Import modules

In [1]:
```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
import sklearn
import imblearn
import sys

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Config
pd.set_option('display.max_columns', None)
np.set_printoptions(threshold=sys.maxsize)
np.set_printoptions(precision=3)
sns.set(style="darkgrid")
plt.rcParams['axes.labelsize'] = 14
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12

print("pandas : {0}".format(pd.__version__))
print("numpy : {0}".format(np.__version__))
print("matplotlib : {0}".format(matplotlib.__version__))
print("seaborn : {0}".format(sns.__version__))
print("sklearn : {0}".format(sklearn.__version__))
print("imblearn : {0}".format(imblearn.__version__))
```

```
pandas : 1.2.4
numpy : 1.20.1
matplotlib : 3.3.4
seaborn : 0.11.1
sklearn : 0.24.1
imblearn : 0.8.0
```

Load DataSet

In [2]:
```
# Dataset field names
datacols = ["duration","protocol_type","service","flag","src_bytes",
    "dst_bytes","land","wrong_fragment","urgent","hot","num_failed_logins",
    "logged_in","num_compromised","root_shell","su_attempted","num_root",
    "num_file_creations","num_shells","num_access_files","num_outbound_cmds"
    "is_host_login","is_guest_login","count","srv_count","serror_rate",
    "srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate",
    "diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_coun
    "dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_por
    "dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_host_srv_serro
    "dst_host_rerror_rate","dst_host_srv_rerror_rate","attack", "last_flag"]

# Load train dataset
dfkdd_train = pd.read_table("./NSL_KDD_dataset/KDDTrain.txt", sep=",", names
dfkdd_train = dfkdd_train.iloc[:,:-1] # removes an unwanted extra field

# Load test dataset
dfkdd_test = pd.read_table("./NSL_KDD_dataset/KDDTest.txt", sep=",", names=d
dfkdd_test = dfkdd_test.iloc[:,:-1]
```

Train dataset

In [3]:
```
# View train data
dfkdd_train.head(3)

# train set dimension
print('Train set dimension: {} rows, {} columns'.format(dfkdd_train.shape[0]
```

```
Train set dimension: 125973 rows, 42 columns
```

Test dataset

In [4]:
```
# View test data
dfkdd_test.head(3)

# test set dimension
print('Test set dimension: {} rows, {} columns'.format(dfkdd_test.shape[0],
```

```
Test set dimension: 22481 rows, 42 columns
```

Data Preprocessing

In [5]:
```
# Map attack type to attack group
mapping = {'ipsweep': 'Probe','satan': 'Probe','nmap': 'Probe','portsweep':
        'teardrop': 'DoS','pod': 'DoS','land': 'DoS','back': 'DoS','neptune'
        'udpstorm': 'DoS','apache2': 'DoS','processtable': 'DoS',
        'perl': 'U2R','loadmodule': 'U2R','rootkit': 'U2R','buffer_overflow'
        'sqlattack': 'U2R','httptunnel': 'U2R',
        'ftp_write': 'R2L','phf': 'R2L','guess_passwd': 'R2L','warezmaster':
        'spy': 'R2L','multihop': 'R2L','named': 'R2L','snmpguess': 'R2L','wo
        'xsnoop': 'R2L','xlock': 'R2L','sendmail': 'R2L',
        'normal': 'Normal'
        }
```

In [6]:
```
# Apply attack type mappings to the dataset
dfkdd_train['attack_class'] = dfkdd_train['attack'].apply(lambda v: mapping[
dfkdd_test['attack_class'] = dfkdd_test['attack'].apply(lambda v: mapping[v]
```

In [7]:
```
# Drop attack field from both train and test data
dfkdd_train.drop(['attack'], axis=1, inplace=True)
dfkdd_test.drop(['attack'], axis=1, inplace=True)
```

In [8]:
```
# check new train data
dfkdd_train.head()
```

Out[8]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot
0	0	tcp	ftp_data	SF	491	0	0	0	0	0
1	0	udp	other	SF	146	0	0	0	0	0
2	0	tcp	private	S0	0	0	0	0	0	0
3	0	tcp	http	SF	232	8153	0	0	0	0
4	0	tcp	http	SF	199	420	0	0	0	0

In [9]:
```
# Descriptive statistics
dfkdd_train.describe()
```

Out[9]:

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent
count	125973.00000	1.259730e+05	1.259730e+05	125973.000000	125973.000000	125973.000000
mean	287.14465	4.556674e+04	1.977911e+04	0.000198	0.022687	0.000111
std	2604.51531	5.870331e+06	4.021269e+06	0.014086	0.253530	0.014366
min	0.00000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000
25%	0.00000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000
50%	0.00000	4.400000e+01	0.000000e+00	0.000000	0.000000	0.000000
75%	0.00000	2.760000e+02	5.160000e+02	0.000000	0.000000	0.000000
max	42908.00000	1.379964e+09	1.309937e+09	1.000000	3.000000	3.000000

In [10]:
```
dfkdd_train['num_outbound_cmds'].value_counts()
dfkdd_test['num_outbound_cmds'].value_counts()
```

Out[10]:
```
0    22481
Name: num_outbound_cmds, dtype: int64
```

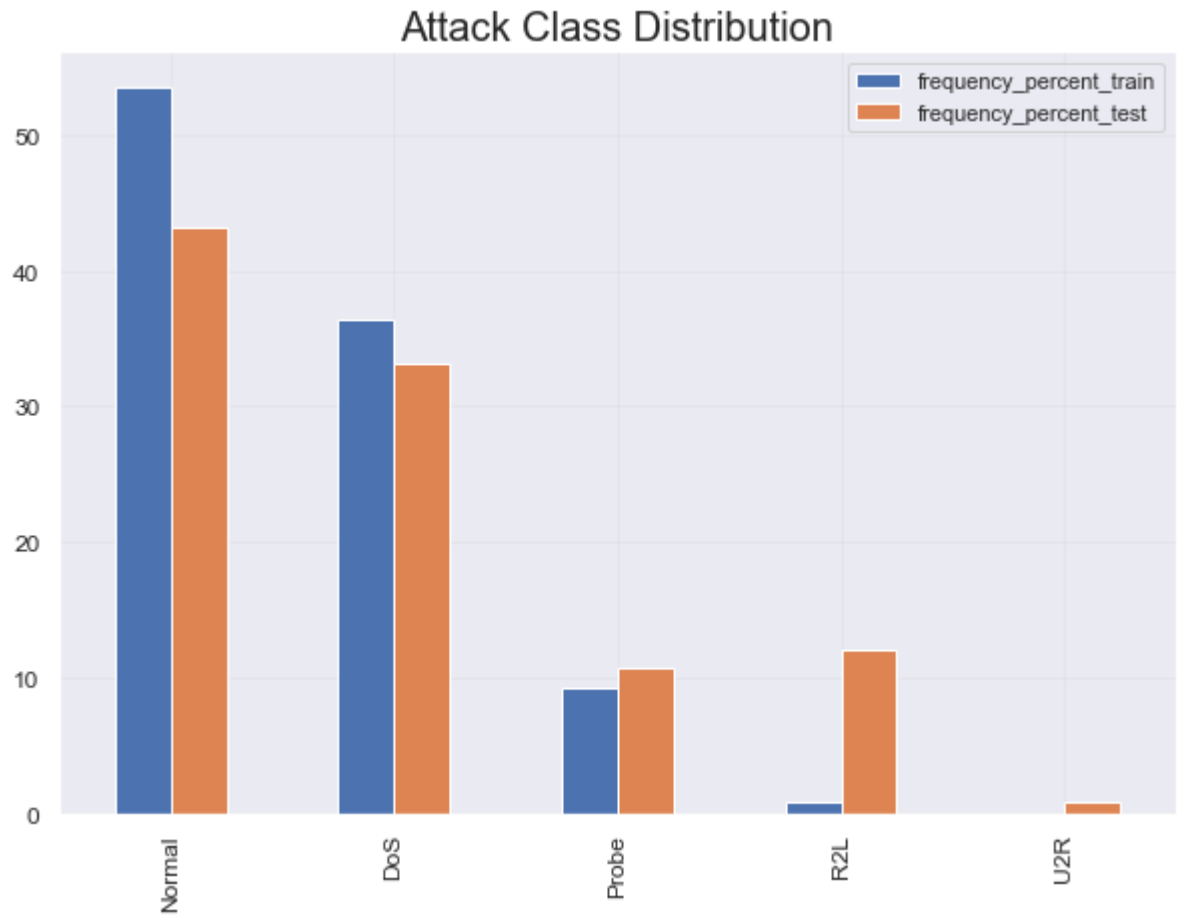
In [11]:
```
# 'num_outbound_cmds' field has value 0 for all rows. Hence, it is removed f
dfkdd_train.drop(['num_outbound_cmds'], axis=1, inplace=True)
dfkdd_test.drop(['num_outbound_cmds'], axis=1, inplace=True)
```

