SIT719_Task5_221426969

February 25, 2024

1 Task 5.1 End-to-end project delivery on cyber-security data analytics

2 SECTION 1: DECLARE THE MODULES

[1]:
```
import os
from collections import defaultdict
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import timeit
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import plot_confusion_matrix
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingRandomSearchCV
import warnings
%matplotlib inline
warnings.filterwarnings('ignore')
```

3 SECTION 2: Data import and preprocess

#Run this but dont worry if it does not make any sense Jump to SECTION 3 that is related to your HD task.

```
[2]: !pip install wget
import wget

link_to_data = 'https://raw.githubusercontent.com/SIT719/2020-S2/master/data/
 ↪Week_5_NSL-KDD-Dataset/training_attack_types.txt?raw=true'
DataSet = wget.download(link_to_data)
```

```
Requirement already satisfied: wget in c:\users\vinit\anaconda3\lib\site-
packages (3.2)
```

```
[3]: DataSet
```

```
[3]: 'training_attack_types (53).txt'
```

```
[4]: header_names = ['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_count', 'dst_host_same_srv_rate', 'dst_host_diff_srv_rate',␣
 ↪'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate',␣
 ↪'dst_host_serror_rate', 'dst_host_srv_serror_rate', 'dst_host_rerror_rate',␣
 ↪'dst_host_srv_rerror_rate', 'attack_type', 'success_pred']


# Differentiating between nominal, binary, and numeric features

# root_shell is marked as a continuous feature in the kddcup.names
# file, but it is supposed to be a binary feature according to the
# dataset documentation

# training_attack_types.txt maps each of the 22 different attacks to 1 of 4␣
 ↪categories
# file obtained from http://kdd.ics.uci.edu/databases/kddcup99/
 ↪training_attack_types

col_names = np.array(header_names)

nominal_idx = [1, 2, 3]
binary_idx = [6, 11, 13, 14, 20, 21]
numeric_idx = list(set(range(41)).difference(nominal_idx).
 ↪difference(binary_idx))

nominal_cols = col_names[nominal_idx].tolist()
binary_cols = col_names[binary_idx].tolist()
```

```
numeric_cols = col_names[numeric_idx].tolist()
```

[5]:
```
# training_attack_types.txt maps each of the 22 different attacks to 1 of 4␣
↪categories
# file obtained from http://kdd.ics.uci.edu/databases/kddcup99/
↪training_attack_types

category = defaultdict(list)
category['benign'].append('normal')

with open(DataSet, 'r') as f:
    for line in f.readlines():
        attack, cat = line.strip().split(' ')
        category[cat].append(attack)

attack_mapping = dict((v,k) for k in category for v in category[k])
```

[6]:
```
attack_mapping
```

[6]:
```
{'normal': 'benign',
 'apache2': 'dos',
 'back': 'dos',
 'mailbomb': 'dos',
 'processtable': 'dos',
 'snmpgetattack': 'dos',
 'teardrop': 'dos',
 'smurf': 'dos',
 'land': 'dos',
 'neptune': 'dos',
 'pod': 'dos',
 'udpstorm': 'dos',
 'ps': 'u2r',
 'buffer_overflow': 'u2r',
 'perl': 'u2r',
 'rootkit': 'u2r',
 'loadmodule': 'u2r',
 'xterm': 'u2r',
 'sqlattack': 'u2r',
 'httptunnel': 'u2r',
 'ftp_write': 'r2l',
 'guess_passwd': 'r2l',
 'snmpguess': 'r2l',
 'imap': 'r2l',
 'spy': 'r2l',
 'warezclient': 'r2l',
 'warezmaster': 'r2l',
 'multihop': 'r2l',
```

```
 'phf': 'r2l',
 'named': 'r2l',
 'sendmail': 'r2l',
 'xlock': 'r2l',
 'xsnoop': 'r2l',
 'worm': 'probe',
 'nmap': 'probe',
 'ipsweep': 'probe',
 'portsweep': 'probe',
 'satan': 'probe',
 'mscan': 'probe',
 'saint': 'probe'}
```

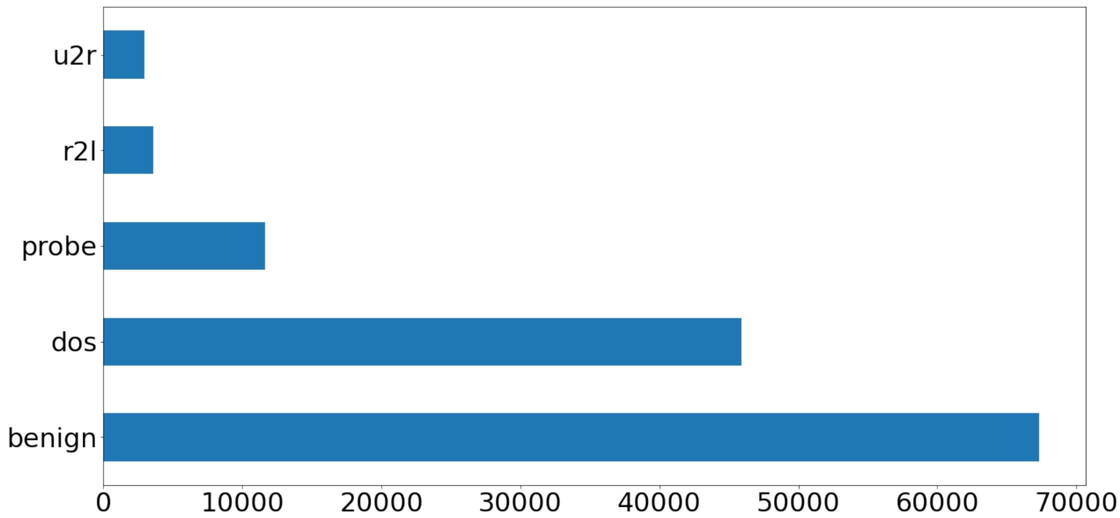
[7]:
```
#Processing Training Data

train_file='https://raw.githubusercontent.com/SIT719/2020-S2/master/data/
 ↪Week_5_NSL-KDD-Dataset/KDDTrain%2B.txt'



train_df = pd.read_csv(train_file, names=header_names)

train_df['attack_category'] = train_df['attack_type'] \
                                .map(lambda x: attack_mapping[x])

train_df.drop(['success_pred'], axis=1, inplace=True)
```

[8]:
```
#Processing test Data
test_file='https://raw.githubusercontent.com/SIT719/2020-S2/master/data/
 ↪Week_5_NSL-KDD-Dataset/KDDTest%2B.txt'

test_df = pd.read_csv(test_file, names=header_names)
test_df['attack_category'] = test_df['attack_type'] \
                                .map(lambda x: attack_mapping[x])
test_df.drop(['success_pred'], axis=1, inplace=True)
```

[9]:
```
train_attack_types = train_df['attack_type'].value_counts()
train_attack_cats = train_df['attack_category'].value_counts()

test_attack_types = test_df['attack_type'].value_counts()
test_attack_cats = test_df['attack_category'].value_counts()

train_attack_types.plot(kind='barh', figsize=(20,10), fontsize=20)

train_attack_cats.plot(kind='barh', figsize=(20,10), fontsize=30)

train_df[binary_cols].describe().transpose()
```

```python
train_df.groupby(['su_attempted']).size()
train_df['su_attempted'].replace(2, 0, inplace=True)
test_df['su_attempted'].replace(2, 0, inplace=True)
train_df.groupby(['su_attempted']).size()
train_df.groupby(['num_outbound_cmds']).size()

#Now, that's not a very useful feature - let's drop it from the dataset

train_df.drop('num_outbound_cmds', axis = 1, inplace=True)
test_df.drop('num_outbound_cmds', axis = 1, inplace=True)
numeric_cols.remove('num_outbound_cmds')



#Data Preparation

train_Y = train_df['attack_category']
train_x_raw = train_df.drop(['attack_category','attack_type'], axis=1)
test_Y = test_df['attack_category']
test_x_raw = test_df.drop(['attack_category','attack_type'], axis=1)


combined_df_raw = pd.concat([train_x_raw, test_x_raw])
combined_df = pd.get_dummies(combined_df_raw, columns=nominal_cols, 
 ↪drop_first=True)

train_x = combined_df[:len(train_x_raw)]
test_x = combined_df[len(train_x_raw):]

# Store dummy variable feature names
dummy_variables = list(set(train_x)-set(combined_df_raw))

#execute the commands in console
train_x.describe()
train_x['duration'].describe()
# Experimenting with StandardScaler on the single 'duration' feature
from sklearn.preprocessing import StandardScaler

durations = train_x['duration'].values.reshape(-1, 1)
standard_scaler = StandardScaler().fit(durations)
scaled_durations = standard_scaler.transform(durations)
pd.Series(scaled_durations.flatten()).describe()

# Experimenting with MinMaxScaler on the single 'duration' feature
from sklearn.preprocessing import MinMaxScaler

min_max_scaler = MinMaxScaler().fit(durations)
```

```
min_max_scaled_durations = min_max_scaler.transform(durations)
pd.Series(min_max_scaled_durations.flatten()).describe()

# Experimenting with RobustScaler on the single 'duration' feature
from sklearn.preprocessing import RobustScaler

min_max_scaler = RobustScaler().fit(durations)
robust_scaled_durations = min_max_scaler.transform(durations)
pd.Series(robust_scaled_durations.flatten()).describe()

# Experimenting with MaxAbsScaler on the single 'duration' feature
from sklearn.preprocessing import MaxAbsScaler

max_Abs_scaler = MaxAbsScaler().fit(durations)
robust_scaled_durations = max_Abs_scaler.transform(durations)
pd.Series(robust_scaled_durations.flatten()).describe()

# Let's proceed with StandardScaler- Apply to all the numeric columns

standard_scaler = StandardScaler().fit(train_x[numeric_cols])

train_x[numeric_cols] = \
    standard_scaler.transform(train_x[numeric_cols])

test_x[numeric_cols] = \
    standard_scaler.transform(test_x[numeric_cols])

train_x.describe()



train_Y_bin = train_Y.apply(lambda x: 0 if x is 'benign' else 1)
test_Y_bin = test_Y.apply(lambda x: 0 if x is 'benign' else 1)
```



4 SECTION 3: Multi class classification

#This is the section where you have to add other algorithms, tune algorithms and visualize to compare and analyze algorithms

Building Model

4.1 1. Random Forest Classifier.

