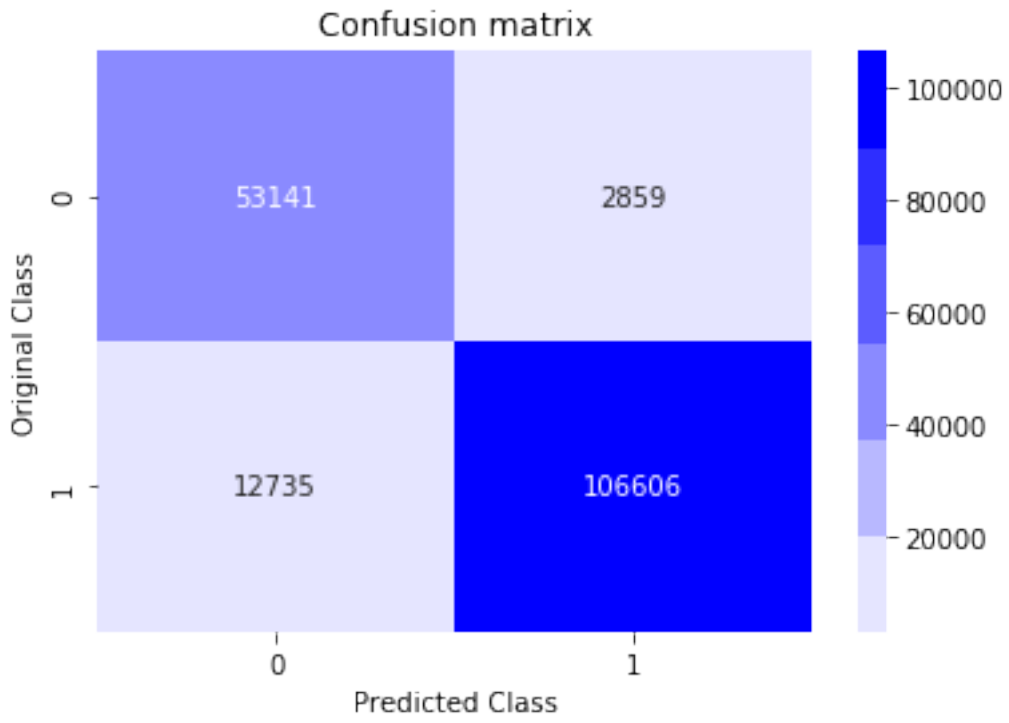
[36]:
```
cm_sc = confusion_matrix(y_test, predict_y_te_sc)
```

[37]:
```
tn, fp, fn, tp = cm_sc.ravel()
```

[38]:
```
fpr_sc = fp/(fp+tn)*100
fnr_sc = fn/(fn+tp)*100
far_sc = (fpr_sc+fnr_sc)/2
print("far:",far_sc)
```

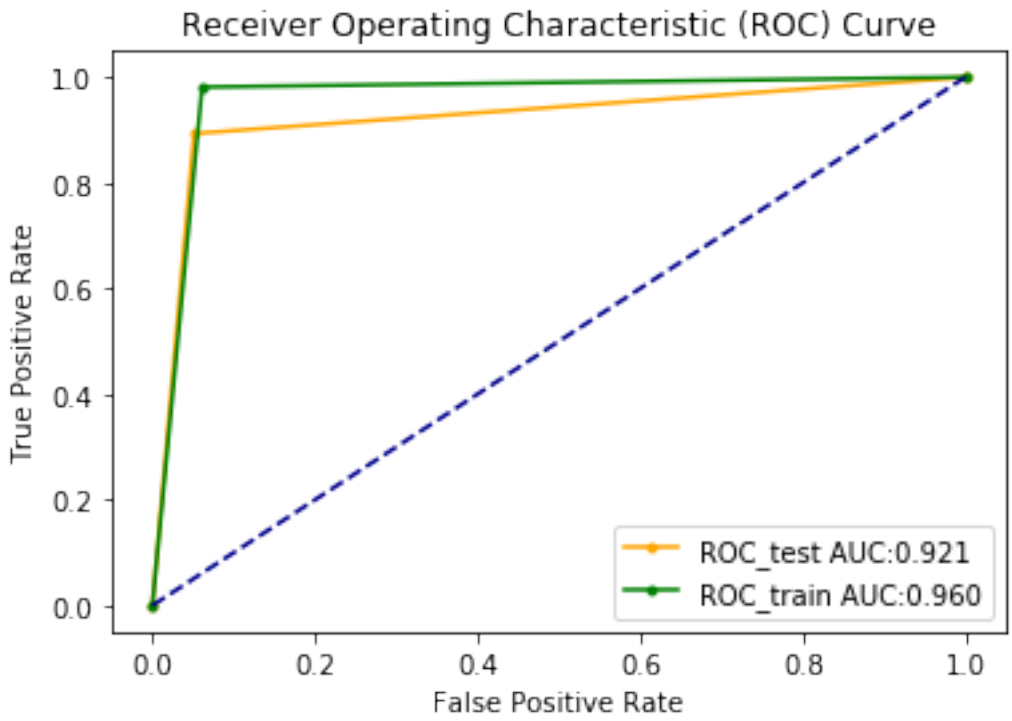
```
far: 7.88822963937672
```

[41]:
```
plot_cm(cm_sc)
```



[44]:
```
#finding the FPR and TPR for RFC set
fpr_te_sc, tpr_te_sc, t_te_sc = roc_curve(y_test, predict_y_te_sc)
fpr_tr_sc, tpr_tr_sc, t_tr_sc = roc_curve(y_train, predict_y_tr_sc)
auc_te_sc = auc(fpr_te_sc, tpr_te_sc)
print("AUC_SC: ",auc_te_sc)
plot_roc_curve(fpr_tr_sc,tpr_tr_sc,fpr_te_sc, tpr_te_sc)
```

```
AUC_SC:  0.9211177036062328
```



6.6 6.6 Model Evaluation

[45]:
```
x = PrettyTable()
x.field_names = ["Model", "F1 Score", "AUC","FPR %","FNR %","FAR %"]
x.add_row(["Logistic Regression", "{0:.4}".format(lr_f1), "{0:.4}".
 ↪format(auc_te_lr),"%.2f" % float(fpr_lr),"%.2f" % float(fnr_lr),"%.2f" %␣
 ↪float(far_lr)])
x.add_row(["Linear SVM", "{0:.4}".format(svm_f1), "{0:.4}".
 ↪format(auc_te_svm),"%.2f" % float(fpr_svm),"%.2f" % float(fnr_svm),"%.2f" %␣
 ↪float(far_svm)])
x.add_row(["Random Forest", "{0:.4}".format(rfc_f1), "{0:.4}".
 ↪format(auc_te_rfc),"%.2f" % float(fpr_rfc),"%.2f" % float(fnr_rfc),"%.2f" %␣
 ↪float(far_rfc)])
x.add_row(["Stacking Classifier", "{0:.4}".format(sc_f1), "{0:.4}".
 ↪format(auc_te_sc),"%.2f" % float(fpr_sc),"%.2f" % float(fnr_sc),"%.2f" %␣
 ↪float(far_sc)])
print(x)
```

```
+---------------------+----------+--------+-------+-------+-------+
|        Model        | F1 Score |  AUC   | FPR % | FNR % | FAR % |
+---------------------+----------+--------+-------+-------+-------+
| Logistic Regression |  0.8029  | 0.8305 |  1.43 | 32.48 | 16.95 |
|      Linear SVM     |  0.857   | 0.8638 |  3.41 | 23.82 | 13.62 |
```

```
|    Random Forest    |  0.9406  | 0.9232 |  7.09 |  8.26 |  7.68 |
| Stacking Classifier |  0.9318  | 0.9211 |  5.11 | 10.67 |  7.89 |
+---------------------+----------+--------+-------+-------+-------+
```