In [12]:
```
# checking attack group Distribution
attack_class_freq_train = dfkdd_train[['attack_class']].apply(lambda x: x.va
attack_class_freq_test = dfkdd_test[['attack_class']].apply(lambda x: x.valu
attack_class_freq_train['frequency_percent_train'] = round((100 * attack_cla
attack_class_freq_test['frequency_percent_test'] = round((100 * attack_class

attack_class_dist = pd.concat([attack_class_freq_train,attack_class_freq_tes
attack_class_dist
```

Out[12]:

	attack_class	frequency_percent_train	attack_class	frequency_percent_test
Normal	67343	53.46	9711	43.20
DoS	45927	36.46	7458	33.17
Probe	11656	9.25	2421	10.77
R2L	995	0.79	2754	12.22
U2R	52	0.04	200	0.89

In [13]:
```
# Attack group bar plot
plot = attack_class_dist[['frequency_percent_train', 'frequency_percent_test
plot.set_title("Attack Class Distribution", fontsize=20);
plot.grid('on', color='lightgray', alpha=0.25);
```



In [14]:
```
1 dfkdd_train.head()
```

Out[14]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot
0	0	tcp	ftp_data	SF	491	0	0	0	0	0
1	0	udp	other	SF	146	0	0	0	0	0
2	0	tcp	private	S0	0	0	0	0	0	0
3	0	tcp	http	SF	232	8153	0	0	0	0
4	0	tcp	http	SF	199	420	0	0	0	0

Scaling Numerical Attributes

In [15]:
```
1  from sklearn.preprocessing import StandardScaler
2  scaler = StandardScaler()
3
4  # extract numerical features and scale it to have zero mean and unit varianc
5  cols = dfkdd_train.select_dtypes(include=['float64','int64']).columns
6  sc_train = scaler.fit_transform(dfkdd_train.select_dtypes(include=['float64'
7  sc_test = scaler.fit_transform(dfkdd_test.select_dtypes(include=['float64','
8
9  # turn the result back to a dataframe
10 sc_traindf = pd.DataFrame(sc_train, columns = cols)
11 sc_testdf = pd.DataFrame(sc_test, columns = cols)
```

Encoding of Categorical Attributes

In [16]:
```
1  from sklearn.preprocessing import LabelEncoder
2  encoder = LabelEncoder()
3
4  # extract categorical features from both training and test sets
5  cattrain = dfkdd_train.select_dtypes(include=['object']).copy()
6  cattest = dfkdd_test.select_dtypes(include=['object']).copy()
7
8  # encoding
9  traincat = cattrain.apply(encoder.fit_transform)
10 testcat = cattest.apply(encoder.fit_transform)
11
12 # separate target column from encoded data
13 enctrain = traincat.drop(['attack_class'], axis=1)
14 enctest = testcat.drop(['attack_class'], axis=1)
15
16 cat_Ytrain = traincat[['attack_class']].copy()
17 cat_Ytest = testcat[['attack_class']].copy()
```

Data Sampling

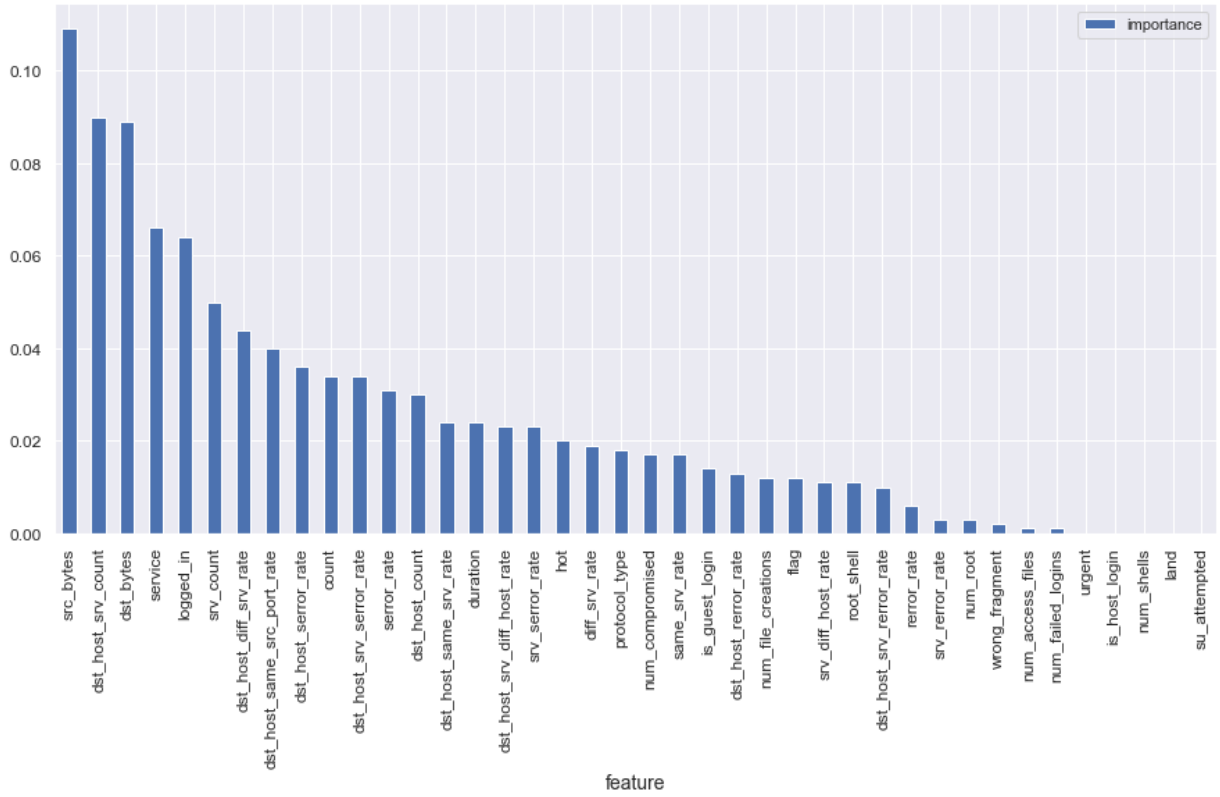
In [17]:
```
from imblearn.over_sampling import RandomOverSampler
from collections import Counter

# define columns and extract encoded train set for sampling
sc_traindf = dfkdd_train.select_dtypes(include=['float64','int64'])
refclasscol = pd.concat([sc_traindf, enctrain], axis=1).columns
refclass = np.concatenate((sc_train, enctrain.values), axis=1)
X = refclass

# reshape target column to 1D array shape
c, r = cat_Ytest.values.shape
y_test = cat_Ytest.values.reshape(c,)

c, r = cat_Ytrain.values.shape
y = cat_Ytrain.values.reshape(c,)

# apply the random over sampling
ros = RandomOverSampler(random_state=42)
X_res, y_res = ros.fit_resample(X, y)
print('Original dataset shape {}'.format(Counter(y)))
print('Resampled dataset shape {}'.format(Counter(y_res)))
```

```
Original dataset shape Counter({1: 67343, 0: 45927, 2: 11656, 3: 995, 4: 52})
Resampled dataset shape Counter({1: 67343, 0: 67343, 3: 67343, 2: 67343, 4: 673
43})
```

Feature Selection

In [18]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier();

# fit random forest classifier on the training set
rfc.fit(X_res, y_res);
# extract important features
score = np.round(rfc.feature_importances_,3)
importances = pd.DataFrame({'feature':refclasscol,'importance':score})
importances = importances.sort_values('importance',ascending=False).set_inde
```

In [19]:
```
# plot importances
plt.rcParams['figure.figsize'] = (15, 7)
importances.plot(kind='bar');
```



In [20]:
```
from sklearn.feature_selection import RFE
import itertools
rfc = RandomForestClassifier()

# create the RFE model and select 10 attributes
rfe = RFE(rfc, n_features_to_select=10)
rfe = rfe.fit(X_res, y_res)

# summarize the selection of the attributes
feature_map = [(i, v) for i, v in itertools.zip_longest(rfe.get_support(), r
selected_features = [v for i, v in feature_map if i==True]
```

In [21]:
```
selected_features =['src_bytes','dst_bytes','logged_in','count','srv_count',
```

Dataset Partition

In [22]:
```
# define columns to new dataframe
newcol = list(refclasscol)
newcol.append('attack_class')

# add a dimension to target
new_y_res = y_res[:, np.newaxis]

# create a dataframe from sampled data
res_arr = np.concatenate((X_res, new_y_res), axis=1)
res_df = pd.DataFrame(res_arr, columns = newcol)

# create test dataframe
reftest = pd.concat([sc_testdf, testcat], axis=1)
reftest['attack_class'] = reftest['attack_class'].astype(np.float64)
reftest['protocol_type'] = reftest['protocol_type'].astype(np.float64)
reftest['flag'] = reftest['flag'].astype(np.float64)
reftest['service'] = reftest['service'].astype(np.float64)

res_df.shape
reftest.shape
```

Out[22]:
```
(22481, 41)
```

In [23]:
```
from collections import defaultdict
classdict = defaultdict(list)