[41]:
```
#Calculate start time
start = timeit.default_timer()

param_grid = [{'n_estimators': [10, 25], 'max_features': [5, 10], 'max_depth':
↪[10, 50, None], 'bootstrap': [True, False]}]

rf = RandomForestClassifier()

search_random = HalvingRandomSearchCV(estimator = rf, param_distributions =
↪param_grid, verbose = 1)

search_random.fit(train_x, train_Y)

#Calculate Stop time
stop = timeit.default_timer()
train_time= stop - start
```

```
n_iterations: 3
n_required_iterations: 3
n_possible_iterations: 8
min_resources_: 50
```

```
max_resources_: 125973
aggressive_elimination: False
factor: 3
----------
iter: 0
n_candidates: 24
n_resources: 50
Fitting 5 folds for each of 24 candidates, totalling 120 fits
----------
iter: 1
n_candidates: 8
n_resources: 150
Fitting 5 folds for each of 8 candidates, totalling 40 fits
----------
iter: 2
n_candidates: 3
n_resources: 450
Fitting 5 folds for each of 3 candidates, totalling 15 fits
```

[42]:
```
search_random.best_params_
```

[42]:
```
{'n_estimators': 25, 'max_features': 10, 'max_depth': None, 'bootstrap': False}
```

[43]:
```
#Calculate start time
start = timeit.default_timer()

pred_y_random = search_random.predict(test_x)

#Calculate Stop time
stop = timeit.default_timer()
test_time= stop - start
```

[44]:
```
results_random = confusion_matrix(test_Y, pred_y_random)

print(results_random)
```

```
[[9464   56  190    0    1]
 [1570 5960  106    0    0]
 [ 720  165 1538    0    0]
 [2511    0    2   59    2]
 [ 195    0    0    3    2]]
```

[45]:
```
#Train time
print('Train Time(s): ',train_time)

#Test time
print('Test Time(s): ',test_time)
```

```
Train Time(s):  85.00679750001291
```

```
Test Time(s):  0.5082461000129115
```

4.2 2. Naive Bayes Classifier

[46]:
```python
#Calculate start time
start = timeit.default_timer()

from sklearn.naive_bayes import GaussianNB
gnb_model = GaussianNB()
params_NB = {'var_smoothing': np.logspace(0,-9, num=100)}

gs_NB = HalvingRandomSearchCV(estimator=gnb_model, param_distributions =␣
 ↪params_NB,
                     verbose=1,
                     scoring='accuracy')

gs_NB.fit(train_x, train_Y)

#Calculate Stop time
stop = timeit.default_timer()
train_time= stop - start
```

```
n_iterations: 5
n_required_iterations: 5
n_possible_iterations: 8
min_resources_: 50
max_resources_: 125973
aggressive_elimination: False
factor: 3
----------
iter: 0
n_candidates: 100
n_resources: 50
Fitting 5 folds for each of 100 candidates, totalling 500 fits
----------
iter: 1
n_candidates: 34
n_resources: 150
Fitting 5 folds for each of 34 candidates, totalling 170 fits
----------
iter: 2
n_candidates: 12
n_resources: 450
Fitting 5 folds for each of 12 candidates, totalling 60 fits
----------
iter: 3
n_candidates: 4
n_resources: 1350
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
----------
iter: 4
n_candidates: 2
n_resources: 4050
Fitting 5 folds for each of 2 candidates, totalling 10 fits
```

[47]:
```
gs_NB.best_estimator_
```

[47]:
```
GaussianNB(var_smoothing=0.0001)
```

[48]:
```
#Calculate start time
start = timeit.default_timer()

predict_gnb = gs_NB.predict(test_x)

#Calculate Stop time
stop = timeit.default_timer()
test_time= stop - start
```

[49]:
```
results_gnb = confusion_matrix(test_Y, predict_gnb)

print(results_gnb)
```

```
[[9002  103  207  109  290]
 [1207 3686 1934  803    6]
 [ 368  384 1338  257   76]
 [1128    6    6  845  589]
 [  20    0  111   14   55]]
```

[50]:
```
#Train time
print('Train Time(s): ',train_time)

#Test time
print('Test Time(s): ',test_time)
```

```
Train Time(s):  63.191990699997405
Test Time(s):  0.7797886000189465
```

4.3 3. K-Nearest Neighbor

[10]:
```
#Calculate start time
start = timeit.default_timer()

parameters = {"leaf_size" : [5,10,15,20,25,30,35,40]}

model_k_neighbors = KNeighborsClassifier()
model_knn = HalvingRandomSearchCV(model_k_neighbors, param_distributions =
 ↪parameters,scoring='accuracy',verbose = 1)
```

```
model_knn.fit(train_x, train_Y)

#Calculate Stop time
stop = timeit.default_timer()
train_time= stop - start
```

```
n_iterations: 2
n_required_iterations: 2
n_possible_iterations: 8
min_resources_: 50
max_resources_: 125973
aggressive_elimination: False
factor: 3
----------
iter: 0
n_candidates: 8
n_resources: 50
Fitting 5 folds for each of 8 candidates, totalling 40 fits
----------
iter: 1
n_candidates: 3
n_resources: 150
Fitting 5 folds for each of 3 candidates, totalling 15 fits
```

[11]:
```
print(f'Best parameters - {model_knn.best_params_}')
```

```
Best parameters - {'leaf_size': 30}
```

[12]:
```
#Calculate start time
start = timeit.default_timer()

pred_knn = model_knn.predict(test_x)

#Calculate Stop time
stop = timeit.default_timer()
test_time= stop - start
```

[13]:
```
results_knn = confusion_matrix(test_Y, pred_knn)
print(results_knn)
```

```
[[9444   54  207    5    1]
 [1630 5925   81    0    0]
 [ 614  180 1629    0    0]
 [2362    2   40  170    0]
 [ 170    0   17    4    9]]
```

[14]:
```
#Train time
print('Train Time(s): ',train_time)
```

```
#Test time
print('Test Time(s): ',test_time)
```

```
Train Time(s):  5.791088500000001
Test Time(s):  166.35820370000002
```

4.4 4. Support Vector Machine Classifier

[83]:
```
#Calculate start time
start = timeit.default_timer()

svc_model= SVC()

params = {'C': [10,20,30,40,50],
          'kernel': ['rbf'],
          'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
         }
model_s = HalvingRandomSearchCV(svc_model, param_distributions = params,␣
 ↪verbose = 1)
model_s.fit(train_x, train_Y)

#Calculate Stop time
stop = timeit.default_timer()
train_time= stop - start
```

```
n_iterations: 3
n_required_iterations: 3
n_possible_iterations: 8
min_resources_: 50
max_resources_: 125973
aggressive_elimination: False
factor: 3
----------
iter: 0
n_candidates: 25
n_resources: 50
Fitting 5 folds for each of 25 candidates, totalling 125 fits
----------
iter: 1
n_candidates: 9
n_resources: 150
Fitting 5 folds for each of 9 candidates, totalling 45 fits
----------
iter: 2
n_candidates: 3
n_resources: 450
Fitting 5 folds for each of 3 candidates, totalling 15 fits
```

[84]:
```
print("Best Hyper Parameters:\n",model_s.best_params_)
```

```
Best Hyper Parameters:
 {'kernel': 'rbf', 'gamma': 0.01, 'C': 40}
```

[85]:
```
#Calculate start time
start = timeit.default_timer()

pred_svc =model_s.predict(test_x)


#Calculate Stop time
stop = timeit.default_timer()
test_time= stop - start
```

[86]:
```
results_svc = confusion_matrix(test_Y, pred_svc)

print(results_svc)
```

```
[[9359   59  290    2    1]
 [1458 6141   37    0    0]
 [ 725  171 1527    0    0]
 [2302    0    7  264    1]
 [ 181    0    0    5   14]]
```

[87]:
```
#Train time
print('Train Time(s): ',train_time)

#Test time
print('Test Time(s): ',test_time)
```

```
Train Time(s):  212.91485150001245
Test Time(s):  74.35728259998723
```