7 7. Machine Learning Models (One Hot Encoding)

7.1 7.1 Reading Train and Test data

[17]:
```
train_data = pd.read_csv("data/UNSW_NB15_training-set.csv")
print(train_data.shape)
train_data.head()
```

```
(82332, 45)
```

[17]:

	id	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	\
0	1	0.000011	udp	-	INT	2	0	496	0	
1	2	0.000008	udp	-	INT	2	0	1762	0	
2	3	0.000005	udp	-	INT	2	0	1068	0	
3	4	0.000006	udp	-	INT	2	0	900	0	
4	5	0.000010	udp	-	INT	2	0	2126	0	

	rate	…	ct_dst_sport_ltm	ct_dst_src_ltm	is_ftp_login	\
0	90909.0902	…	1	2	0	
1	125000.0003	…	1	2	0	
2	200000.0051	…	1	3	0	
3	166666.6608	…	1	3	0	
4	100000.0025	…	1	3	0	

	ct_ftp_cmd	ct_flw_http_mthd	ct_src_ltm	ct_srv_dst	is_sm_ips_ports	\
0	0	0	1	2	0	
1	0	0	1	2	0	
2	0	0	1	3	0	
3	0	0	2	3	0	
4	0	0	2	3	0	

	attack_cat	label
0	Normal	0
1	Normal	0
2	Normal	0
3	Normal	0
4	Normal	0

[5 rows x 45 columns]

[18]:
```
df_train = train_data[list(col_set)]
df_train.head()
```

[18]:
```
   sttl  ct_state_ttl  stcpb  trans_depth  dttl proto  dmean  \
0   254             2      0            0     0   udp      0
1   254             2      0            0     0   udp      0
2   254             2      0            0     0   udp      0
3   254             2      0            0     0   udp      0
4   254             2      0            0     0   udp      0

   ct_dst_sport_ltm  dinpkt  ackdat  …  response_body_len  djit  swin  \
0                 1     0.0     0.0  …                 0   0.0     0
1                 1     0.0     0.0  …                 0   0.0     0
2                 1     0.0     0.0  …                 0   0.0     0
3                 1     0.0     0.0  …                 0   0.0     0
4                 1     0.0     0.0  …                 0   0.0     0

   dload  ct_flw_http_mthd  ct_src_dport_ltm  ct_dst_ltm is_ftp_login dloss  \
0    0.0                 0                 1           1            0     0
1    0.0                 0                 1           1            0     0
2    0.0                 0                 1           1            0     0
3    0.0                 0                 2           2            0     0
4    0.0                 0                 2           2            0     0

   ct_dst_src_ltm
0               2
1               2
2               3
3               3
4               3

[5 rows x 30 columns]
```

[19]:
```
cat_features = df_train.select_dtypes(include=['category', object]).columns
cat_features
```

[19]:
```
Index(['proto', 'state', 'service'], dtype='object')
```

[20]:
```
ohe = OneHotEncoder()
cat_f = pd.DataFrame(ohe.fit_transform(train_data[cat_features]).toarray())
```

[21]:
```
cat_f.shape
```

[21]:
```
(82332, 151)
```

[22]:
```
df_train = df_train.drop(cat_features, axis=1)
df_train.shape
```

[22]:
```
(82332, 27)
```

[23]:
```
df_train = df_train.join(cat_f)
```

[24]:
```
df_train.head()
```

[24]:

	sttl	ct_state_ttl	stcpb	trans_depth	dttl	dmean	ct_dst_sport_ltm	\
0	254	2	0	0	0	0	1	
1	254	2	0	0	0	0	1	
2	254	2	0	0	0	0	1	
3	254	2	0	0	0	0	1	
4	254	2	0	0	0	0	1	

	dinpkt	ackdat	sloss	…	141	142	143	144	145	146	147	148	149	\
0	0.0	0.0	0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

	150
0	0.0
1	0.0
2	0.0
3	0.0
4	0.0

[5 rows x 178 columns]

[25]:
```
from sklearn.feature_selection import SelectKBest, chi2
X_new = SelectKBest(chi2, k=150).fit_transform(df_train, train_data['label'])
```

[26]:
```
X_new.shape
```

[26]: (82332, 150)

[27]:
```
test_data = pd.read_csv("data/UNSW_NB15_testing-set.csv")
print(test_data.shape)
test_data.head()
```

(175341, 45)

[27]:

	id	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	rate	\
0	1	0.121478	tcp	-	FIN	6	4	258	172	74.087490	
1	2	0.649902	tcp	-	FIN	14	38	734	42014	78.473372	
2	3	1.623129	tcp	-	FIN	8	16	364	13186	14.170161	
3	4	1.681642	tcp	ftp	FIN	12	12	628	770	13.677108	
4	5	0.449454	tcp	-	FIN	10	6	534	268	33.373826	

```
   …  ct_dst_sport_ltm  ct_dst_src_ltm  is_ftp_login  ct_ftp_cmd  \
0  …                 1               1             0           0
1  …                 1               2             0           0
2  …                 1               3             0           0
3  …                 1               3             1           1
4  …                 1              40             0           0

   ct_flw_http_mthd  ct_src_ltm  ct_srv_dst  is_sm_ips_ports  attack_cat  \
0                 0           1           1                0      Normal
1                 0           1           6                0      Normal
2                 0           2           6                0      Normal
3                 0           2           1                0      Normal
4                 0           2          39                0      Normal

   label
0      0
1      0
2      0
3      0
4      0

[5 rows x 45 columns]
```

[28]:
```
df_test = test_data[list(col_set)]
df_test.head()
```

[28]:
```
   sttl  ct_state_ttl       stcpb  trans_depth  dttl proto  dmean  \
0   252             0   621772692            0   254   tcp     43
1    62             1  1417884146            0   252   tcp   1106
2    62             1  2116150707            0   252   tcp    824
3    62             1  1107119177            0   252   tcp     64
4   254             1  2436137549            0   252   tcp     45

   ct_dst_sport_ltm      dinpkt    ackdat  …  response_body_len  \
0                 1    8.375000  0.000000  …                  0
1                 1   15.432865  0.000000  …                  0
2                 1  102.737203  0.050439  …                  0
3                 1   90.235726  0.000000  …                  0
4                 1   75.659602  0.057234  …                  0

          djit  swin         dload  ct_flw_http_mthd  ct_src_dport_ltm  \
0    11.830604   255    8495.365234                 0                 1
1  1387.778330   255  503571.312500                 0                 1
2 11420.926230   255   60929.230470                 0                 1
3  4991.784669   255    3358.622070                 0                 1
4   115.807000   255    3987.059814                 0                 2
```

```
   ct_dst_ltm is_ftp_login dloss  ct_dst_src_ltm
0           1            0     0               1
1           1            0    17               2
2           2            0     6               3
3           2            1     3               3
4           2            0     1              40

[5 rows x 30 columns]
```

[29]:
```
cat_feature = df_test.select_dtypes(include=['category', object]).columns
cat_feature
```

[29]:
```
Index(['proto', 'state', 'service'], dtype='object')
```

[30]:
```
ohe = OneHotEncoder()
cat_f_t = pd.DataFrame(ohe.fit_transform(df_test[cat_feature]).toarray())
```