# create two-target classes (normal class and an attack class)
attacklist = [('DoS', 0.0), ('Probe', 2.0), ('R2L', 3.0), ('U2R', 4.0)]
normalclass = [('Normal', 1.0)]

def create_classdict():
    '''This function subdivides train and test dataset into two-class attack
    for j, k in normalclass:
        for i, v in attacklist:
            restrain_set = res_df.loc[(res_df['attack_class'] == k) | (res_d
            classdict[j +'_' + i].append(restrain_set)
            # test labels
            reftest_set = reftest.loc[(reftest['attack_class'] == k) | (reft
            classdict[j +'_' + i].append(reftest_set)

create_classdict()
```

In [24]:
```
for k, v in classdict.items():
    print(k)
```

```
Normal_DoS
Normal_Probe
Normal_R2L
Normal_U2R
```

In [25]:
```
pretrain = classdict['Normal_Probe'][0]
pretest = classdict['Normal_DoS'][1]
grpclass = 'Normal_DoS'
```

Finalize data preprocessing for training

In [26]:
```
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder(handle_unknown='ignore')

#Xresdf = pretrain
#newtest = pretest

Xresdf = res_df
newtest = reftest

Xresdfnew = Xresdf[selected_features]
Xresdfnum = Xresdfnew.drop(['service'], axis=1)
Xresdfcat = Xresdfnew[['service']].copy()

Xtest_features = newtest[selected_features]
Xtestdfnum = Xtest_features.drop(['service'], axis=1)
Xtestcat = Xtest_features[['service']].copy()


# Fit train data
enc.fit(Xresdfcat)

# Transform train data
X_train_1hotenc = enc.transform(Xresdfcat).toarray()

# Transform test data
X_test_1hotenc = enc.transform(Xtestcat).toarray()

X_train = np.concatenate((Xresdfnum.values, X_train_1hotenc), axis=1)
X_test = np.concatenate((Xtestdfnum.values, X_test_1hotenc), axis=1)

y_train = Xresdf[['attack_class']].copy()
c, r = y_train.values.shape
Y_train = y_train.values.reshape(c,)

y_test = newtest[['attack_class']].copy()
c, r = y_test.values.shape
Y_test = y_test.values.reshape(c,)
```

Train Models

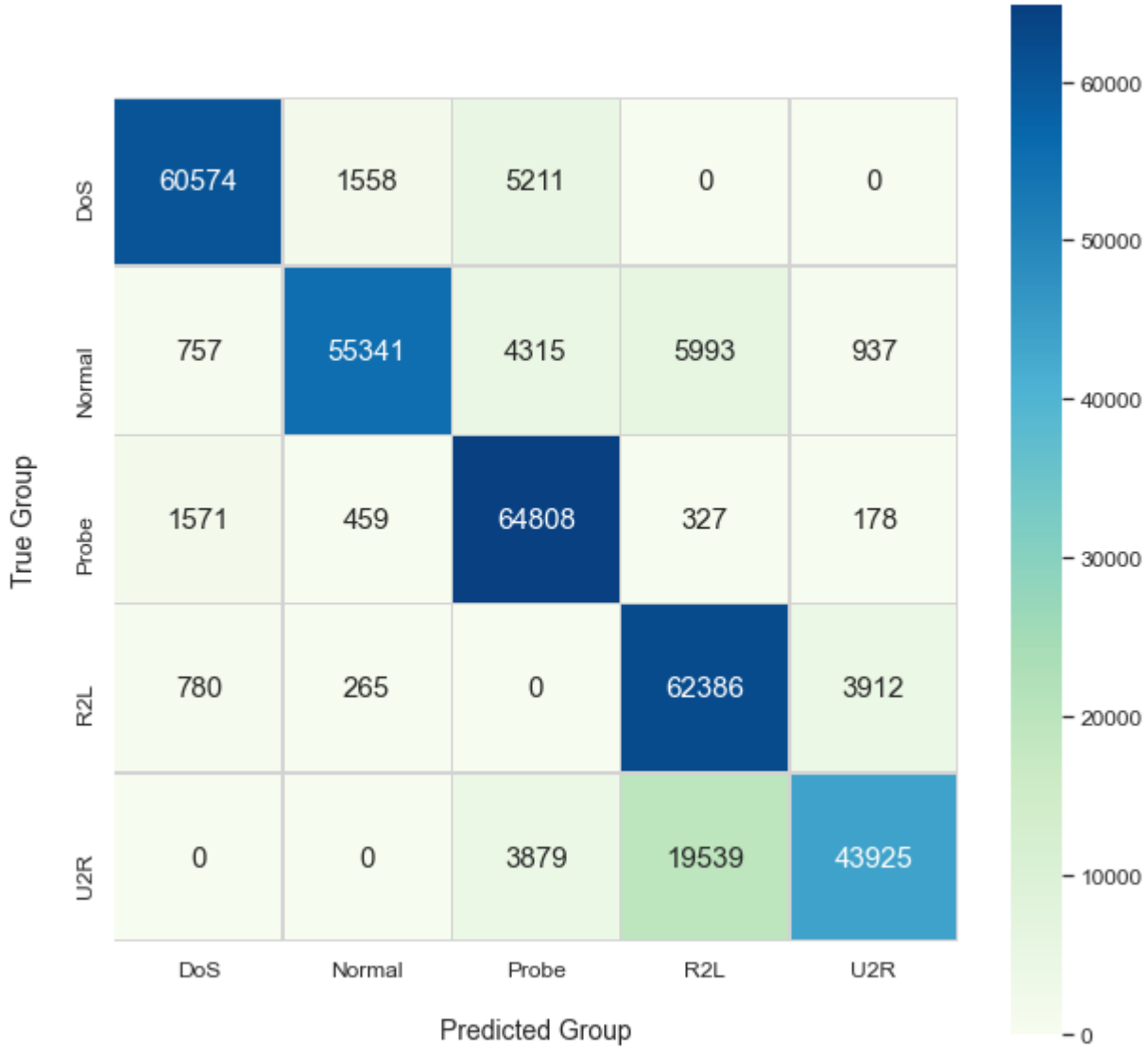
In [27]:
```
from sklearn.svm import SVC
from sklearn.naive_bayes import BernoulliNB
from sklearn import tree
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import VotingClassifier

# Train KNeighborsClassifier Model
KNN_Classifier = KNeighborsClassifier(n_jobs=-1)
KNN_Classifier.fit(X_train, Y_train);

# Train Gaussian Naive Baye Model
BNB_Classifier = BernoulliNB()
BNB_Classifier.fit(X_train, Y_train)

# Train Decision Tree Model
DTC_Classifier = tree.DecisionTreeClassifier(criterion='entropy', max_depth
DTC_Classifier.fit(X_train, Y_train);

# Train RandomForestClassifier Model
RF_Classifier = RandomForestClassifier(criterion='entropy', n_jobs=-1, n_est
RF_Classifier.fit(X_train, Y_train);