4.5 5. AdaBoost Classifier

[88]:
```
#Calculate start time
start = timeit.default_timer()

params =  { 'n_estimators': [20,25,30,35,40,45,50],
    'learning_rate': [(0.97 + x / 100) for x in range(0, 25)],
    'algorithm': ['SAMME', 'SAMME.R']}

ada = AdaBoostClassifier()

ada_model= HalvingRandomSearchCV(ada, param_distributions =params, verbose = 1)
ada_model.fit(train_x, train_Y)

#Calculate Stop time
```

```
stop = timeit.default_timer()
train_time= stop - start
```

```
n_iterations: 6
n_required_iterations: 6
n_possible_iterations: 8
min_resources_: 50
max_resources_: 125973
aggressive_elimination: False
factor: 3
----------
iter: 0
n_candidates: 350
n_resources: 50
Fitting 5 folds for each of 350 candidates, totalling 1750 fits
----------
iter: 1
n_candidates: 117
n_resources: 150
Fitting 5 folds for each of 117 candidates, totalling 585 fits
----------
iter: 2
n_candidates: 39
n_resources: 450
Fitting 5 folds for each of 39 candidates, totalling 195 fits
----------
iter: 3
n_candidates: 13
n_resources: 1350
Fitting 5 folds for each of 13 candidates, totalling 65 fits
----------
iter: 4
n_candidates: 5
n_resources: 4050
Fitting 5 folds for each of 5 candidates, totalling 25 fits
----------
iter: 5
n_candidates: 2
n_resources: 12150
Fitting 5 folds for each of 2 candidates, totalling 10 fits
```

[89]:
```
print("Best Hyper Parameters:\n",ada_model.best_params_)
```

```
Best Hyper Parameters:
 {'n_estimators': 50, 'learning_rate': 0.99, 'algorithm': 'SAMME'}
```

[90]:
```
#Calculate start time
start = timeit.default_timer()
```

```
pred_ada = ada_model.predict(test_x)


#Calculate Stop time
stop = timeit.default_timer()
test_time= stop - start
```

[91]:
```
results_ada = confusion_matrix(test_Y, pred_ada)
print(results_ada)
```

```
[[9438   73  200    0    0]
 [1518 5558  560    0    0]
 [ 646  395 1382    0    0]
 [2529    0   44    1    0]
 [ 172    3   25    0    0]]
```

[92]:
```
#Train time
print('Train Time(s): ',train_time)

#Test time
print('Test Time(s): ',test_time)
```

```
Train Time(s):  1490.375949499983
Test Time(s):  3.53567819998716
```

4.6 6. Logistic Regression

[93]:
```
#Calculate start time
start = timeit.default_timer()

LR = LogisticRegression()

LRparam_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    'max_iter': list(range(100,800,100)),
}

LR_search = HalvingRandomSearchCV(LR, param_distributions=LRparam_grid, verbose␣
 ↪= 1)

# fitting the model for grid search
LR_search.fit(train_x, train_Y)

#Calculate Stop time
stop = timeit.default_timer()
train_time= stop - start
```

```
n_iterations: 4
n_required_iterations: 4
n_possible_iterations: 8
min_resources_: 50
max_resources_: 125973
aggressive_elimination: False
factor: 3
----------
iter: 0
n_candidates: 49
n_resources: 50
Fitting 5 folds for each of 49 candidates, totalling 245 fits
----------
iter: 1
n_candidates: 17
n_resources: 150
Fitting 5 folds for each of 17 candidates, totalling 85 fits
----------
iter: 2
n_candidates: 6
n_resources: 450
Fitting 5 folds for each of 6 candidates, totalling 30 fits
----------
iter: 3
n_candidates: 2
n_resources: 1350
Fitting 5 folds for each of 2 candidates, totalling 10 fits
```

[94]:
```
LR_search.best_params_
```

[94]:
```
{'max_iter': 200, 'C': 10}
```

[95]:
```
#Calculate start time
start = timeit.default_timer()

pred_lr = LR_search.predict(test_x)
#Calculate Stop time
stop = timeit.default_timer()
test_time= stop - start
```

[96]:
```
results_lr = confusion_matrix(test_Y, pred_lr)

print(results_lr)
```

```
[[9001   87  616    3    4]
 [1760 5868    8    0    0]
 [ 523  128 1671  101    0]
 [2511    2    3   56    2]
```

```
[ 179    5    0    5   11]]
```

[97]:
```
#Train time
print('Train Time(s): ',train_time)

#Test time
print('Test Time(s): ',test_time)
```

```
Train Time(s):  217.5603517999989
Test Time(s):  0.09459520000382327
```

Evaluating Model

[15]:
```
# Function to obtain metrics for all classifiers using the confusion matrix

def metrics(matrix):
    FP = matrix.sum(axis=0) - np.diag(matrix)
    FN = matrix.sum(axis=1) - np.diag(matrix)
    TP = np.diag(matrix)
    TN = matrix.sum() - (FP + FN + TP)

    FP = FP.astype(float)
    FN = FN.astype(float)
    TP = TP.astype(float)
    TN = TN.astype(float)

    # Accuracy
    accuracy = (TP+TN)/(TP+TN+FP+FN)
    accuracy = (accuracy*100).round(2)

    # Precision
    precision = (TP/(TP+FP))
    precision = (precision*100).round(2)

    #  recall
    recall = TP/(TP+FN)
    recall = (recall*100).round(2)

    # F score
    f1_score = 2*(precision*recall)/(precision+recall)
    f1_score = (f1_score).round(2)

    # false positive rate
    FPR = FP/(FP+TN)
    FPR = (FPR*100).round(2)

    return accuracy, precision , recall , f1_score, FPR
```

4.7 1. Random Forest

[240]:
```
random_metrics = metrics(results_random)
```

[241]:
```
print("Attack Classes Identified -'normal','dos','probe','r2L','u2r'")
print('\n')
print('Accuracy for each attack class',random_metrics[0])
print('\n')
print('Precision for each attack class',random_metrics[1])
print('\n')
print('Recall for each attack class',random_metrics[2])
print('\n')
print('F Score for each attack class',random_metrics[3])
print('\n')
print('FPR for each attack class',random_metrics[4])
```

```
Attack Classes Identified -'normal','dos','probe','r2L','u2r'


Accuracy for each attack class [76.74 91.59 94.75 88.83 99.11]


Precision for each attack class [65.45 96.42 83.77 95.16 40.  ]


Recall for each attack class [97.46 78.05 63.48  2.29  1.  ]


F Score for each attack class [78.31 86.27 72.23  4.47  1.95]


FPR for each attack class [3.893e+01 1.480e+00 1.480e+00 2.000e-02 1.000e-02]
```

[229]:
```
print(classification_report(test_Y, pred_y_random))
```

```
              precision    recall  f1-score   support

      benign       0.65      0.97      0.78      9711
         dos       0.96      0.78      0.86      7636
       probe       0.84      0.63      0.72      2423
         r2l       0.95      0.02      0.04      2574
         u2r       0.40      0.01      0.02       200

    accuracy                           0.76     22544
   macro avg       0.76      0.48      0.49     22544
weighted avg       0.81      0.76      0.71     22544
```

4.8 2. Naive Bayes

[242]:
```
nb_metrics = metrics(results_gnb)
```

[243]:
```
print("Attack Classes Identified -'normal','dos','probe','r2L','u2r'")
print('\n')
print('Accuracy for each attack class',nb_metrics[0])
print('\n')
print('Precision for each attack class',nb_metrics[1])
print('\n')
print('Recall for each attack class',nb_metrics[2])
print('\n')
print('F Score for each attack class',nb_metrics[3])
print('\n')
print('FPR for each attack class',nb_metrics[4])
```

```
Attack Classes Identified -'normal','dos','probe','r2L','u2r'


Accuracy for each attack class [84.78 80.29 85.17 87.08 95.09]


Precision for each attack class [76.78 88.2  37.21 41.67  5.41]


Recall for each attack class [92.7  48.27 55.22 32.83 27.5 ]


F Score for each attack class [83.99 62.39 44.46 36.73  9.04]


FPR for each attack class [21.22  3.31 11.22  5.92  4.3 ]
```

[190]:
```
print(classification_report(test_Y, predict_gnb))
```

```
              precision    recall  f1-score   support

      benign       0.77      0.93      0.84      9711
         dos       0.88      0.48      0.62      7636
       probe       0.37      0.55      0.44      2423
         r2l       0.42      0.33      0.37      2574
         u2r       0.05      0.28      0.09       200

    accuracy                           0.66     22544
   macro avg       0.50      0.51      0.47     22544
weighted avg       0.72      0.66      0.66     22544
```

4.9 3. KNN

[16]:
```
knn_metrics = metrics(results_knn)
```

[17]:
```
print("Attack Classes Identified -'normal','dos','probe','r2L','u2r'")
print('\n')
print('Accuracy for each attack class',knn_metrics[0])
print('\n')
print('Precision for each attack class',knn_metrics[1])
print('\n')
print('Recall for each attack class',knn_metrics[2])
print('\n')
print('F Score for each attack class',knn_metrics[3])
print('\n')
print('FPR for each attack class',knn_metrics[4])
```

```
Attack Classes Identified -'normal','dos','probe','r2L','u2r'


Accuracy for each attack class [77.63 91.36 94.95 89.3  99.15]


Precision for each attack class [66.41 96.17 82.52 94.97 90.  ]


Recall for each attack class [97.25 77.59 67.23  6.6   4.5 ]


F Score for each attack class [78.92 85.89 74.09 12.34  8.57]


FPR for each attack class [37.22  1.58  1.71  0.05  0.  ]
```

[18]:
```
print(classification_report(test_Y, pred_knn))
```

```
              precision    recall  f1-score   support

      benign       0.66      0.97      0.79      9711
         dos       0.96      0.78      0.86      7636
       probe       0.83      0.67      0.74      2423
         r2l       0.95      0.07      0.12      2574
         u2r       0.90      0.04      0.09       200

    accuracy                           0.76     22544
   macro avg       0.86      0.51      0.52     22544
weighted avg       0.82      0.76      0.73     22544
```