[31]:
```
cat_f_t.shape
```

[31]:
```
(175341, 155)
```

[32]:
```
df_test = df_test.drop(cat_features, axis=1)
df_test.shape
```

[32]:
```
(175341, 27)
```

[33]:
```
df_test = df_test.join(cat_f_t)
```

[34]:
```
df_test.head()
```

[34]:
```
   sttl  ct_state_ttl       stcpb  trans_depth  dttl  dmean  ct_dst_sport_ltm  \
0   252             0   621772692            0   254     43                 1
1    62             1  1417884146            0   252   1106                 1
2    62             1  2116150707            0   252    824                 1
3    62             1  1107119177            0   252     64                 1
4   254             1  2436137549            0   252     45                 1

       dinpkt    ackdat  sloss  …  145  146  147  148  149  150  151  152  \
0    8.375000  0.000000      0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
1   15.432865  0.000000      2  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
2  102.737203  0.050439      1  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
3   90.235726  0.000000      1  …  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
4   75.659602  0.057234      2  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

   153  154
0  0.0  0.0
1  0.0  0.0
```

```
2  0.0  0.0
3  0.0  0.0
4  0.0  0.0

[5 rows x 182 columns]
```

[35]:
```
from sklearn.feature_selection import SelectKBest, chi2
X_new_test = SelectKBest(chi2, k=150).fit_transform(df_test, test_data['label'])
```

[36]:
```
X_new_test.shape
```

[36]:
```
(175341, 150)
```

7.1.1 7.1.1 Standardize the data

[37]:
```
std_scaler = preprocessing.MinMaxScaler()
std_scaler.fit(X_new)
x_scaled = std_scaler.transform(X_new)
df_train = pd.DataFrame(x_scaled)
x_scaled_test = std_scaler.transform(X_new_test)
df_test = pd.DataFrame(x_scaled_test)
```

[38]:
```
df_train.head()
```

[38]:
```
        0         1    2    3    4    5    6    7    8    9   …  140  141  \
0  0.996078  0.333333  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  1.0
1  0.996078  0.333333  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  1.0
2  0.996078  0.333333  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  1.0
3  0.996078  0.333333  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  1.0
4  0.996078  0.333333  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  1.0

   142  143  144  145  146  147  148  149
0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0
1  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0
2  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0
3  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0
4  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0

[5 rows x 150 columns]
```

[39]:
```
y_train = train_data['label']
y_test = test_data['label']
print("train data shape", df_train.shape, y_train.shape)
print("test data shape", df_test.shape, y_test.shape)
```

```
train data shape (82332, 150) (82332,)
test data shape (175341, 150) (175341,)
```

7.2 7.2 Logistic Regression Model

[180]:
```
prams={
    'alpha':[10 ** x for x in range(-4, 1)],
     'max_iter':[5, 10, 20, 50, 100],
    'eta0': [10 ** x for x in range(-4, 1)]
}
lr_cfl=GridSearchCV(SGDClassifier(penalty='l2', loss='log', n_jobs = -1),␣
 ↪param_grid=prams,verbose=10,n_jobs=-1)
lr_cfl.fit(df_train,y_train)
```

```
Fitting 5 folds for each of 125 candidates, totalling 625 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    1.2s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:    2.5s
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:    3.3s
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    5.1s
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    6.7s
[Parallel(n_jobs=-1)]: Done  45 tasks      | elapsed:    8.6s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:   11.0s
[Parallel(n_jobs=-1)]: Done  69 tasks      | elapsed:   12.9s
[Parallel(n_jobs=-1)]: Done  82 tasks      | elapsed:   15.7s
[Parallel(n_jobs=-1)]: Done  97 tasks      | elapsed:   18.4s
[Parallel(n_jobs=-1)]: Done 112 tasks      | elapsed:   20.6s
[Parallel(n_jobs=-1)]: Done 129 tasks      | elapsed:   23.6s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:   26.5s
[Parallel(n_jobs=-1)]: Done 165 tasks      | elapsed:   29.5s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   32.2s
[Parallel(n_jobs=-1)]: Done 205 tasks      | elapsed:   35.6s
[Parallel(n_jobs=-1)]: Done 226 tasks      | elapsed:   38.9s
[Parallel(n_jobs=-1)]: Done 249 tasks      | elapsed:   43.4s
[Parallel(n_jobs=-1)]: Done 272 tasks      | elapsed:   47.8s
[Parallel(n_jobs=-1)]: Done 297 tasks      | elapsed:   52.0s
[Parallel(n_jobs=-1)]: Done 322 tasks      | elapsed:   56.2s
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:  1.0min
[Parallel(n_jobs=-1)]: Done 376 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done 405 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done 465 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done 496 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done 529 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done 562 tasks      | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done 597 tasks      | elapsed:  1.7min
[Parallel(n_jobs=-1)]: Done 625 out of 625 | elapsed:  1.8min finished
```

[180]:
```
GridSearchCV(cv=None, error_score=nan,
             estimator=SGDClassifier(alpha=0.0001, average=False,
```

```
                                   class_weight=None, early_stopping=False,
                                   epsilon=0.1, eta0=0.0, fit_intercept=True,
                                   l1_ratio=0.15, learning_rate='optimal',
                                   loss='log', max_iter=1000,
                                   n_iter_no_change=5, n_jobs=-1,
                                   penalty='l2', power_t=0.5,
                                   random_state=None, shuffle=True, tol=0.001,
                                   validation_fraction=0.1, verbose=0,
                                   warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1],
                         'eta0': [0.0001, 0.001, 0.01, 0.1, 1],
                         'max_iter': [5, 10, 20, 50, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=10)
```

[181]:
```
results = pd.DataFrame.from_dict(lr_cfl.cv_results_)
results = results.sort_values(['rank_test_score'])
results.head()
```

[181]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	param_eta0	param_max_iter	params
13	1.402684	0.131904	0.020544	0.008756	0.0001	0.01	50	{'alpha': 0.0001, 'eta0': 0.01, 'max_iter': 50}
16	1.436356	0.037683	0.018948	0.002602	0.0001	0.1	10	{'alpha': 0.0001, 'eta0': 0.1, 'max_iter': 10}
11	1.397862	0.072990	0.018155	0.001464	0.0001	0.01	10	{'alpha': 0.0001, 'eta0': 0.01, 'max_iter': 10}
12	1.544993	0.075344	0.016157	0.004106	0.0001	0.01	20	{'alpha': 0.0001, 'eta0': 0.01, 'max_iter': 20}
17	1.420799	0.126149	0.015758	0.001716	0.0001	0.1	20	{'alpha': 0.0001, 'eta0': 0.1, 'max_iter': 20}

	split0_test_score	split1_test_score	split2_test_score	split3_test_score	split4_test_score	mean_test_score	std_test_score
13	0.922390	0.986458	0.870643	0.812037	0.802745	0.878855	0.068991
16	0.894941	0.968057	0.888498	0.819750	0.804992	0.875248	0.058639
11	0.927552	0.971276	0.859893	0.819446	0.792785	0.874190	0.066485
12	0.924698	0.933928	0.871918	0.818414	0.821693	0.874130	0.048973
17	0.934111	0.945102	0.862201	0.813069	0.795822	0.870061	0.060918

```
    rank_test_score
13                1
16                2
11                3
12                4
17                5
```

[182]:
```
print(lr_cfl.best_params_)
```

```
{'alpha': 0.0001, 'eta0': 0.01, 'max_iter': 50}
```

[183]:
```
logisticR=SGDClassifier(alpha=lr_cfl.best_params_['alpha'],eta0=lr_cfl.
 ↪best_params_['eta0'], penalty='l2', loss='log', n_jobs = -1, max_iter=lr_cfl.
 ↪best_params_['max_iter'])
logisticR.fit(df_train,y_train)
sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
sig_clf.fit(df_train, y_train)
predict_y_tr_lr = sig_clf.predict(df_train)
predict_y_te_lr = sig_clf.predict(df_test)
lr_f1 = f1_score(y_test, predict_y_te_lr)
print(lr_f1)
```

```
0.9306771659712835
```