# Train Ensemble Model (This method combines all the individual models above
combined_model = [('Naive Baye Classifier', BNB_Classifier),
                  ("RF_Classifier", RF_Classifier),
                  ('Decision Tree Classifier', DTC_Classifier),
                  ('KNeighborsClassifier', KNN_Classifier),
                 ]
EnsembleClassifier =  VotingClassifier(estimators = combined_model,voting =
EnsembleClassifier.fit(X_train, Y_train);
```

Evaluate Models

In [28]:

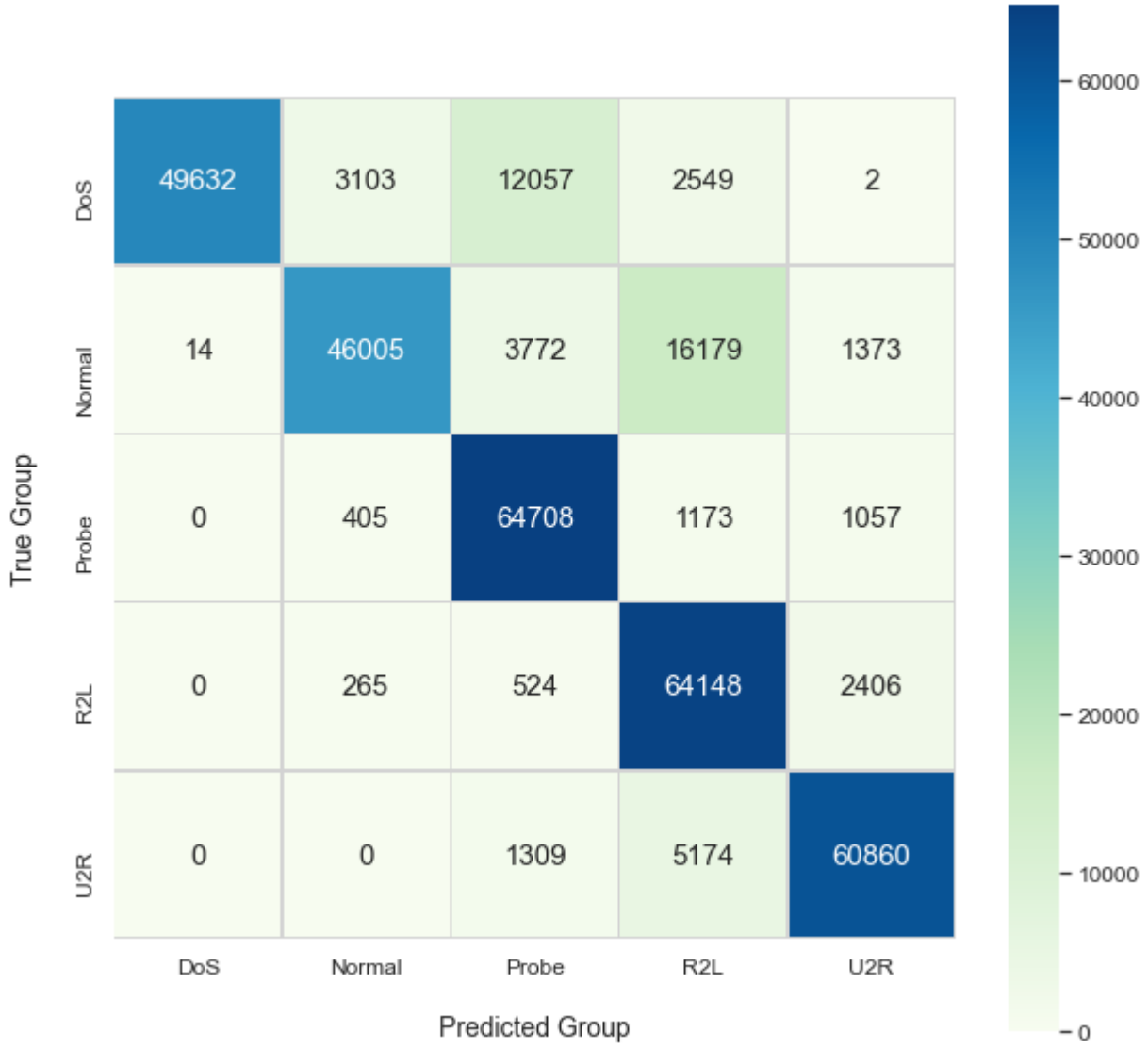
```
from sklearn import metrics
from sklearn.metrics import plot_confusion_matrix

def plotHeatmap(cm):
    fig, ax = plt.subplots(figsize=(10,10))
    heatmap = sns.heatmap(cm, annot=True, linewidths=0.5, linecolor="lightgr
    heatmap.set_xticklabels(['DoS','Normal','Probe','R2L','U2R'])
    heatmap.set_yticklabels(['DoS','Normal','Probe','R2L','U2R'])
    plt.xlabel("Predicted Group", labelpad=20)
    plt.ylabel("True Group", labelpad=20)
    plt.show()


models = []
models.append(('Naive Baye Classifier', BNB_Classifier))
models.append(('Decision Tree Classifier', DTC_Classifier))
models.append(('RandomForest Classifier', RF_Classifier))
#models.append(('KNeighborsClassifier', KNN_Classifier))
models.append(('EnsembleClassifier', EnsembleClassifier))

for i, v in models:
    Y_pred = v.predict(X_train)
    accuracy = metrics.accuracy_score(Y_train, Y_pred)
    confusion_matrix = metrics.confusion_matrix(Y_train, Y_pred, labels = v.
    classification = metrics.classification_report(Y_train, Y_pred)
    print()
    print('============================== {} Model Evaluation ==============
    print()
    print ("Model Accuracy:" "\n", accuracy)
    print()
    print("Confusion matrix:" "\n", confusion_matrix)
    plotHeatmap(confusion_matrix)
    print()
    print("Classification report:" "\n", classification)
    print()
```

```
============================== Naive Baye Classifier Model Evaluation =========
======================

Model Accuracy:
 0.8524538556345871

Confusion matrix:
 [[60574  1558  5211     0     0]
 [  757 55341  4315  5993   937]
 [ 1571   459 64808   327   178]
 [  780   265     0 62386  3912]
 [    0     0  3879 19539 43925]]
```

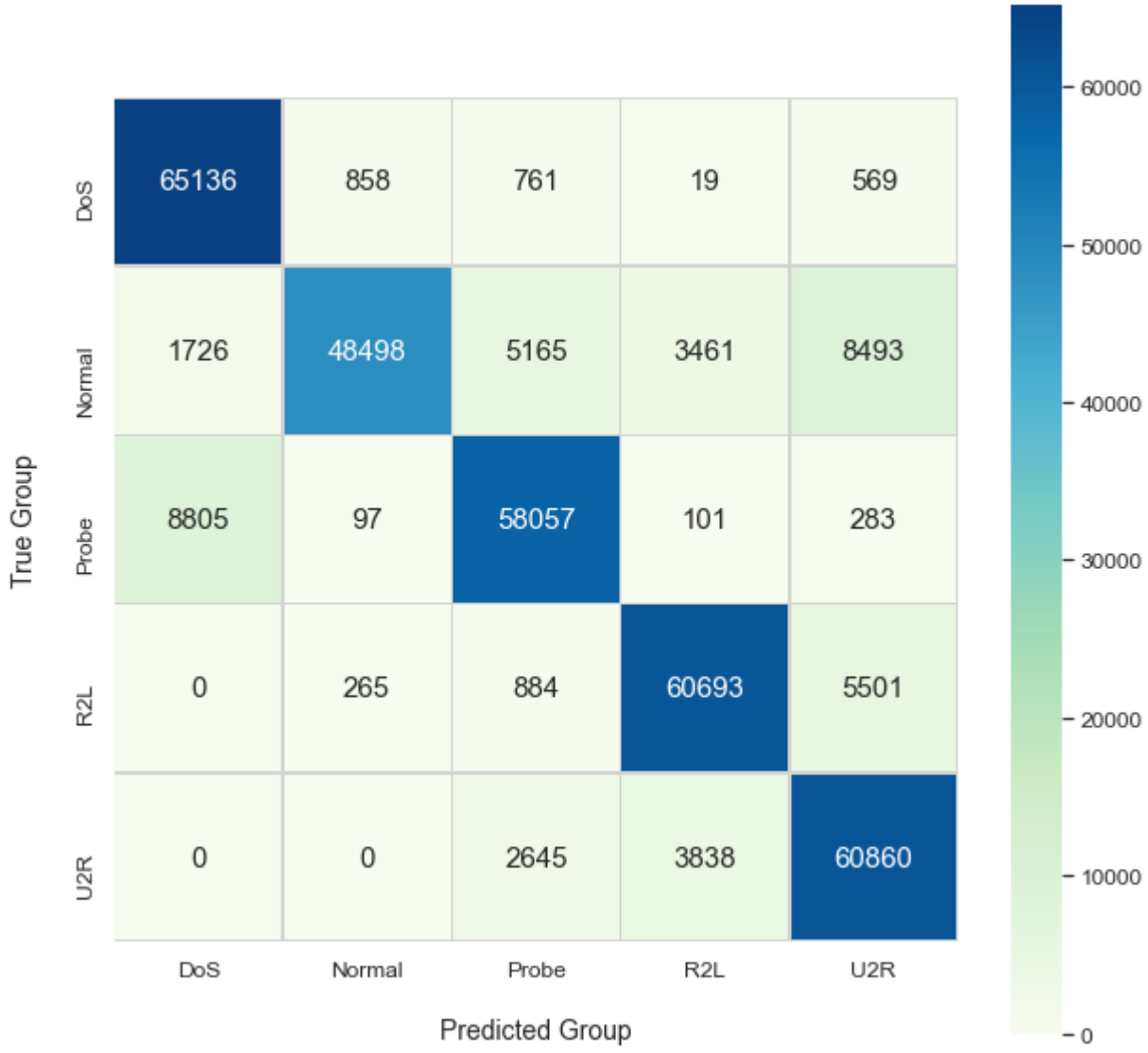



```
Classification report:
              precision    recall  f1-score   support

         0.0       0.95      0.90      0.92     67343
         1.0       0.96      0.82      0.89     67343
         2.0       0.83      0.96      0.89     67343
         3.0       0.71      0.93      0.80     67343
         4.0       0.90      0.65      0.76     67343

    accuracy                           0.85    336715
```

```
   macro avg       0.87      0.85      0.85    336715
weighted avg       0.87      0.85      0.85    336715


============================== Decision Tree Classifier Model Evaluation ====
==========================