4.10 4. SVM

[244]:
```
svm_metrics = metrics(results_svc)
```

[245]:
```
print("Attack Classes Identified -'normal','dos','probe','r2L','u2r'")
print('\n')
print('Accuracy for each attack class',svm_metrics[0])
print('\n')
print('Precision for each attack class',svm_metrics[1])
print('\n')
print('Recall for each attack class',svm_metrics[2])
print('\n')
print('F Score for each attack class',svm_metrics[3])
print('\n')
print('FPR for each attack class',svm_metrics[4])
```

```
Attack Classes Identified -'normal','dos','probe','r2L','u2r'


Accuracy for each attack class [77.74 92.35 94.54 89.72 99.17]


Precision for each attack class [66.73 96.39 82.05 97.42 87.5 ]


Recall for each attack class [96.38 80.42 63.02 10.26  7.  ]


F Score for each attack class [78.86 87.68 71.29 18.56 12.96]


FPR for each attack class [3.636e+01 1.540e+00 1.660e+00 4.000e-02 1.000e-02]
```

[202]:
```
print(classification_report(test_Y, pred_svc))
```

```
              precision    recall  f1-score   support

      benign       0.67      0.96      0.79      9711
         dos       0.96      0.80      0.88      7636
       probe       0.82      0.63      0.71      2423
         r2l       0.97      0.10      0.19      2574
         u2r       0.88      0.07      0.13       200

    accuracy                           0.77     22544
   macro avg       0.86      0.51      0.54     22544
weighted avg       0.82      0.77      0.74     22544
```

4.11 5. AdaBoost

[246]:
```
ada_metrics = metrics(results_ada)
```

[247]:
```
print("Attack Classes Identified -'normal','dos','probe','r2L','u2r'")
print('\n')
print('Accuracy for each attack class',ada_metrics[0])
print('\n')
print('Precision for each attack class',ada_metrics[1])
print('\n')
print('Recall for each attack class',ada_metrics[2])
print('\n')
print('F Score for each attack class',ada_metrics[3])
print('\n')
print('FPR for each attack class',ada_metrics[4])
```

```
Attack Classes Identified -'normal','dos','probe','r2L','u2r'


Accuracy for each attack class [77.21 88.69 91.71 88.59 99.11]


Precision for each attack class [ 65.99  92.19  62.51 100.      nan]


Recall for each attack class [9.719e+01 7.279e+01 5.704e+01 4.000e-02 0.000e+00]


F Score for each attack class [7.861e+01 8.135e+01 5.965e+01 8.000e-02
nan]


FPR for each attack class [37.91  3.16  4.12  0.    0.  ]
```

[166]:
```
print(classification_report(test_Y, pred_ada))
```

```
              precision    recall  f1-score   support

      benign       0.66      0.97      0.79      9711
         dos       0.92      0.73      0.81      7636
       probe       0.63      0.57      0.60      2423
         r2l       1.00      0.00      0.00      2574
         u2r       0.00      0.00      0.00       200

    accuracy                           0.73     22544
   macro avg       0.64      0.45      0.44     22544
weighted avg       0.78      0.73      0.68     22544
```

4.12 6. Logistic Regression

[248]:
```
lr_metrics = metrics(results_lr)
```

[249]:
```
print("Attack Classes Identified -'normal','dos','probe','r2L','u2r'")
print('\n')
print('Accuracy for each attack class',lr_metrics[0])
print('\n')
print('Precision for each attack class',lr_metrics[1])
print('\n')
print('Recall for each attack class',lr_metrics[2])
print('\n')
print('F Score for each attack class',lr_metrics[3])
print('\n')
print('FPR for each attack class',lr_metrics[4])
```

```
Attack Classes Identified -'normal','dos','probe','r2L','u2r'


Accuracy for each attack class [74.79 91.17 93.88 88.35 99.14]


Precision for each attack class [64.41 96.35 72.72 33.94 64.71]


Recall for each attack class [92.69 76.85 68.96  2.18  5.5 ]


F Score for each attack class [76.   85.5  70.79  4.1  10.14]


FPR for each attack class [3.875e+01 1.490e+00 3.120e+00 5.500e-01 3.000e-02]
```

[169]:
```
print(classification_report(test_Y, pred_lr))
```

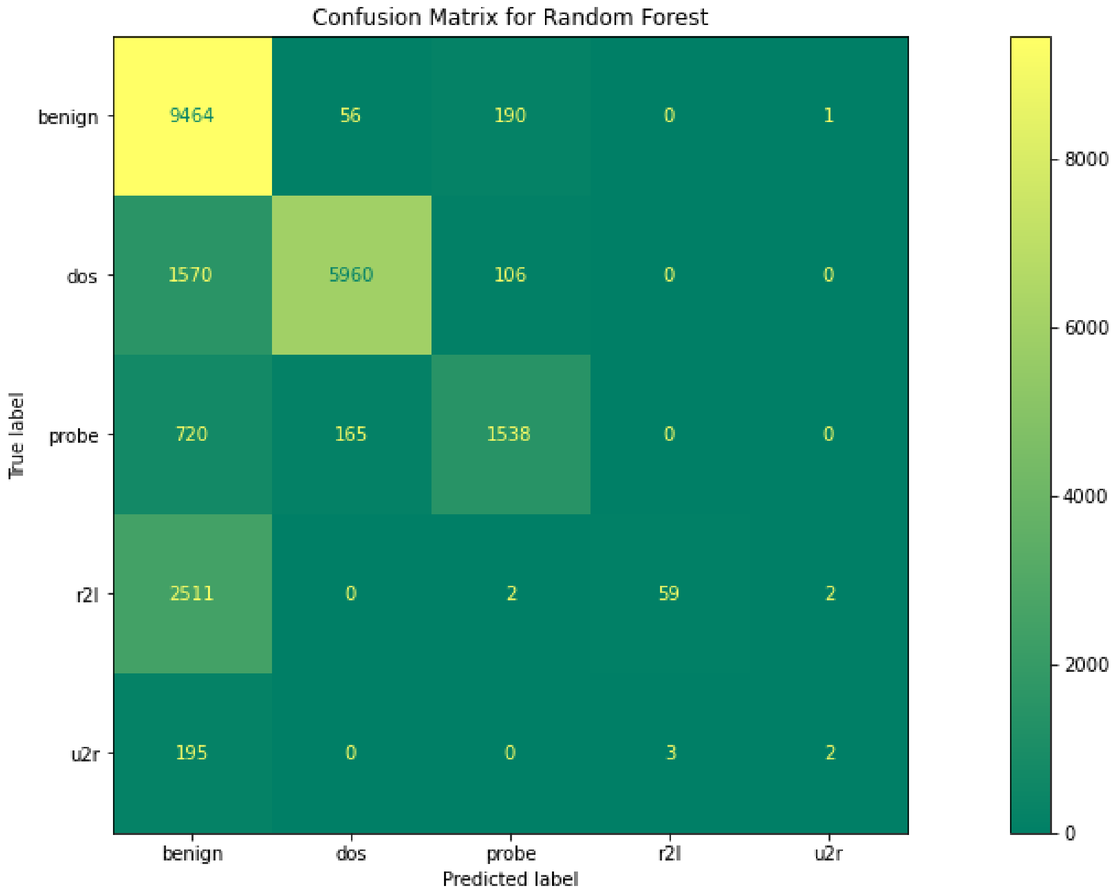
```
              precision    recall  f1-score   support

      benign       0.64      0.93      0.76      9711
         dos       0.96      0.77      0.86      7636
       probe       0.73      0.69      0.71      2423
         r2l       0.34      0.02      0.04      2574
         u2r       0.65      0.06      0.10       200

    accuracy                           0.74     22544
   macro avg       0.66      0.49      0.49     22544
weighted avg       0.73      0.74      0.70     22544
```

Visualizing the Model

4.13 1.Random Forest

[253]:
```
plt.rcParams["figure.figsize"] = (20,8)
plot_confusion_matrix(search_random, test_x, test_Y, cmap = 'summer')
plt.title("Confusion Matrix for Random Forest")
```