[184]:
```
cm_lr = confusion_matrix(y_test, predict_y_te_lr)
```

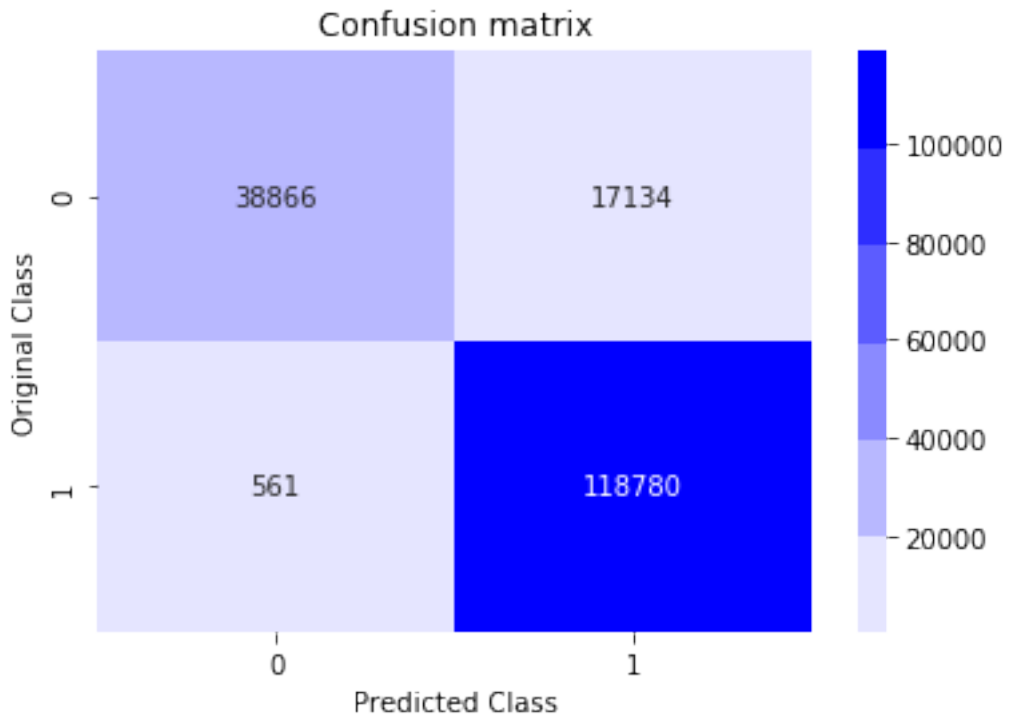
[185]:
```
tn, fp, fn, tp = cm_lr.ravel()
```

[186]:
```
fpr_lr = (fp/(fp+tn))*100
fnr_lr = (fn/(fn+tp))*100
far_lr = (fpr_lr+fnr_lr)/2
print("FAR:",far_lr)
```

```
FAR: 15.533255051251693
```

[46]:
```
def plot_cm(cm):
    sns.heatmap(cm, annot=True, cmap=sns.light_palette("blue"), fmt="g")
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")
    plt.show()
```

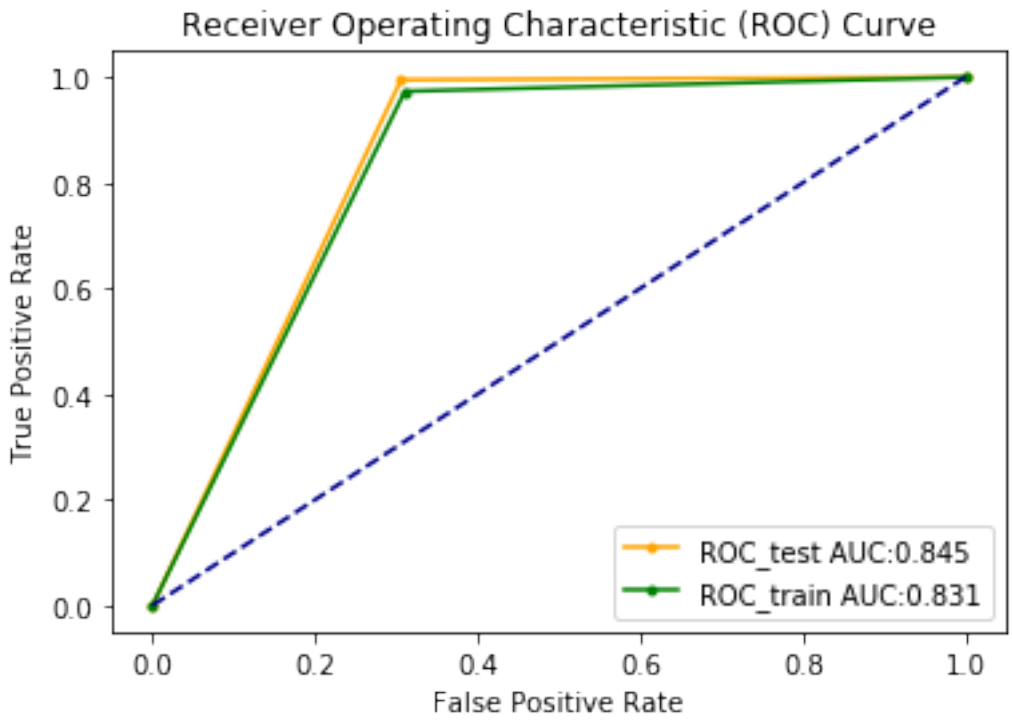
[188]:
```
plot_cm(cm_lr)
```



[47]:
```
from sklearn.metrics import roc_curve, auc
def plot_roc_curve(fpr_tr, tpr_tr,fpr_te, tpr_te):
    '''
    plot the ROC curve for the FPR and TPR value
    '''
    plt.plot(fpr_te, tpr_te, 'k.-', color='orange', label='ROC_test AUC:%.3f'%␣
 ↪auc(fpr_te, tpr_te))
    plt.plot(fpr_tr, tpr_tr, 'k.-', color='green', label='ROC_train AUC:%.3f'%␣
 ↪auc(fpr_tr, tpr_tr))
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()
```

[189]:
```
#finding the FPR and TPR for logistic reg model set
fpr_te_lr, tpr_te_lr, t_te_lr = roc_curve(y_test, predict_y_te_lr)
fpr_tr_lr, tpr_tr_lr, t_tr_lr = roc_curve(y_train, predict_y_tr_lr)
auc_te_lr = auc(fpr_te_lr, tpr_te_lr)
print("AUC_LR: ",auc_te_lr)
plot_roc_curve(fpr_tr_lr,tpr_tr_lr,fpr_te_lr, tpr_te_lr)
```

```
AUC_LR:  0.8446674494874831
```