Model Accuracy:
 0.8474615030515421

Confusion matrix:
 [[49632  3103 12057  2549     2]
 [   14 46005  3772 16179  1373]
 [    0   405 64708  1173  1057]
 [    0   265   524 64148  2406]
 [    0     0  1309  5174 60860]]
```

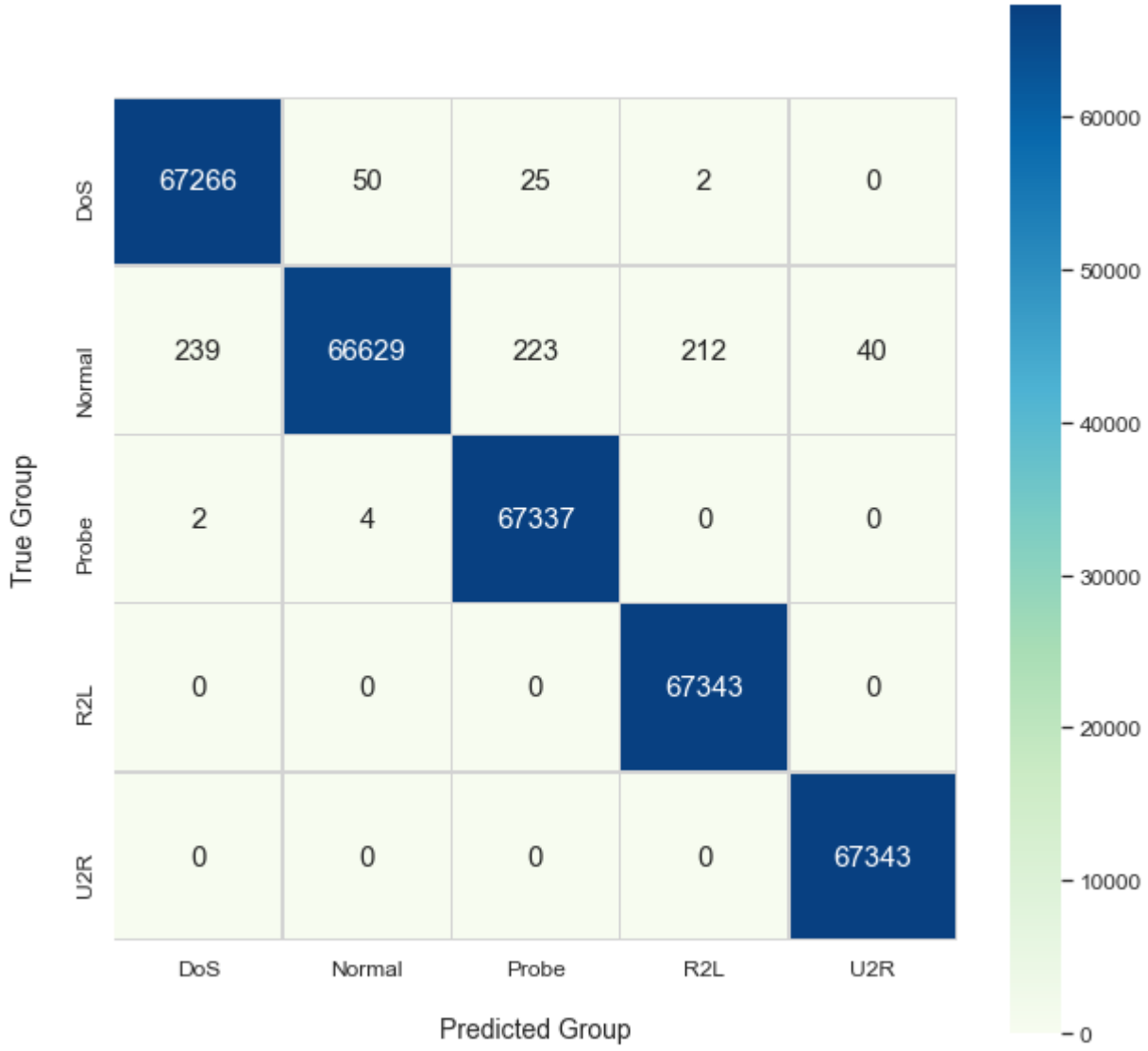


```
Classification report:
               precision    recall  f1-score   support

         0.0       1.00      0.74      0.85     67343
         1.0       0.92      0.68      0.79     67343
         2.0       0.79      0.96      0.86     67343
         3.0       0.72      0.95      0.82     67343
         4.0       0.93      0.90      0.91     67343

    accuracy                           0.85    336715
   macro avg       0.87      0.85      0.85    336715
weighted avg       0.87      0.85      0.85    336715



============================== RandomForest Classifier Model Evaluation =======
========================

Model Accuracy:
 0.8708967524464309

Confusion matrix:
 [[65136   858   761    19   569]
 [ 1726 48498  5165  3461  8493]
 [ 8805    97 58057   101   283]
 [    0   265   884 60693  5501]
 [    0     0  2645  3838 60860]]
```

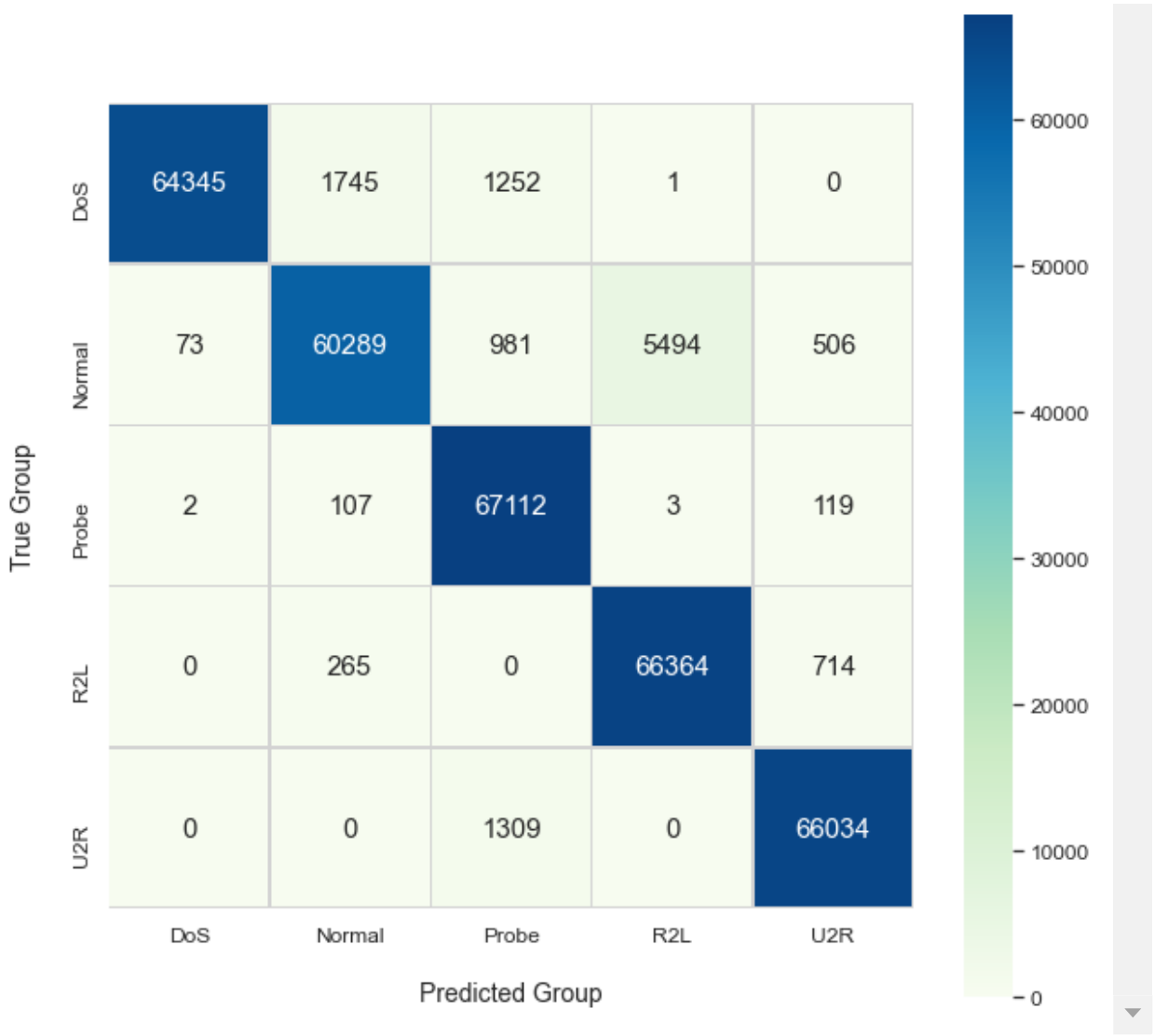



```
Classification report:
              precision    recall  f1-score   support

         0.0       0.86      0.97      0.91     67343
         1.0       0.98      0.72      0.83     67343
         2.0       0.86      0.86      0.86     67343
         3.0       0.89      0.90      0.90     67343
         4.0       0.80      0.90      0.85     67343

    accuracy                           0.87    336715
   macro avg       0.88      0.87      0.87    336715
weighted avg       0.88      0.87      0.87    336715


============================== KNeighborsClassifier Model Evaluation ==========
====================