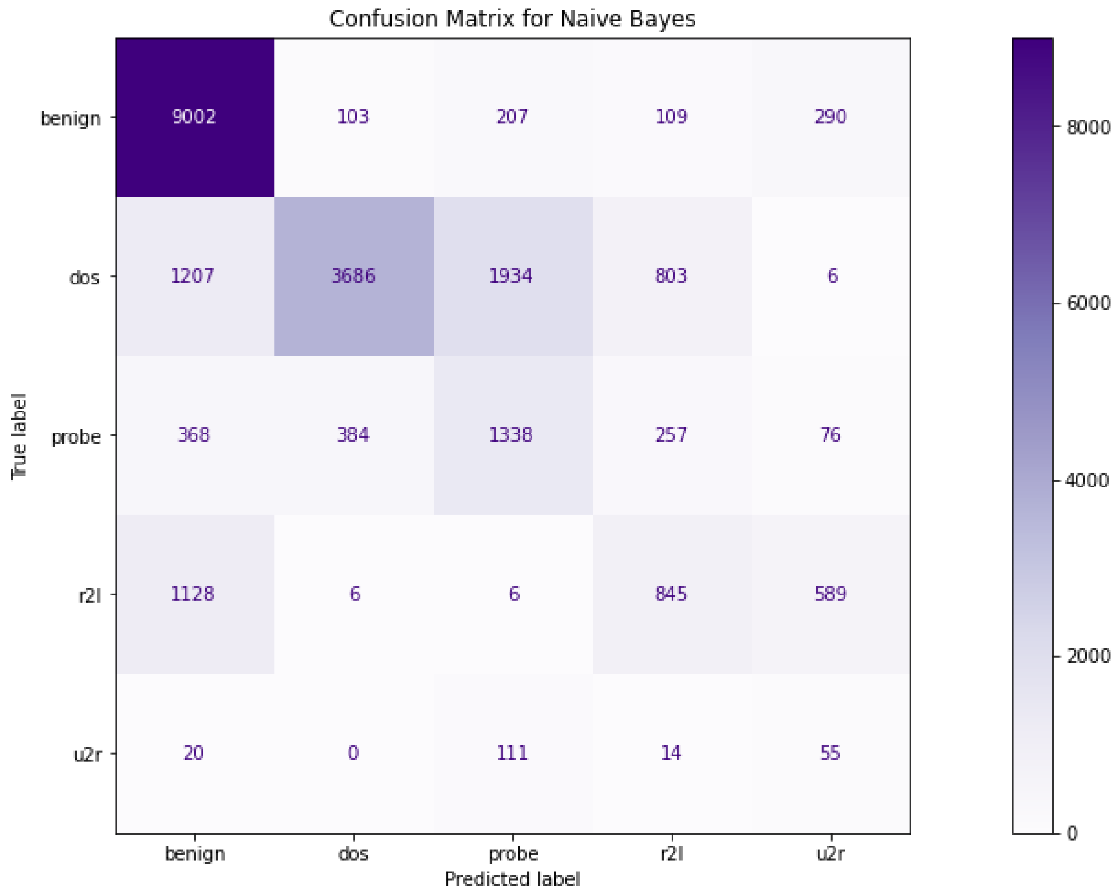
[253]: Text(0.5, 1.0, 'Confusion Matrix for Random Forest')



4.14 2. Naive Bayes

[254]:
```
plt.rcParams["figure.figsize"] = (20,8)
plot_confusion_matrix(gs_NB, test_x, test_Y, cmap = 'Purples')
plt.title("Confusion Matrix for Naive Bayes")
```

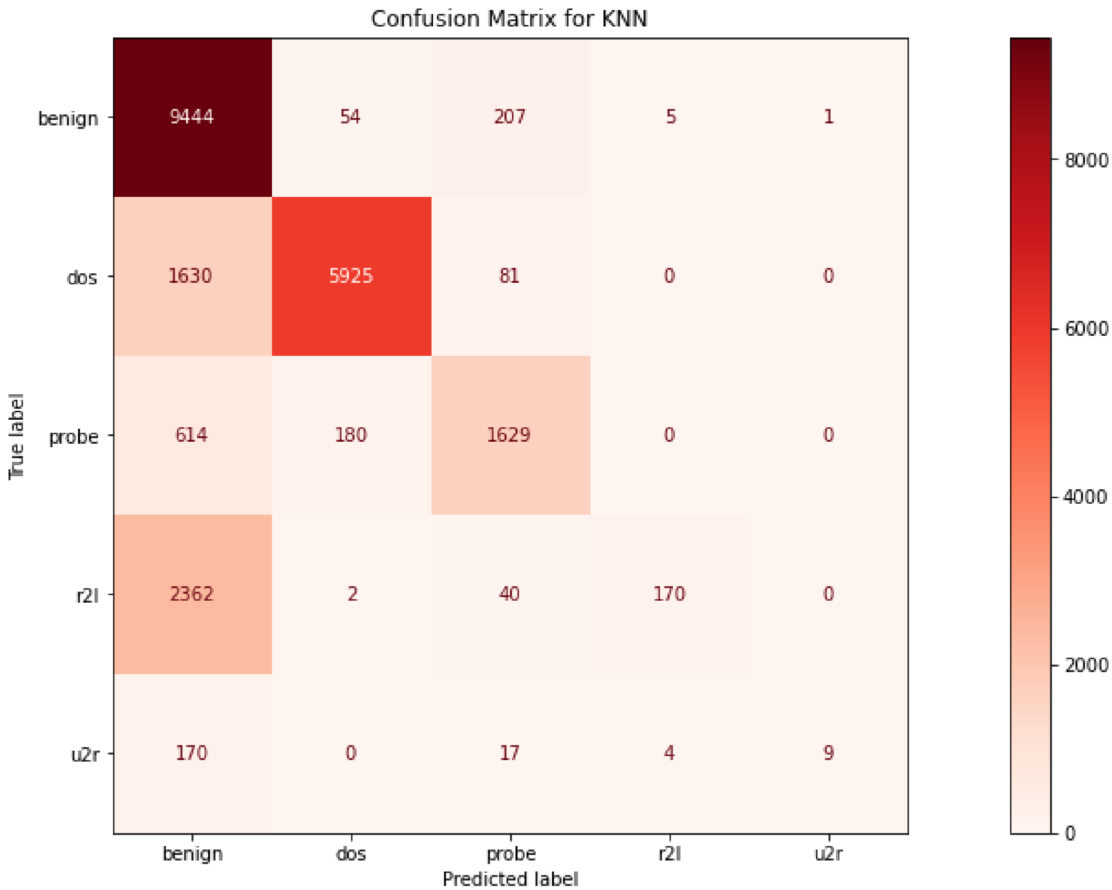
[254]: Text(0.5, 1.0, 'Confusion Matrix for Naive Bayes')



4.15 3. KNN

[19]:
```
plt.rcParams["figure.figsize"] = (20,8)
plot_confusion_matrix(model_knn, test_x, test_Y, cmap = 'Reds')
plt.title("Confusion Matrix for KNN")
```

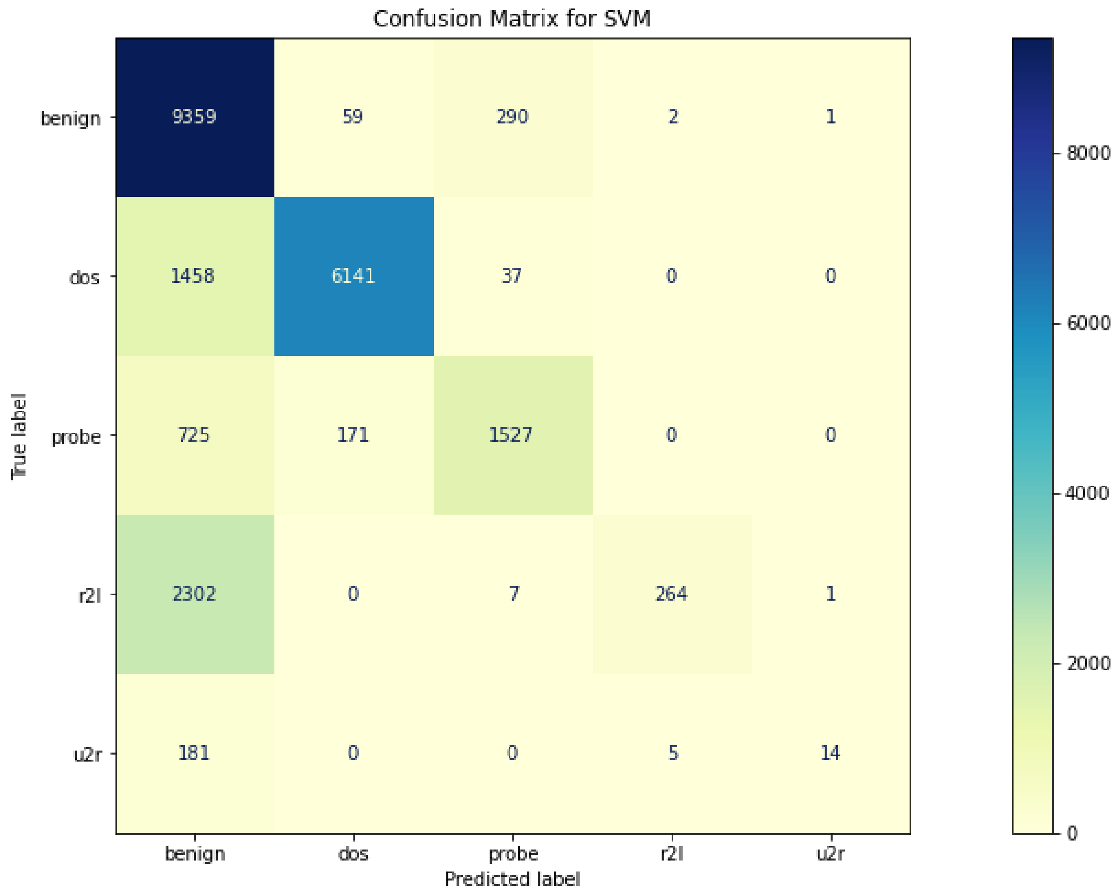
[19]: Text(0.5, 1.0, 'Confusion Matrix for KNN')



4.16 4. SVM

[255]:
```
plt.rcParams["figure.figsize"] = (20,8)
plot_confusion_matrix(model_s, test_x, test_Y, cmap = 'YlGnBu')
plt.title("Confusion Matrix for SVM")
```