7.3 7.3 Support Vector Machine Model

[199]:
```
prams={
    'alpha':[10 ** x for x in range(-5, 1)],
     'max_iter':[5, 10, 20, 50, 100],
    'eta0': [10 ** x for x in range(-5, 1)]
}
svm_cfl=GridSearchCV(SGDClassifier(penalty='l2', loss='hinge', n_jobs = -1),␣
 ↪param_grid=prams,verbose=10,n_jobs=-1)
svm_cfl.fit(df_train,y_train)
```

```
Fitting 5 folds for each of 180 candidates, totalling 900 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    7.8s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:   11.9s
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:   17.5s
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   23.6s
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   26.7s
[Parallel(n_jobs=-1)]: Done  45 tasks      | elapsed:   35.4s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:   38.6s
[Parallel(n_jobs=-1)]: Done  69 tasks      | elapsed:   47.8s
[Parallel(n_jobs=-1)]: Done  82 tasks      | elapsed:   51.2s
[Parallel(n_jobs=-1)]: Done  97 tasks      | elapsed:  1.0min
```

```
[Parallel(n_jobs=-1)]: Done 112 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done 129 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done 165 tasks      | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:  1.7min
[Parallel(n_jobs=-1)]: Done 205 tasks      | elapsed:  1.8min
[Parallel(n_jobs=-1)]: Done 226 tasks      | elapsed:  1.9min
[Parallel(n_jobs=-1)]: Done 249 tasks      | elapsed:  2.0min
[Parallel(n_jobs=-1)]: Done 272 tasks      | elapsed:  2.2min
[Parallel(n_jobs=-1)]: Done 297 tasks      | elapsed:  2.3min
[Parallel(n_jobs=-1)]: Done 322 tasks      | elapsed:  2.4min
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:  2.5min
[Parallel(n_jobs=-1)]: Done 376 tasks      | elapsed:  2.6min
[Parallel(n_jobs=-1)]: Done 405 tasks      | elapsed:  2.8min
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:  2.9min
[Parallel(n_jobs=-1)]: Done 465 tasks      | elapsed:  3.0min
[Parallel(n_jobs=-1)]: Done 496 tasks      | elapsed:  3.2min
[Parallel(n_jobs=-1)]: Done 529 tasks      | elapsed:  3.3min
[Parallel(n_jobs=-1)]: Done 562 tasks      | elapsed:  3.4min
[Parallel(n_jobs=-1)]: Done 597 tasks      | elapsed:  3.6min
[Parallel(n_jobs=-1)]: Done 632 tasks      | elapsed:  3.8min
[Parallel(n_jobs=-1)]: Done 669 tasks      | elapsed:  3.9min
[Parallel(n_jobs=-1)]: Done 706 tasks      | elapsed:  4.1min
[Parallel(n_jobs=-1)]: Done 745 tasks      | elapsed:  4.2min
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:  4.4min
[Parallel(n_jobs=-1)]: Done 825 tasks      | elapsed:  4.5min
[Parallel(n_jobs=-1)]: Done 866 tasks      | elapsed:  4.7min
[Parallel(n_jobs=-1)]: Done 900 out of 900 | elapsed:  4.8min finished
```

[199]:
```
GridSearchCV(cv=None, error_score=nan,
             estimator=SGDClassifier(alpha=0.0001, average=False,
                                     class_weight=None, early_stopping=False,
                                     epsilon=0.1, eta0=0.0, fit_intercept=True,
                                     l1_ratio=0.15, learning_rate='optimal',
                                     loss='hinge', max_iter=1000,
                                     n_iter_no_change=5, n_jobs=-1,
                                     penalty='l2', power_t=0.5,
                                     random_state=None, shuffle=True, tol=0.001,
                                     validation_fraction=0.1, verbose=0,
                                     warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'alpha': [1e-05, 0.0001, 0.001, 0.01, 0.1, 1],
                         'eta0': [1e-05, 0.0001, 0.001, 0.01, 0.1, 1],
                         'max_iter': [5, 10, 20, 50, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=10)
```

[200]:
```
print(svm_cfl.best_params_)
```

```
{'alpha': 1e-05, 'eta0': 1e-05, 'max_iter': 100}
```

[201]:
```
svm=SGDClassifier(alpha=svm_cfl.best_params_['alpha'],eta0=svm_cfl.
 ↪best_params_['eta0'], penalty='l2', loss='hinge', n_jobs = -1,␣
 ↪max_iter=svm_cfl.best_params_['max_iter'])
svm.fit(df_train,y_train)
sig_clf_svm = CalibratedClassifierCV(svm, method="sigmoid")
sig_clf_svm.fit(df_train, y_train)
predict_y_tr_svm = sig_clf.predict(df_train)
predict_y_te_svm = sig_clf_svm.predict(df_test)
svm_f1 = f1_score(y_test, predict_y_te_svm)
print("F1-Score", svm_f1)
```

```
F1-Score 0.8934157791398737
```

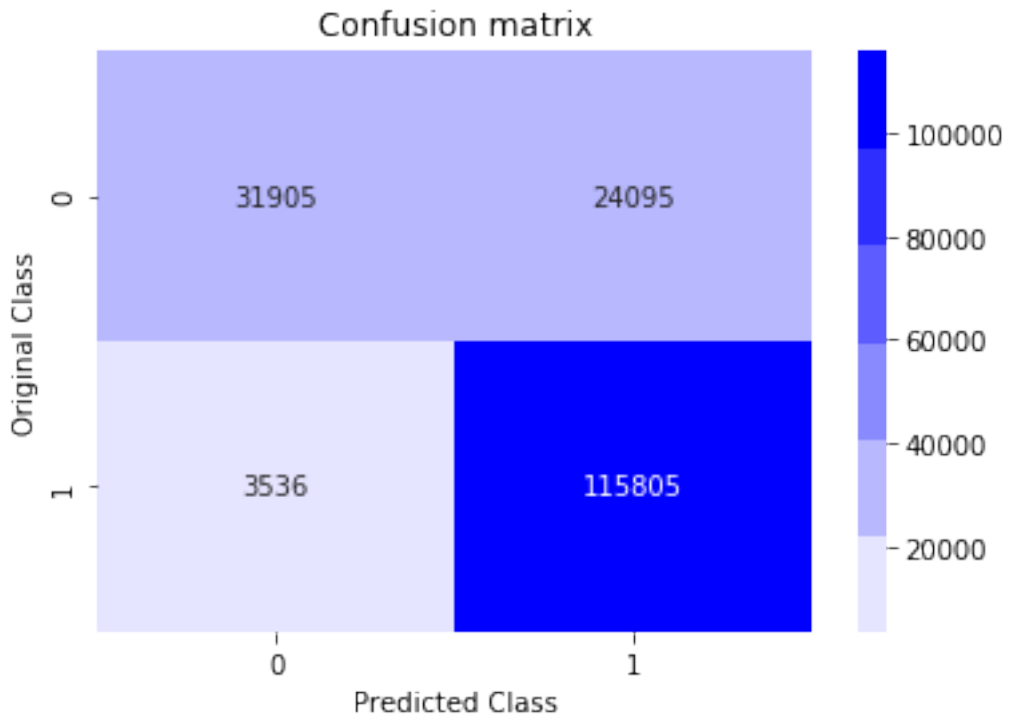
[202]:
```
cm_svm = confusion_matrix(y_test, predict_y_te_svm)
```

[203]:
```
tn, fp, fn, tp = cm_svm.ravel()
```

[204]:
```
fpr_svm = fp/(fp+tn)*100
fnr_svm = fn/(fn+tp)*100
far_svm = (fpr_svm+fnr_svm)/2
print("FAR:", far_svm)
```