Model Accuracy:
 0.9576330130822803
```

```
Confusion matrix:
 [[67266    50    25     2     0]
 [  239 66629   223   212    40]
 [    2     4 67337     0     0]
 [    0     0     0 67343     0]
 [    0     0     0     0 67343]]
```



```
Classification report:
               precision    recall  f1-score   support

          0.0       1.00      1.00      1.00     67343
          1.0       1.00      0.99      0.99     67343
          2.0       1.00      1.00      1.00     67343
          3.0       1.00      1.00      1.00     67343
          4.0       1.00      1.00      1.00     67343

    accuracy                           1.00    336715
   macro avg       1.00      1.00      1.00    336715
weighted avg       1.00      1.00      1.00    336715



============================== EnsembleClassifier Model Evaluation ============
==================

Model Accuracy:
 0.9626657559063303

Confusion matrix:
 [[64345  1745  1252     1     0]
 [   73 60289   981  5494   506]
 [    2   107 67112     3   119]
 [    0   265     0 66364   714]
 [    0     0  1309     0 66034]]
```




```
Classification report:
              precision    recall  f1-score   support

         0.0       1.00      0.96      0.98     67343
         1.0       0.97      0.90      0.93     67343
         2.0       0.95      1.00      0.97     67343
         3.0       0.92      0.99      0.95     67343
         4.0       0.98      0.98      0.98     67343

    accuracy                           0.96    336715
   macro avg       0.96      0.96      0.96    336715
weighted avg       0.96      0.96      0.96    336715
```

In [25]:
```
import pickle
for i, v in models:
    pickle.dump(v,open(i,'wb'))

KNN_Classifier = pickle.load(open("./models/multiclass/KNeighborsClassifier"
BNB_Classifier = pickle.load(open("./models/multiclass/Naive Baye Classifier
DTC_Classifier = pickle.load(open("./models/multiclass/Decision Tree Classif
RF_Classifier = pickle.load(open("./models/multiclass/RandomForest Classifie
EnsembleClassifier = pickle.load(open("./models/multiclass/EnsembleClassifie
```

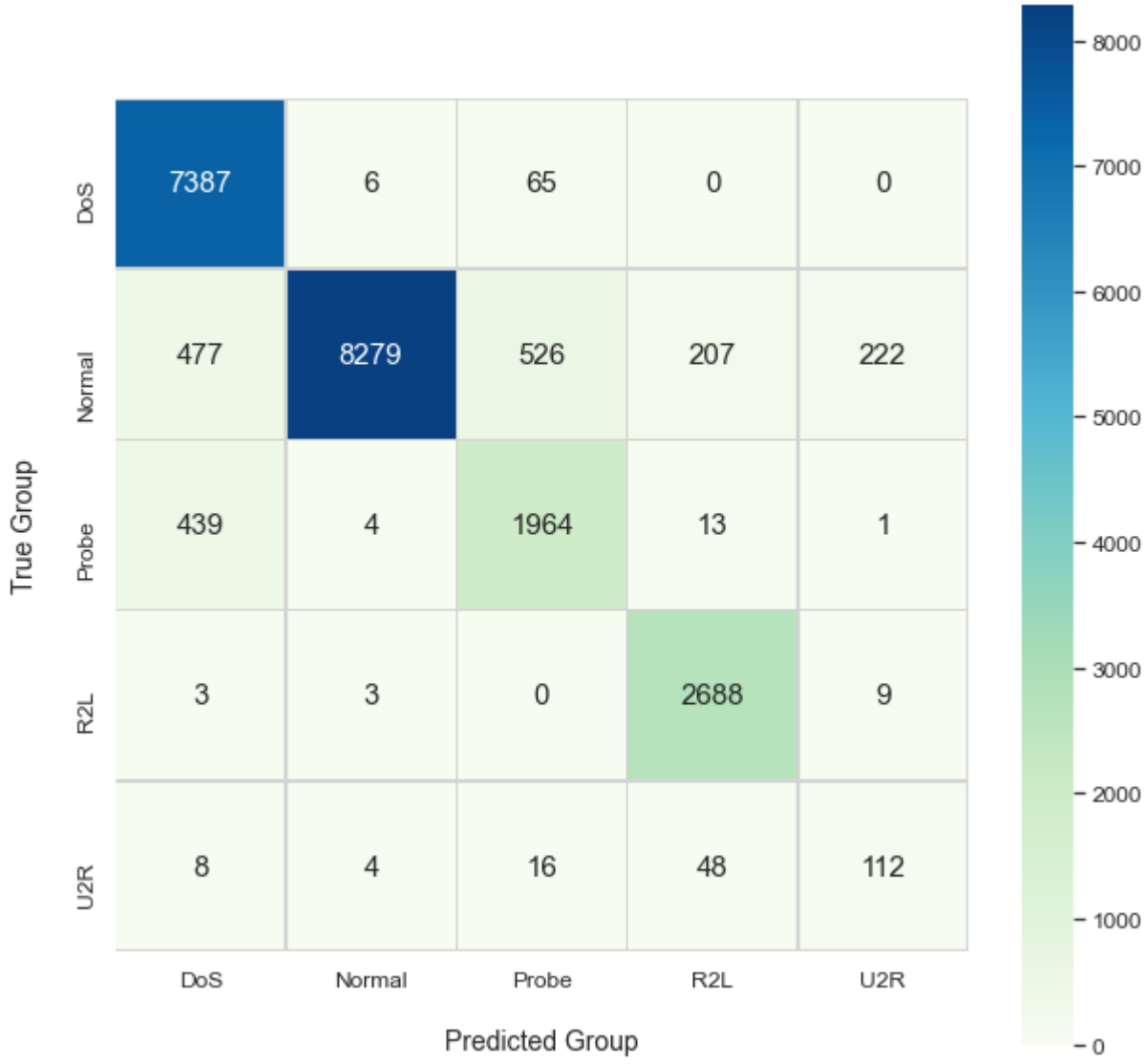
In [28]:
```
from sklearn import metrics
from sklearn.metrics import plot_confusion_matrix
models = []
models.append(('Naive Baye Classifier', BNB_Classifier))
models.append(('Decision Tree Classifier', DTC_Classifier))
models.append(('RandomForest Classifier', RF_Classifier))
models.append(('KNeighborsClassifier', KNN_Classifier))
models.append(('EnsembleClassifier', EnsembleClassifier))
```

Test Models

In [29]:

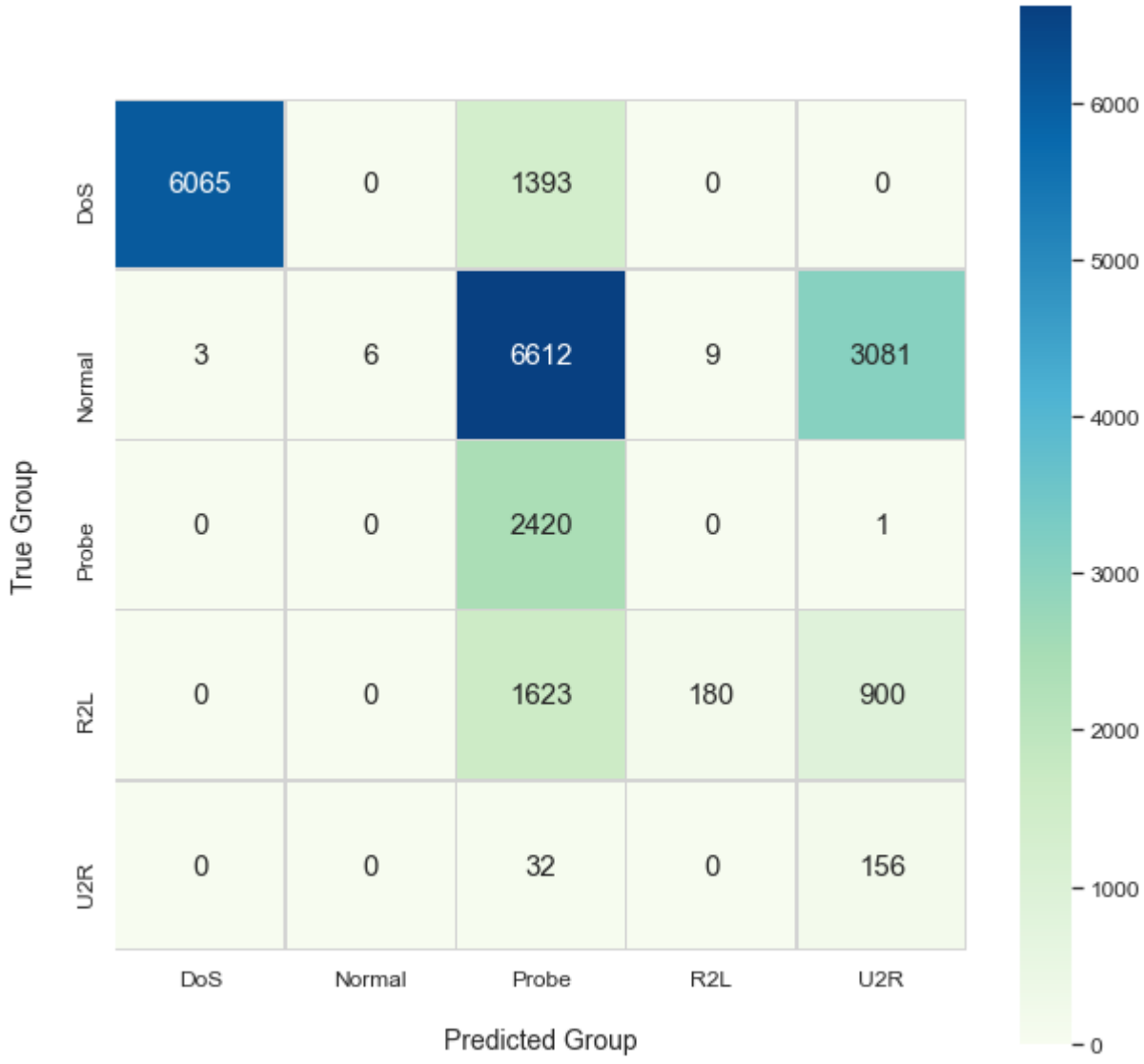
```
def plotHeatmap(cm):
    fig, ax = plt.subplots(figsize=(10,10))
    heatmap = sns.heatmap(cm, annot=True, linewidths=0.5, linecolor="lightgr
    heatmap.set_xticklabels(['DoS','Normal','Probe','R2L','U2R'])
    heatmap.set_yticklabels(['DoS','Normal','Probe','R2L','U2R'])
    plt.xlabel("Predicted Group", labelpad=20)
    plt.ylabel("True Group", labelpad=20)
    plt.show()