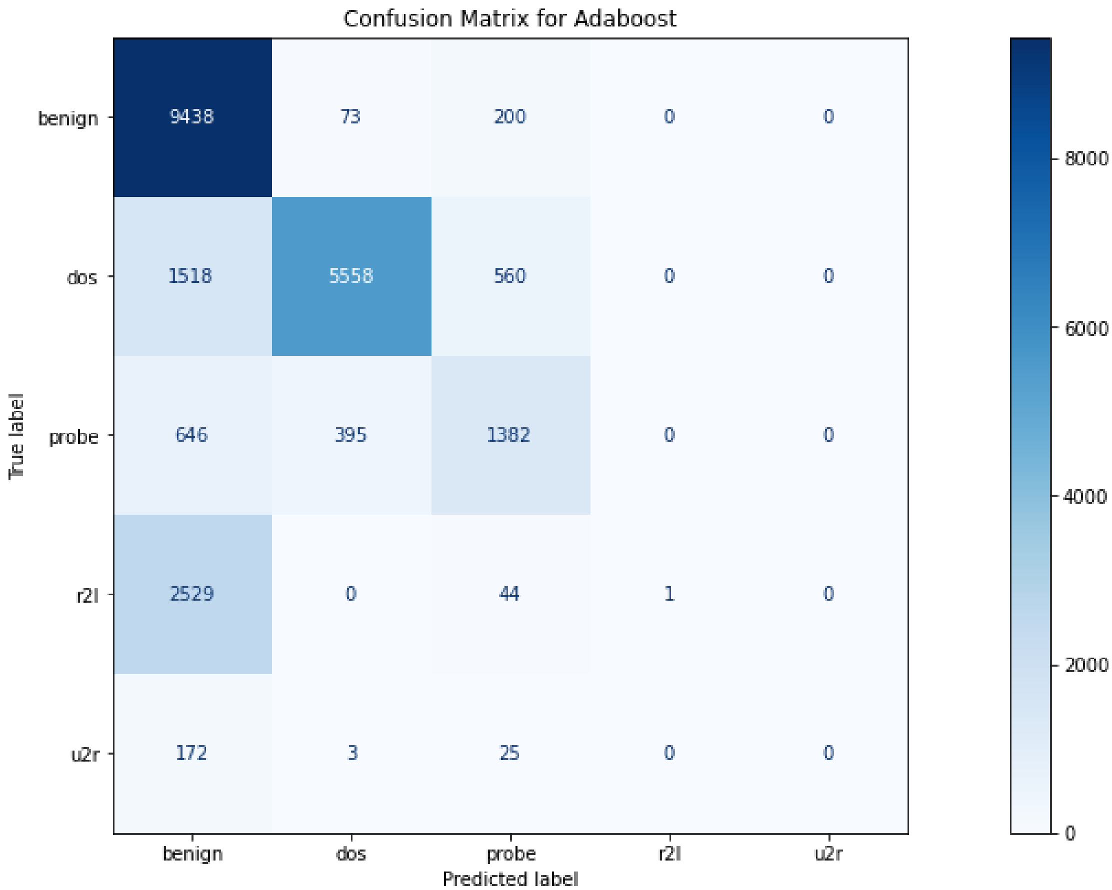
[255]: Text(0.5, 1.0, 'Confusion Matrix for SVM')



4.17 5. Adaboost

[256]:
```
plt.rcParams["figure.figsize"] = (20,8)
plot_confusion_matrix(ada_model, test_x, test_Y, cmap = 'Blues')
plt.title("Confusion Matrix for Adaboost")
```

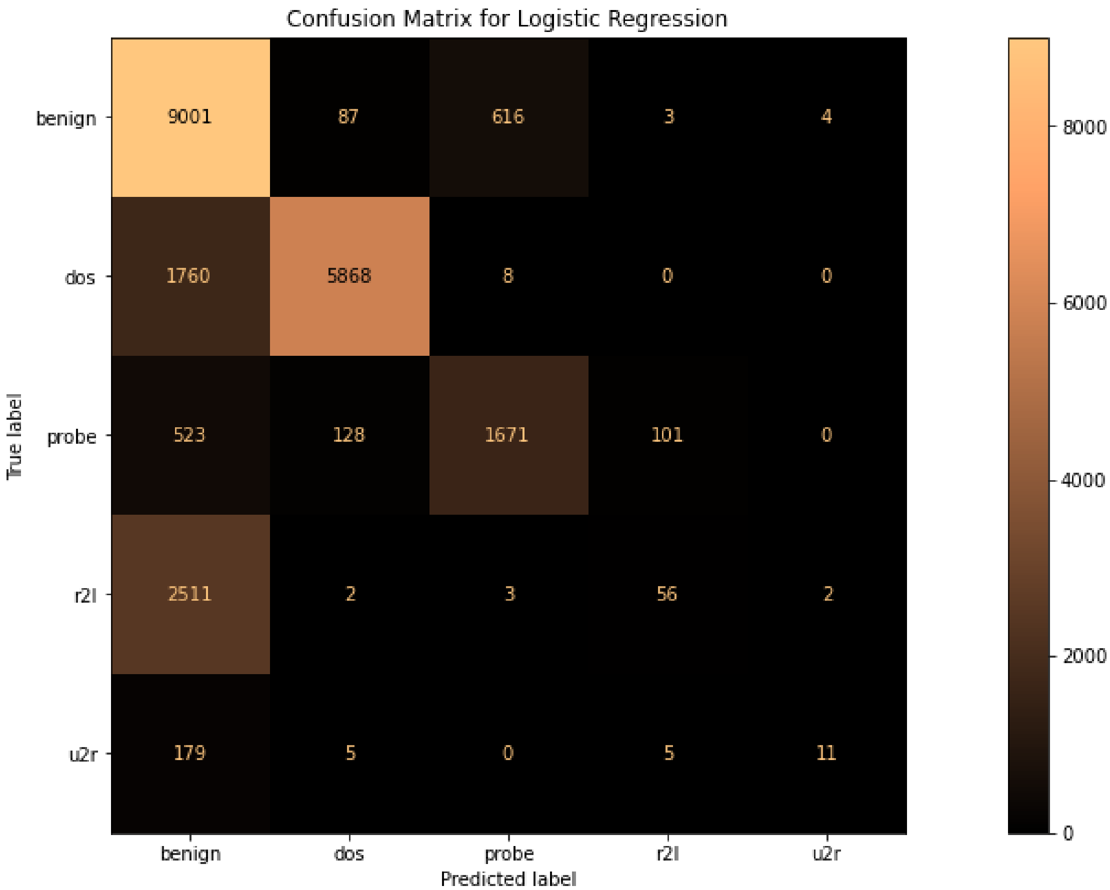
[256]: Text(0.5, 1.0, 'Confusion Matrix for Adaboost')



4.18 6. Logistic Regression

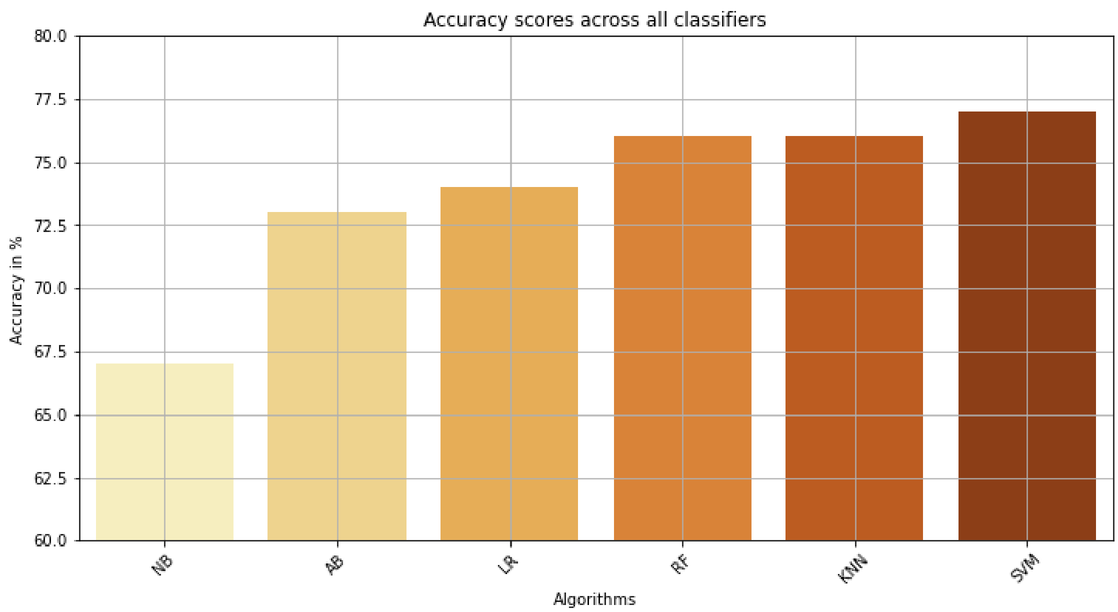
[257]:
```
plt.rcParams["figure.figsize"] = (20,8)
plot_confusion_matrix(LR_search, test_x, test_Y, cmap = 'copper')
plt.title("Confusion Matrix for Logistic Regression")
```

[257]: Text(0.5, 1.0, 'Confusion Matrix for Logistic Regression')



[77]:
```
# Accuracy
x = 'RF','NB','KNN','SVM','AB','LR'
y = 76, 67,76,77,73,74
acc_df = pd.DataFrame({'Algorithms':x, 'Accuracy in %':y})

plt.figure(figsize=(12,6))
plt.xticks(rotation=45)
sns.barplot(x = 'Algorithms' ,y = 'Accuracy in %', data = acc_df,palette =
    'YlOrBr' , order=acc_df.sort_values('Accuracy in %').Algorithms)
plt.title('Accuracy scores across all classifiers')
plt.grid()
plt.ylim((60,80))
```

[77]: (60.0, 80.0)



5 Dataset 2

[20]:
```
dataset = pd.read_csv('E:\Deakin\Tri 3 -␣
↪2021\SIT719\W5\Processed_Combined_IoT_dataset.csv')
```