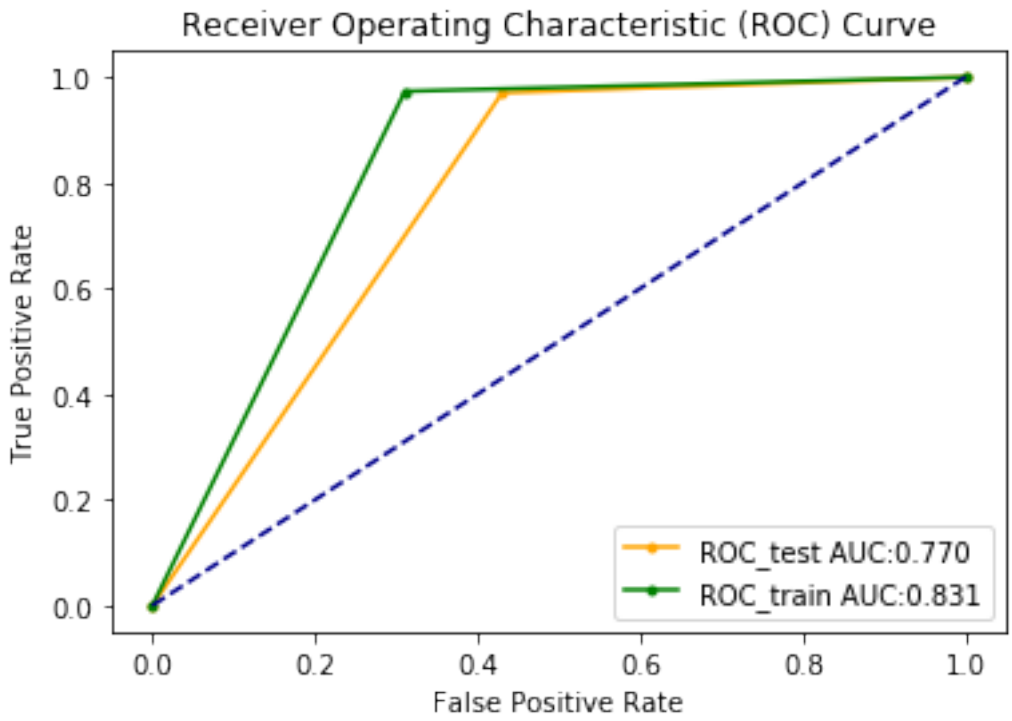
```
FAR: 22.99486192477259
```

[205]:
```
plot_cm(cm_svm)
```



[206]:
```
#finding the FPR and TPR for SVM set
fpr_te_svm, tpr_te_svm, t_te_svm = roc_curve(y_test, predict_y_te_svm)
fpr_tr_svm, tpr_tr_svm, t_tr_svm = roc_curve(y_train, predict_y_tr_svm)
auc_te_svm = auc(fpr_te_svm, tpr_te_svm)
print("AUC_SVM: ",auc_te_svm)
plot_roc_curve(fpr_tr_svm,tpr_tr_svm,fpr_te_svm, tpr_te_svm)
```

```
AUC_SVM:  0.7700513807522741
```



7.4 7.4 Random Forest Model

[207]:
```
param_grid = {"n_estimators": [10,100,500,1000, 2000],
    "min_samples_split": [50, 80, 120, 200],
              "max_depth": [3, 5, 10, 50, 100]}
rfc = RandomForestClassifier(criterion='gini', random_state=42, n_jobs=-1)
gridCV_rfc = GridSearchCV(rfc, param_grid, cv=3, verbose=10, n_jobs=-1)
gridCV_rfc.fit(df_train, y_train)
#grid Search cv results are stored in result for future use
results_rfc = pd.DataFrame.from_dict(gridCV_rfc.cv_results_)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    7.4s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:   31.7s
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:   59.4s
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:  2.0min
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  2.7min
[Parallel(n_jobs=-1)]: Done  45 tasks      | elapsed:  3.4min
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:  4.5min
[Parallel(n_jobs=-1)]: Done  69 tasks      | elapsed:  5.9min
[Parallel(n_jobs=-1)]: Done  82 tasks      | elapsed:  7.8min
[Parallel(n_jobs=-1)]: Done  97 tasks      | elapsed:  9.6min
```

```
[Parallel(n_jobs=-1)]: Done 112 tasks      | elapsed: 11.4min
[Parallel(n_jobs=-1)]: Done 129 tasks      | elapsed: 13.3min
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed: 16.4min
[Parallel(n_jobs=-1)]: Done 165 tasks      | elapsed: 19.4min
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed: 23.6min
[Parallel(n_jobs=-1)]: Done 205 tasks      | elapsed: 29.2min
[Parallel(n_jobs=-1)]: Done 226 tasks      | elapsed: 33.0min
[Parallel(n_jobs=-1)]: Done 249 tasks      | elapsed: 39.2min
[Parallel(n_jobs=-1)]: Done 272 tasks      | elapsed: 43.4min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 51.2min finished
```

[208]:
```
results_rfc = results_rfc.sort_values(['rank_test_score'])
results_rfc.head()
```

[208]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_min_samples_split	param_n_estimators	params	split0_test_score	split1_test_score	split2_test_score	mean_test_score	std_test_score	rank_test_score
83	142.432115	18.089001	9.515215	0.412303	100	50	1000	{'max_depth': 100, 'min_samples_split': 50, 'n…	0.926541	0.942355	0.896626	0.921841	0.018963	1
63	152.819562	14.167800	13.583502	1.334664	50	50	1000	{'max_depth': 50, 'min_samples_split': 50, 'n_…	0.926541	0.942282	0.896444	0.921756	0.019017	2
64	302.903203	23.717455	18.424916	2.786622	50	50	2000	{'max_depth': 50, 'min_samples_split': 50, 'n_…	0.926468	0.942355	0.896189	0.921671	0.019150	3
84	270.264782	25.018238	16.924602	1.252264	100	50	2000	{'max_depth': 100, 'min_samples_split': 50, 'n…	0.926541	0.942282	0.896116	0.921647	0.019163	4
62	81.389524	6.615336	9.358697	1.878683	50	50	500	{'max_depth': 50, 'min_samples_split': 50, 'n_…	0.926396	0.942501	0.895934	0.921610	0.019310	5

```
62                  5
```

[209]:
```
print(gridCV_rfc.best_params_)
```

```
{'max_depth': 100, 'min_samples_split': 50, 'n_estimators': 1000}
```

[210]:
```
rfc= RandomForestClassifier(criterion='gini', random_state=42, n_jobs=-1,␣
↪max_depth=gridCV_rfc.best_params_['max_depth'],min_samples_split=gridCV_rfc.
↪best_params_['min_samples_split'], n_estimators=gridCV_rfc.
↪best_params_['n_estimators'])
rfc.fit(df_train,y_train)
sig_clf_rfc = CalibratedClassifierCV(rfc, method="sigmoid")
sig_clf_rfc.fit(df_train, y_train)
predict_y_tr_rfc = sig_clf_rfc.predict(df_train)
predict_y_te_rfc = sig_clf_rfc.predict(df_test)
rfc_f1 = f1_score(y_test, predict_y_te_rfc)
print(rfc_f1)
```

```
0.9523705581088892
```

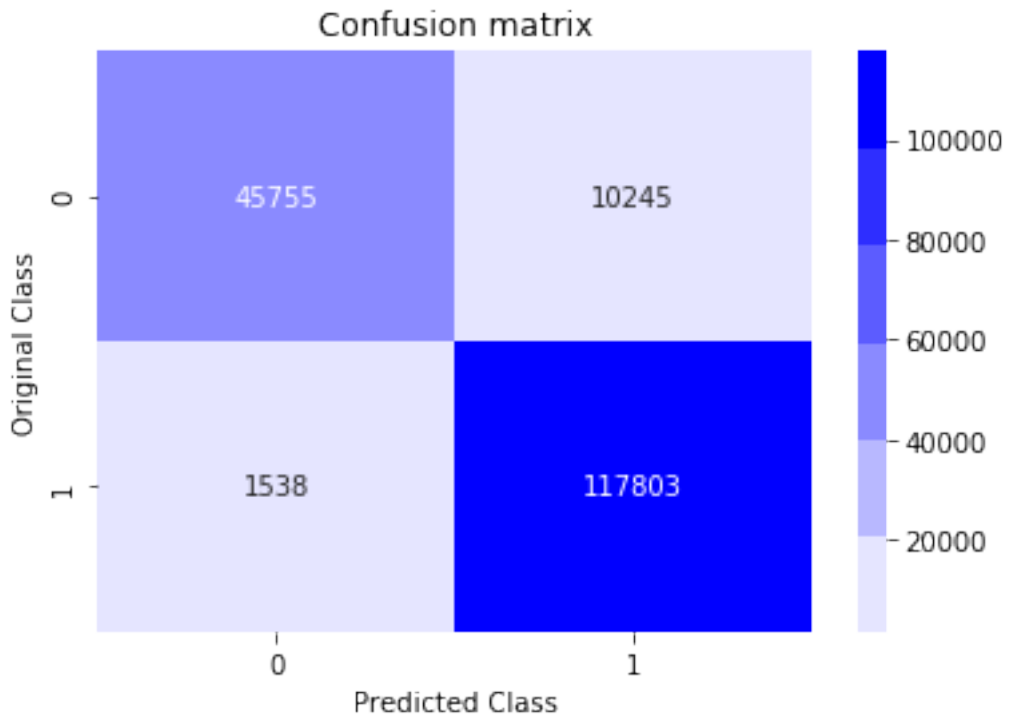
[211]:
```
cm_rfc = confusion_matrix(y_test, predict_y_te_rfc)
```