for i, v in models:
    accuracy = metrics.accuracy_score(Y_test, v.predict(X_test))
    confusion_matrix = metrics.confusion_matrix(Y_test, v.predict(X_test))
    classification = metrics.classification_report(Y_test, v.predict(X_test)
    print()
    print('============================== {} Model Test Results ============
    print()
    print ("Model Accuracy:" "\n", accuracy)
    print()
    print("Confusion matrix:" "\n", confusion_matrix)
    plotHeatmap(confusion_matrix)
    print()
    print("Classification report:" "\n", classification)
    print()

```

```
============================== Naive Baye Classifier Model Test Results =======
========================

Model Accuracy:
 0.9087674035852498

Confusion matrix:
 [[7387    6   65    0    0]
 [ 477 8279  526  207  222]
 [ 439    4 1964   13    1]
 [   3    3    0 2688    9]
 [   8    4   16   48  112]]
```

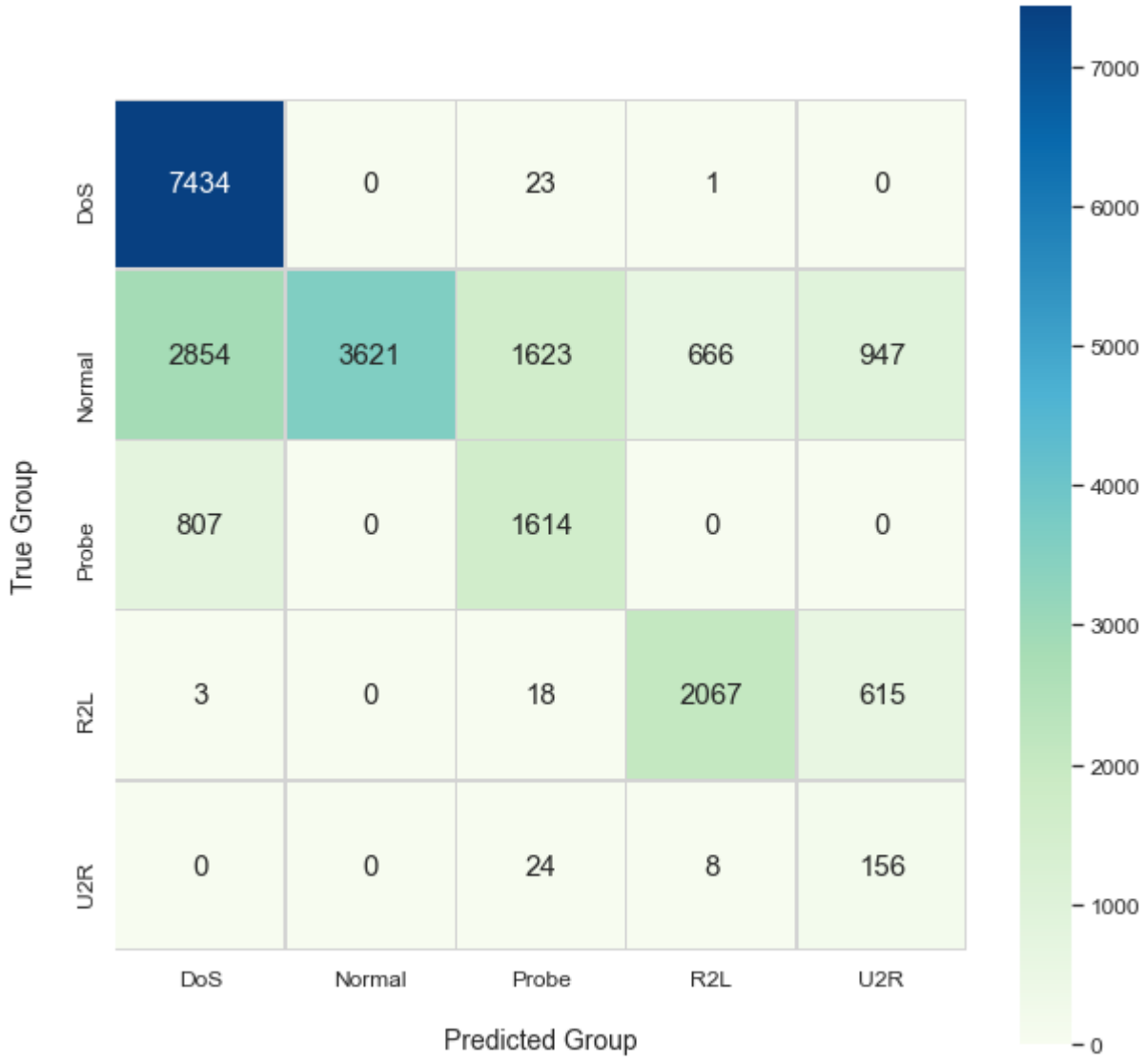



```
Classification report:
              precision    recall  f1-score   support

         0.0       0.89      0.99      0.94      7458
         1.0       1.00      0.85      0.92      9711
         2.0       0.76      0.81      0.79      2421
         3.0       0.91      0.99      0.95      2703
```

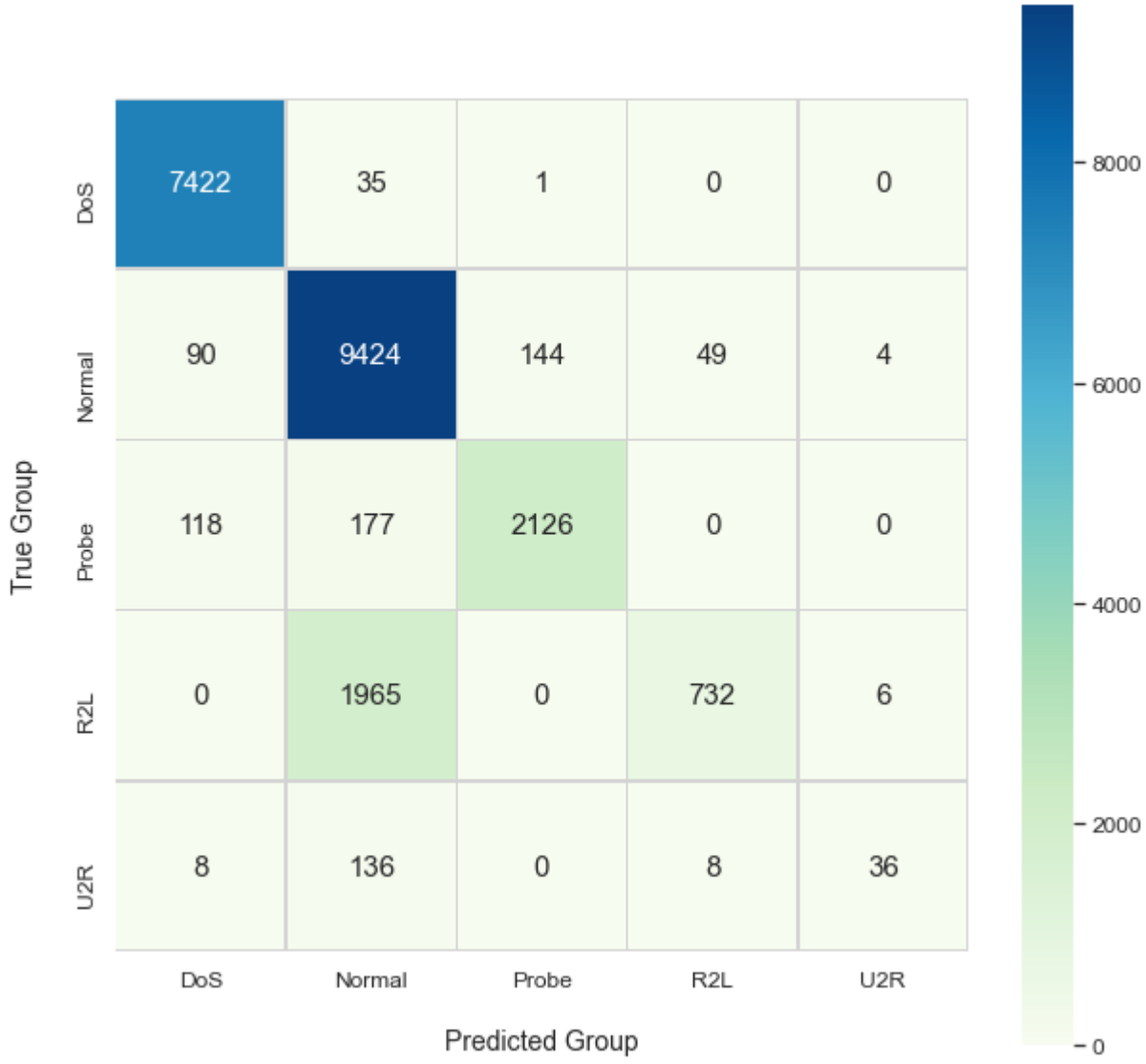
```
         4.0       0.33      0.60      0.42       188