Exploratory Data Analysis

[21]:
```
dataset.head()
```

[21]:
```
   FC1_Read_Input_Register  FC2_Read_Discrete_Value  \
0                 0.495216                 0.499092
1                 0.495216                 0.499092
2                 0.495216                 0.499092
3                 0.495216                 0.499092
4                 0.495216                 0.499092

   FC3_Read_Holding_Register  FC4_Read_Coil  current_temperature  door_state  \
0                   0.488897       0.499405             0.344399           0
1                   0.488897       0.499405             0.344399           0
2                   0.488897       0.499405             0.344399           0
3                   0.488897       0.499405             0.344399           0
4                   0.488897       0.499405             0.344399           0

   fridge_temperature  humidity  latitude  light_status  longitude  \
0            0.930769  0.462511  0.008217             0   0.008112
1            0.588462  0.462511  0.008217             0   0.008112
2            0.076923  0.462511  0.008217             0   0.008112
```

```
3           0.292308  0.462511  0.008217             0   0.008112
4           0.746154  0.462511  0.008217             0   0.008112

   motion_status  pressure  sphone_signal  temp_condition  temperature  \
0              0  0.533556       0.666667             0.2     0.517307
1              0  0.533556       0.666667             0.2     0.517307
2              0  0.533556       0.666667             0.8     0.517307
3              0  0.533556       0.666667             0.8     0.517307
4              0  0.533556       0.666667             0.2     0.517307

   thermostat_status  label
0                  1      0
1                  1      0
2                  1      0
3                  1      0
4                  1      0
```

[22]:
```python
print(dataset.shape)
```

```
(401119, 18)
```

[23]:
```python
print(list(dataset.columns))
```

```
['FC1_Read_Input_Register', 'FC2_Read_Discrete_Value',
'FC3_Read_Holding_Register', 'FC4_Read_Coil', 'current_temperature',
'door_state', 'fridge_temperature', 'humidity', 'latitude', 'light_status',
'longitude', 'motion_status', 'pressure', 'sphone_signal', 'temp_condition',
'temperature', 'thermostat_status', 'label']
```

[24]:
```python
target_cols=list(dataset.columns[-1:])
target_cols
```

[24]:
```
['label']
```

[25]:
```python
feature_cols= list(dataset.columns[:-1])
feature_cols
```

[25]:
```
['FC1_Read_Input_Register',
 'FC2_Read_Discrete_Value',
 'FC3_Read_Holding_Register',
 'FC4_Read_Coil',
 'current_temperature',
 'door_state',
 'fridge_temperature',
 'humidity',
 'latitude',
 'light_status',
 'longitude',
```

```
'motion_status',
'pressure',
'sphone_signal',
'temp_condition',
'temperature',
'thermostat_status']
```

Split Dataset

[26]:
```
#split dataset in features and target variable
X = dataset.drop('label', axis=1) # Features
y = dataset['label'] # Target variable
```

[27]:
```
X.head()
```

[27]:
```
   FC1_Read_Input_Register  FC2_Read_Discrete_Value  \
0                 0.495216                 0.499092
1                 0.495216                 0.499092
2                 0.495216                 0.499092
3                 0.495216                 0.499092
4                 0.495216                 0.499092

   FC3_Read_Holding_Register  FC4_Read_Coil  current_temperature  door_state  \
0                   0.488897       0.499405             0.344399           0
1                   0.488897       0.499405             0.344399           0
2                   0.488897       0.499405             0.344399           0
3                   0.488897       0.499405             0.344399           0
4                   0.488897       0.499405             0.344399           0

   fridge_temperature  humidity  latitude  light_status  longitude  \
0            0.930769  0.462511  0.008217             0   0.008112
1            0.588462  0.462511  0.008217             0   0.008112
2            0.076923  0.462511  0.008217             0   0.008112
3            0.292308  0.462511  0.008217             0   0.008112
4            0.746154  0.462511  0.008217             0   0.008112

   motion_status  pressure  sphone_signal  temp_condition  temperature  \
0              0  0.533556       0.666667             0.2     0.517307
1              0  0.533556       0.666667             0.2     0.517307
2              0  0.533556       0.666667             0.8     0.517307
3              0  0.533556       0.666667             0.8     0.517307
4              0  0.533556       0.666667             0.2     0.517307

   thermostat_status
0                  1
1                  1
2                  1
3                  1
```

4   1

Splitting Data

[28]:
```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=1) # 70% training and 30% test
```

[29]:
```
# Check the shape of all of these
print("X_train shape is : ", X_train.shape)
print("X_test shape  is : ", X_test.shape)
print("y_train shape is : ", y_train.shape)
print("y_test shape is  : ", y_test.shape)
```

```
X_train shape is :  (280783, 17)
X_test shape  is :  (120336, 17)
y_train shape is :  (280783,)
y_test shape is  :  (120336,)
```

Building Model

5.1 1. Random Forest

[37]:
```
#Calculate start time
start = timeit.default_timer()

#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

#Calculate Stop time
stop = timeit.default_timer()
train_time= stop - start
```

[38]:
```
#Calculate start time
start = timeit.default_timer()

# Predict the model
y_pred=clf.predict(X_test)

#Calculate Stop time
stop = timeit.default_timer()
test_time= stop - start
```

[39]:
```
random_matrix = confusion_matrix(y_test,y_pred)
print(classification_report(y_test,y_pred))
print(random_matrix)
```

```
              precision    recall  f1-score   support

           0       0.85      0.95      0.90     73495
           1       0.91      0.74      0.82     46841

    accuracy                           0.87    120336
   macro avg       0.88      0.85      0.86    120336
weighted avg       0.87      0.87      0.87    120336

[[69920  3575]
 [12070 34771]]
```

[40]:
```
#Train time
print('Train Time(s): ',train_time)

#Test time
print('Test Time(s): ',test_time)
```

```
Train Time(s):  311.38271350000014
Test Time(s):  15.256666600000244
```

[41]:
```
random_eval = metrics(random_matrix)

print('Accuracy for each attack class',random_eval[0])
print('\n')
print('Precision for each attack class',random_eval[1])
print('\n')
print('Recall for each attack class',random_eval[2])
print('\n')
print('F Score for each attack class',random_eval[3])
print('\n')
print('FPR for each attack class',random_eval[4])
```

```
Accuracy for each attack class [87. 87.]


Precision for each attack class [85.28 90.68]


Recall for each attack class [95.14 74.23]


F Score for each attack class [89.94 81.63]


FPR for each attack class [25.77  4.86]
```

5.2 2. Naive Bayes

[42]:
```
#Calculate start time
start = timeit.default_timer()

#Create a NB Classifier

gnb = GaussianNB()
params_NB = {'var_smoothing': np.logspace(0,-9, num=100)}


NB_model = RandomizedSearchCV(estimator=gnb, param_distributions = params_NB,
                     verbose=1,
                     scoring='accuracy')

NB_model.fit(X_train,y_train)

#Calculate Stop time
stop = timeit.default_timer()
train_time= stop - start
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

[43]:
```
#Calculate start time
start = timeit.default_timer()

# Predict the model
gnb_pred= NB_model.predict(X_test)

#Calculate Stop time
stop = timeit.default_timer()
test_time= stop - start
```

[44]:
```
gnb_matrix = confusion_matrix(y_test,gnb_pred)
print(classification_report(y_test,gnb_pred))
print(gnb_matrix)
```

```
              precision    recall  f1-score   support

           0       0.69      0.94      0.79     73495
           1       0.77      0.33      0.46     46841

    accuracy                           0.70    120336
   macro avg       0.73      0.63      0.63    120336
weighted avg       0.72      0.70      0.66    120336

[[68741  4754]
 [31360 15481]]
```

[45]:
```
#Train time
print('Train Time(s): ',train_time)

#Test time
print('Test Time(s): ',test_time)
```

```
Train Time(s):  37.07624699999997
Test Time(s):  0.2857869999998002
```

[46]:
```
gnb_eval = metrics(gnb_matrix)

print('Accuracy for each attack class',gnb_eval[0])
print('\n')
print('Precision for each attack class',gnb_eval[1])
print('\n')
print('Recall for each attack class',gnb_eval[2])
print('\n')
print('F Score for each attack class',gnb_eval[3])
print('\n')
print('FPR for each attack class',gnb_eval[4])
```

```
Accuracy for each attack class [69.99 69.99]


Precision for each attack class [68.67 76.51]


Recall for each attack class [93.53 33.05]


F Score for each attack class [79.19 46.16]


FPR for each attack class [66.95  6.47]
```

5.3 3. KNN

[30]:
```
#Calculate start time
start = timeit.default_timer()

parameters = {"leaf_size" : [5,10,15,20,25,30,35,40]}

knn_model = KNeighborsClassifier()

#knn_model = RandomizedSearchCV(model_k, param_distributions =␣
↪parameters,scoring='accuracy',verbose = 1)

knn_model.fit(X_train,y_train)
```

```python

#Calculate Stop time
stop = timeit.default_timer()
train_time= stop - start

#print("Best Hyper Parameters:\n",knn_model.best_params_)
```

[31]:
```python
#Calculate start time
start = timeit.default_timer()

knn_pred =knn_model.predict(X_test)

#Calculate Stop time
stop = timeit.default_timer()
test_time= stop - start
```

[32]:
```python
knn_matrix = confusion_matrix(y_test,knn_pred)
print(classification_report(y_test,knn_pred))
print(knn_matrix)
```