[212]:
```
tn, fp, fn, tp = cm_rfc.ravel()
```

[213]:
```
fpr_rfc = fp/(fp+tn)*100
fnr_rfc = fn/(fn+tp)*100
far_rfc = (fpr_rfc+fnr_rfc)/2
print("far:",far_rfc)
```

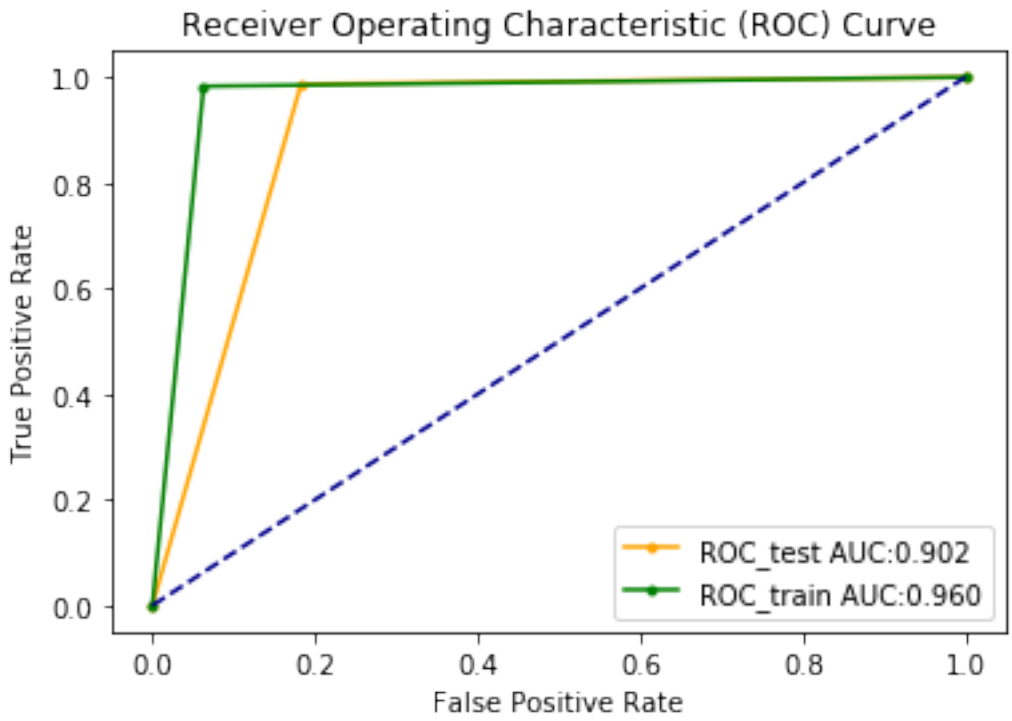
```
far: 9.791693438190922
```

[214]:
```
plot_cm(cm_rfc)
```



[215]:
```
#finding the FPR and TPR for RFC set
fpr_te_rfc, tpr_te_rfc, t_te_rfc = roc_curve(y_test, predict_y_te_rfc)
fpr_tr_rfc, tpr_tr_rfc, t_tr_rfc = roc_curve(y_train, predict_y_tr_rfc)
auc_te_rfc = auc(fpr_te_rfc, tpr_te_rfc)
print("AUC_RFC: ",auc_te_rfc)
plot_roc_curve(fpr_tr_rfc,tpr_tr_rfc,fpr_te_rfc, tpr_te_rfc)
```

```
AUC_RFC:  0.9020830656180907
```



7.5 7.5 Stacking classifier

[41]:
```
clf1 = SGDClassifier(alpha=0.0001,eta0=0.01, penalty='l2', loss='log', n_jobs =␣
↪-1, max_iter=50)
clf1.fit(df_train, y_train)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

clf2 = SGDClassifier(alpha=1e-05,eta0=1e-05, penalty='l2', loss='hinge', n_jobs␣
↪= -1, max_iter=100)
clf2.fit(df_train, y_train)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


clf3 = RandomForestClassifier(criterion='gini', random_state=42, n_jobs=-1,␣
↪max_depth=100,min_samples_split=50, n_estimators=1000)
clf3.fit(df_train, y_train)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")
```

[44]:
```
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
```

```
    sclf = StackingClassifier(estimators=[("lr",sig_clf1), ("svm",␣
↪sig_clf2),("RF", sig_clf3)], final_estimator=lr, n_jobs=-1)
    sclf.fit(df_train, y_train)
    print("Stacking Classifer : for the value of alpha: %f Log loss: %0.3f␣
↪F1-score: %0.3f" % (i, log_loss(y_test, sclf.
↪predict_proba(df_test)),f1_score(y_test, sclf.predict(df_test))))
```

```
Stacking Classifer : for the value of alpha: 0.000100 Log loss: 0.402 F1-score:
0.933
Stacking Classifer : for the value of alpha: 0.001000 Log loss: 0.258 F1-score:
0.938
Stacking Classifer : for the value of alpha: 0.010000 Log loss: 0.233 F1-score:
0.942
Stacking Classifer : for the value of alpha: 0.100000 Log loss: 0.222 F1-score:
0.943
Stacking Classifer : for the value of alpha: 1.000000 Log loss: 0.225 F1-score:
0.943
Stacking Classifer : for the value of alpha: 10.000000 Log loss: 0.225 F1-score:
0.943
```

[45]:
```
lr = LogisticRegression(C=10)
sig_clf_sc = StackingClassifier(estimators=[("lr",sig_clf1), ("svm",␣
 ↪sig_clf2),("RF", sig_clf3)], final_estimator=lr, n_jobs=-1)
sig_clf_sc.fit(df_train, y_train)
predict_y_tr_sc= sig_clf_sc.predict(df_train)
predict_y_te_sc = sig_clf_sc.predict(df_test)
sc_f1 = f1_score(y_test, predict_y_te_sc)
print(sc_f1)
```

```
0.94252901026609
```