    accuracy                           0.91     22481
   macro avg       0.78      0.85      0.80     22481
weighted avg       0.92      0.91      0.91     22481


============================== Decision Tree Classifier Model Test Results ==
============================

Model Accuracy:
 0.3926426760375428

Confusion matrix:
 [[6065    0 1393    0    0]
 [   3    6 6612    9 3081]
 [   0    0 2420    0    1]
 [   0    0 1623  180  900]
 [   0    0   32    0  156]]
```



```
Classification report:
               precision    recall  f1-score   support

         0.0       1.00      0.81      0.90      7458
         1.0       1.00      0.00      0.00      9711
         2.0       0.20      1.00      0.33      2421
         3.0       0.95      0.07      0.12      2703
         4.0       0.04      0.83      0.07       188

    accuracy                           0.39     22481
   macro avg       0.64      0.54      0.29     22481
weighted avg       0.90      0.39      0.35     22481



============================== RandomForest Classifier Model Test Results =====
=========================

Model Accuracy:
 0.6624260486633157

Confusion matrix:
 [[7434    0   23    1    0]
 [2854 3621 1623  666  947]
 [ 807    0 1614    0    0]
 [   3    0   18 2067  615]
 [   0    0   24    8  156]]
```

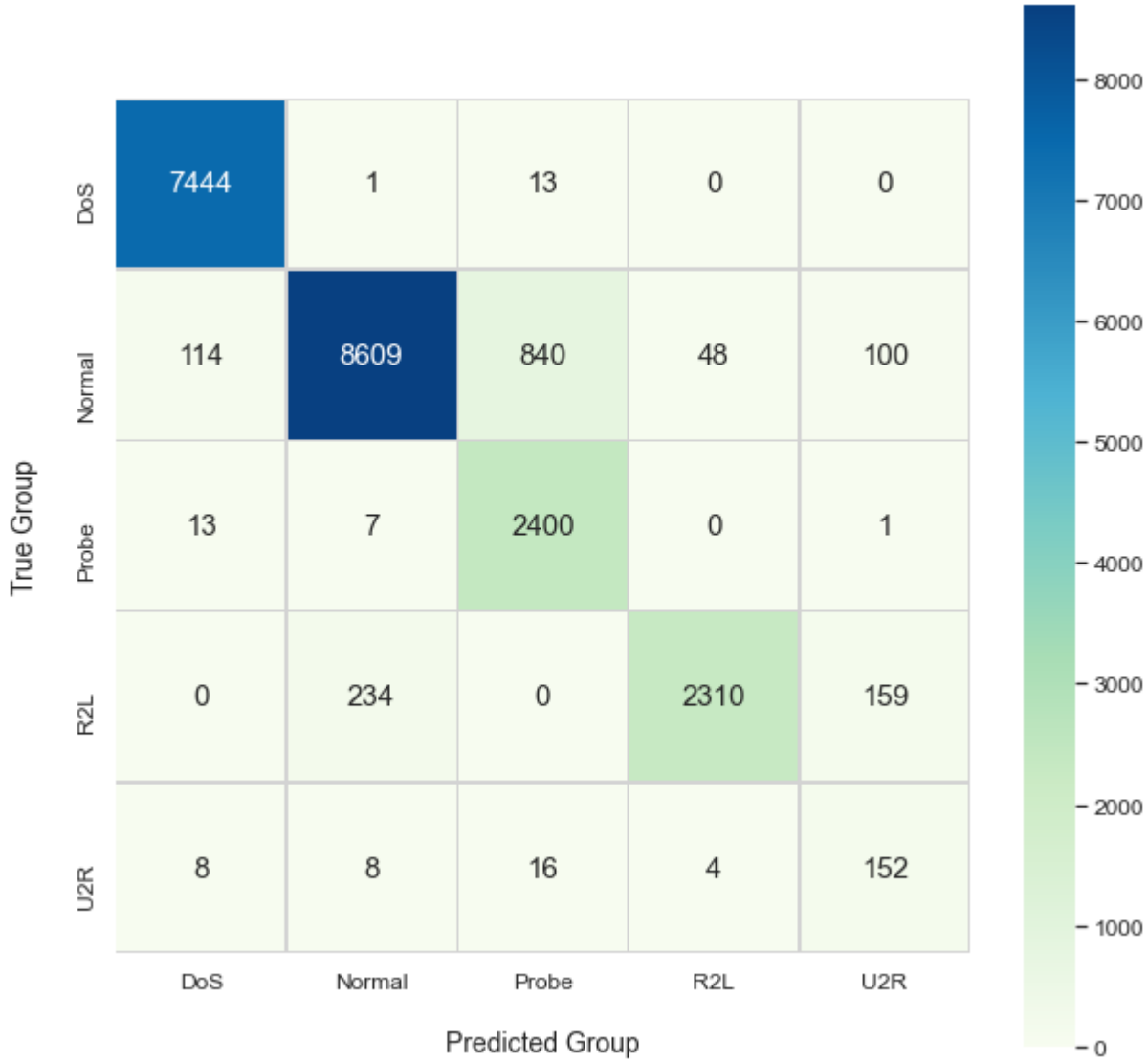

True Group
DoS
Normal
Probe
R2L
U2R
7434
0
23
1
0
2854
3621
1623
666
947
807
0
1614
0
0
3
0
18
2067
615
0
0
24
8
156
DoS
Normal
Probe
R2L
U2R
Predicted Group
7000
6000
5000
4000
3000
2000
1000
0

```
Classification report:
               precision    recall  f1-score   support

          0.0       0.67      1.00      0.80      7458
          1.0       1.00      0.37      0.54      9711
          2.0       0.49      0.67      0.56      2421
          3.0       0.75      0.76      0.76      2703
          4.0       0.09      0.83      0.16       188

    accuracy                           0.66     22481
   macro avg       0.60      0.73      0.57     22481
weighted avg       0.80      0.66      0.65     22481


============================== KNeighborsClassifier Model Test Results ========
======================

Model Accuracy:
 0.8780748187358214

Confusion matrix:
 [[7422   35    1    0    0]
 [  90 9424  144   49    4]
 [ 118  177 2126    0    0]
 [   0 1965    0  732    6]
 [   8  136    0    8   36]]
```




```
Classification report:
              precision    recall  f1-score   support

         0.0       0.97      1.00      0.98      7458
         1.0       0.80      0.97      0.88      9711
         2.0       0.94      0.88      0.91      2421
         3.0       0.93      0.27      0.42      2703
         4.0       0.78      0.19      0.31       188
```

```
    accuracy                           0.88     22481
   macro avg       0.88      0.66      0.70     22481
weighted avg       0.89      0.88      0.86     22481


============================== EnsembleClassifier Model Test Results ========
======================

Model Accuracy:
 0.930341176993906

Confusion matrix:
 [[7444    1   13    0    0]
 [ 114 8609  840   48  100]
 [  13    7 2400    0    1]
 [   0  234    0 2310  159]
 [   8    8   16    4  152]]
```



```
Classification report:
              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99      7458
         1.0       0.97      0.89      0.93      9711
         2.0       0.73      0.99      0.84      2421
         3.0       0.98      0.85      0.91      2703
         4.0       0.37      0.81      0.51       188

    accuracy                           0.93     22481
   macro avg       0.81      0.91      0.84     22481
weighted avg       0.95      0.93      0.93     22481
```