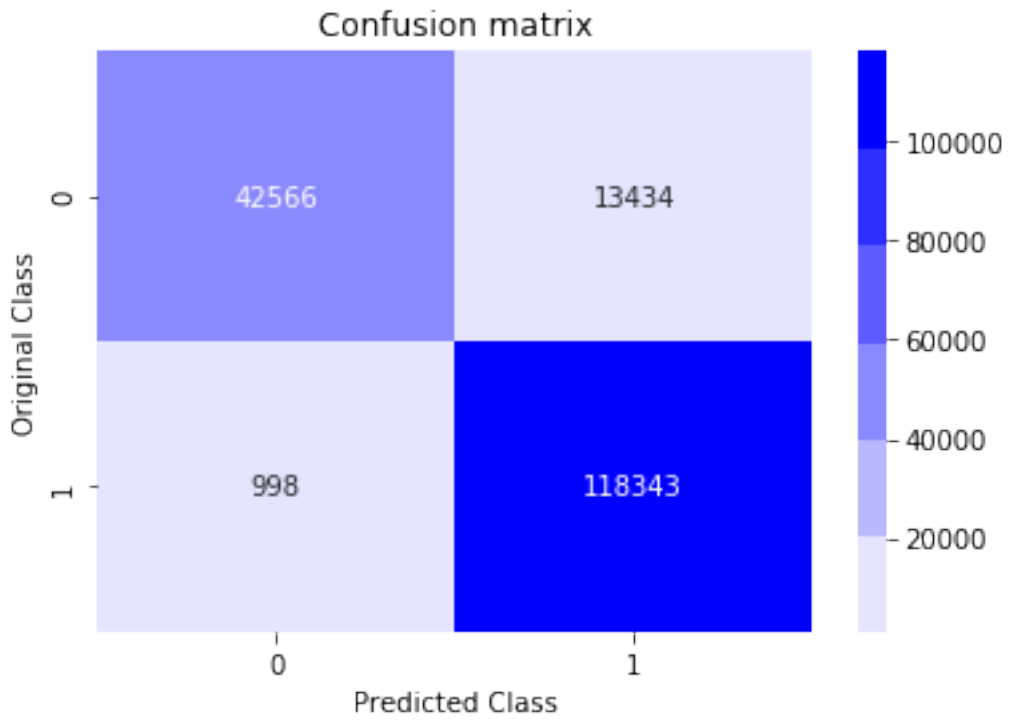
[48]:
```
cm_sc = confusion_matrix(y_test, predict_y_te_sc)
```

[49]:
```
tn, fp, fn, tp = cm_sc.ravel()
```

[50]:
```
fpr_sc = fp/(fp+tn)*100
fnr_sc = fn/(fn+tp)*100
far_sc = (fpr_sc+fnr_sc)/2
print("far:",far_sc)
```

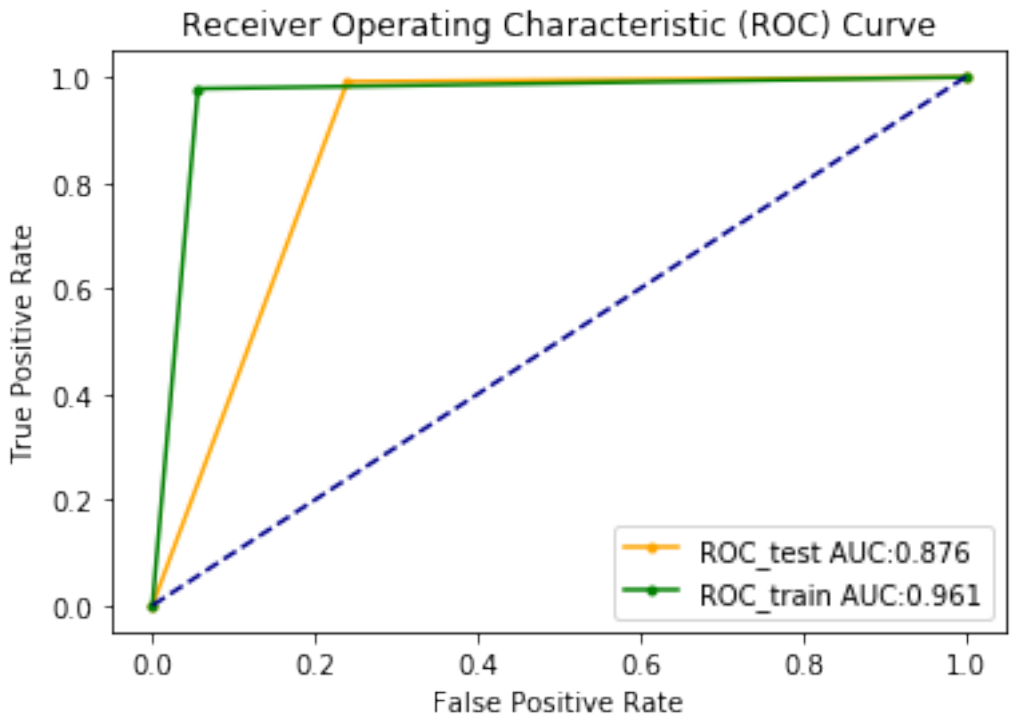
```
far: 12.412772418651475
```

[51]:
```
plot_cm(cm_sc)
```



[52]:
```
#finding the FPR and TPR for RFC set
fpr_te_sc, tpr_te_sc, t_te_sc = roc_curve(y_test, predict_y_te_sc)
fpr_tr_sc, tpr_tr_sc, t_tr_sc = roc_curve(y_train, predict_y_tr_sc)
auc_te_sc = auc(fpr_te_sc, tpr_te_sc)
print("AUC_SC: ",auc_te_sc)
plot_roc_curve(fpr_tr_sc,tpr_tr_sc,fpr_te_sc, tpr_te_sc)
```

```
AUC_SC:  0.8758722758134853
```



8 7.6. Model Evaluation

[53]:
```
x = PrettyTable()
x.field_names = ["Model", "F1 Score", "AUC","FPR %","FNR %","FAR %"]
x.add_row(["Logistic Regression", "{0:.4}".format(lr_f1), "{0:.4}".
 ↪format(auc_te_lr),"%.2f" % float(fpr_lr),"%.2f" % float(fnr_lr),"%.2f" %␣
 ↪float(far_lr)])
x.add_row(["Linear SVM", "{0:.4}".format(svm_f1), "{0:.4}".
 ↪format(auc_te_svm),"%.2f" % float(fpr_svm),"%.2f" % float(fnr_svm),"%.2f" %␣
 ↪float(far_svm)])
x.add_row(["Random Forest", "{0:.4}".format(rfc_f1), "{0:.4}".
 ↪format(auc_te_rfc),"%.2f" % float(fpr_rfc),"%.2f" % float(fnr_rfc),"%.2f" %␣
 ↪float(far_rfc)])
x.add_row(["Stacking Classifier", "{0:.4}".format(sc_f1), "{0:.4}".
 ↪format(auc_te_sc),"%.2f" % float(fpr_sc),"%.2f" % float(fnr_sc),"%.2f" %␣
 ↪float(far_sc)])
print(x)
```

```
+---------------------+----------+--------+-------+-------+-------+
|        Model        | F1 Score |  AUC   | FPR % | FNR % | FAR % |
+---------------------+----------+--------+-------+-------+-------+
| Logistic Regression |  0.9307  | 0.8447 |  30.6 |  0.47 | 15.53 |
|      Linear SVM     |  0.8934  | 0.7701 | 43.03 |  2.96 | 22.99 |
```

Random Forest	0.9524	0.9021	18.29	1.29	9.79
Stacking Classifier	0.9425	0.8759	23.99	0.84	12.41

9 8. Conclusion

- In this work, i have implemented Association rule based feature mining technique for features selection. i have used mode() selection of point for each attribute this reduces the processing time for identifying frequent value and Association Rule Mining (ARM) customized to find the highest ranked features by removing irrelevant or noisy features. Final features are than input to the machine learning model.
- To differentiate between normal and attack i have used Logistic regression, linear SVM Random forest and stacking are used.
- The experimental results show that, Stacking classifier model performed well compared to other model. it has **94% of f1 measure** and **7.3% of False Alarm Rate** which is significantly lower than other models.
- Also we can understand that Response encoding preformed well compared to other categorical data encoding techniques.
- **False negative rate** also very much low in this case the cost associated with False Negative should be very low. because an intrusion cannot be predicted as normal.
- F1-score is weighted average of precision and recall. Precision is the measure of the correctly identified intrusion from all the predicted intrusion. Recall is the measure of the correctly identified intrusion from all the actual labeled intrusion.