```
              precision    recall  f1-score   support

           0       0.82      0.93      0.87     73495
           1       0.86      0.68      0.76     46841

    accuracy                           0.83    120336
   macro avg       0.84      0.81      0.82    120336
weighted avg       0.84      0.83      0.83    120336

[[68291  5204]
 [14867 31974]]
```

[33]:
```python
#Train time
print('Train Time(s): ',train_time)

#Test time
print('Test Time(s): ',test_time)
```

```
Train Time(s):  0.1738517999999658
Test Time(s):  2190.7781003
```

[34]:
```python
knn_eval = metrics(knn_matrix)

print('Accuracy for each attack class',knn_eval[0])
print('\n')
print('Precision for each attack class',knn_eval[1])
print('\n')
print('Recall for each attack class',knn_eval[2])
```

```
print('\n')
print('F Score for each attack class',knn_eval[3])
print('\n')
print('FPR for each attack class',knn_eval[4])
```

```
Accuracy for each attack class [83.32 83.32]


Precision for each attack class [82.12 86.  ]


Recall for each attack class [92.92 68.26]


F Score for each attack class [87.19 76.11]


FPR for each attack class [31.74  7.08]
```

5.4 4. CART

[47]:
```
#Calculate start time
start = timeit.default_timer()

cart_model = DecisionTreeClassifier()

cart_params = {'max_depth': range(1, 11),
               "min_samples_split": [2, 3, 4]}

cart_cv = RandomizedSearchCV(cart_model, param_distributions = cart_params,␣
 ↪verbose = 1)

cart_cv.fit(X_train, y_train)

#Calculate Stop time
stop = timeit.default_timer()
train_time= stop - start
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

[48]:
```
cart_cv.best_params_
```

[48]:
```
{'min_samples_split': 4, 'max_depth': 9}
```

[49]:
```
#Calculate start time
start = timeit.default_timer()
cart_pred = cart_cv.predict(X_test)
```

```
#Calculate Stop time
stop = timeit.default_timer()
test_time= stop - start
```

[50]:
```
cart_matrix = confusion_matrix(y_test,cart_pred)
print(classification_report(y_test,cart_pred))
print(cart_matrix)
```

```
              precision    recall  f1-score   support

           0       0.77      0.96      0.85     73495
           1       0.89      0.55      0.68     46841

    accuracy                           0.80    120336
   macro avg       0.83      0.75      0.77    120336
weighted avg       0.82      0.80      0.78    120336

[[70360  3135]
 [21208 25633]]
```

[51]:
```
#Train time
print('Train Time(s): ',train_time)

#Test time
print('Test Time(s): ',test_time)
```

```
Train Time(s):  92.84460470000067
Test Time(s):  0.20801899999969464
```

[52]:
```
cart_eval = metrics(cart_matrix)

print('Accuracy for each attack class',cart_eval[0])
print('\n')
print('Precision for each attack class',cart_eval[1])
print('\n')
print('Recall for each attack class',cart_eval[2])
print('\n')
print('F Score for each attack class',cart_eval[3])
print('\n')
print('FPR for each attack class',cart_eval[4])
```

```
Accuracy for each attack class [79.77 79.77]


Precision for each attack class [76.84 89.1 ]


Recall for each attack class [95.73 54.72]
```

```
F Score for each attack class [85.25 67.8 ]
```

```
FPR for each attack class [45.28  4.27]
```

5.5 5. Logistic Regression

[53]:
```
#Calculate start time
start = timeit.default_timer()

LR = LogisticRegression()

param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    'max_iter': list(range(100,800,100)),
}

LR_model = RandomizedSearchCV(LR, param_distributions=param_grid, verbose = 1)

LR_model.fit(X_train, y_train)

#Calculate Stop time
stop = timeit.default_timer()
train_time= stop - start
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

[54]:
```
LR_model.best_params_
```

[54]:
```
{'max_iter': 200, 'C': 1000}
```

[55]:
```
#Calculate start time
start = timeit.default_timer()
lr_pred = LR_model.predict(X_test)

#Calculate Stop time
stop = timeit.default_timer()
test_time= stop - start
```

[56]:
```
lr_matrix = confusion_matrix(y_test,lr_pred)
print(classification_report(y_test,lr_pred))
print(lr_matrix)
```

```
              precision    recall  f1-score   support

           0       0.67      0.98      0.79     73495
           1       0.88      0.24      0.37     46841
```

```
    accuracy                            0.69    120336
   macro avg       0.77      0.61      0.58    120336
weighted avg       0.75      0.69      0.63    120336

[[71960  1535]
 [35794 11047]]
```

[57]:
```
#Train time
print('Train Time(s): ',train_time)

#Test time
print('Test Time(s): ',test_time)
```

```
Train Time(s):  939.2908571000007
Test Time(s):  0.08327809999991587
```

[58]:
```
lr_eval = metrics(lr_matrix)

print('Accuracy for each attack class',lr_eval[0])
print('\n')
print('Precision for each attack class',lr_eval[1])
print('\n')
print('Recall for each attack class',lr_eval[2])
print('\n')
print('F Score for each attack class',lr_eval[3])
print('\n')
print('FPR for each attack class',lr_eval[4])
```

```
Accuracy for each attack class [68.98 68.98]


Precision for each attack class [66.78 87.8 ]


Recall for each attack class [97.91 23.58]


F Score for each attack class [79.4  37.18]


FPR for each attack class [76.42  2.09]
```

5.6 6. LDA

[59]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

start = timeit.default_timer()

lda = LinearDiscriminantAnalysis()
```

```
parameters = {"n_components" : [1,2,3,4,5]}


lda_model = RandomizedSearchCV(lda, param_distributions=parameters, verbose = 1)

lda_model.fit(X_train, y_train)

#Calculate Stop time
stop = timeit.default_timer()
train_time= stop - start
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
```

[60]:
```
lda_model.best_params_
```

[60]:
```
{'n_components': 1}
```

[61]:
```
#Calculate start time
start = timeit.default_timer()

lda_pred = lda_model.predict(X_test)

#Calculate Stop time
stop = timeit.default_timer()
test_time= stop - start
```

[62]:
```
lda_matrix = confusion_matrix(y_test,lda_pred)
print(classification_report(y_test,lda_pred))
print(lda_matrix)
```

```
              precision    recall  f1-score   support

           0       0.66      0.98      0.79     73495
           1       0.87      0.22      0.35     46841

    accuracy                           0.68    120336
   macro avg       0.77      0.60      0.57    120336
weighted avg       0.74      0.68      0.62    120336

[[72017  1478]
 [36662 10179]]
```

[63]:
```
#Train time
print('Train Time(s): ',train_time)

#Test time
print('Test Time(s): ',test_time)
```

```
Train Time(s):  32.00438490000033
Test Time(s):  0.05626200000006065
```

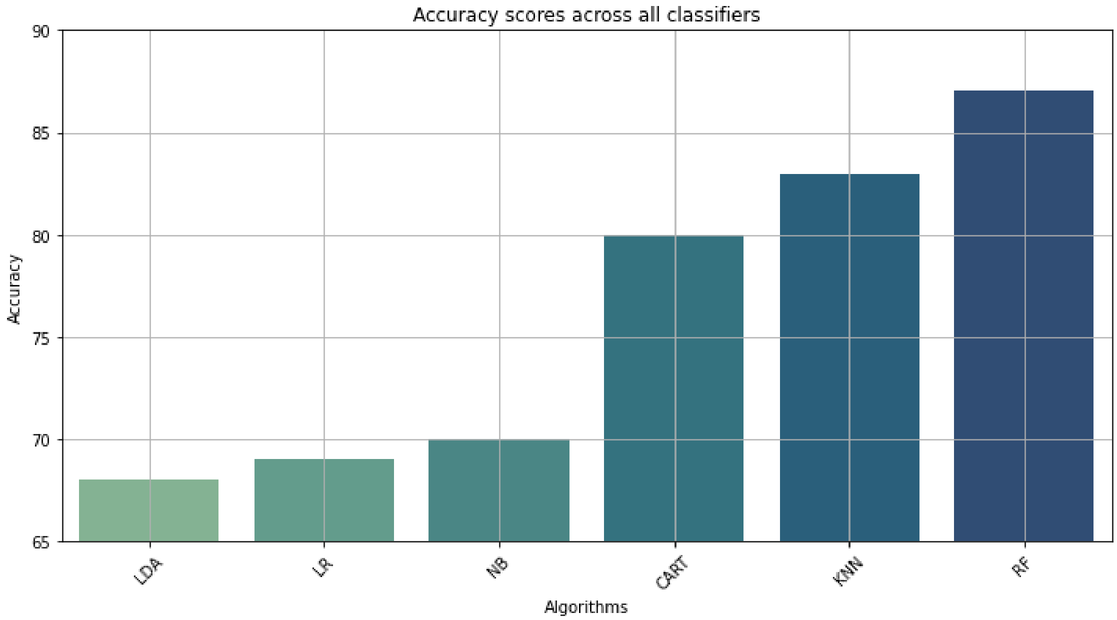
[64]:
```
lda_eval = metrics(lda_matrix)

print('Accuracy for each attack class',lda_eval[0])
print('\n')
print('Precision for each attack class',lda_eval[1])
print('\n')
print('Recall for each attack class',lda_eval[2])
print('\n')
print('F Score for each attack class',lda_eval[3])
print('\n')
print('FPR for each attack class',lda_eval[4])
```

```
Accuracy for each attack class [68.31 68.31]


Precision for each attack class [66.27 87.32]


Recall for each attack class [97.99 21.73]


F Score for each attack class [79.07 34.8 ]


FPR for each attack class [78.27  2.01]
```

Visualization

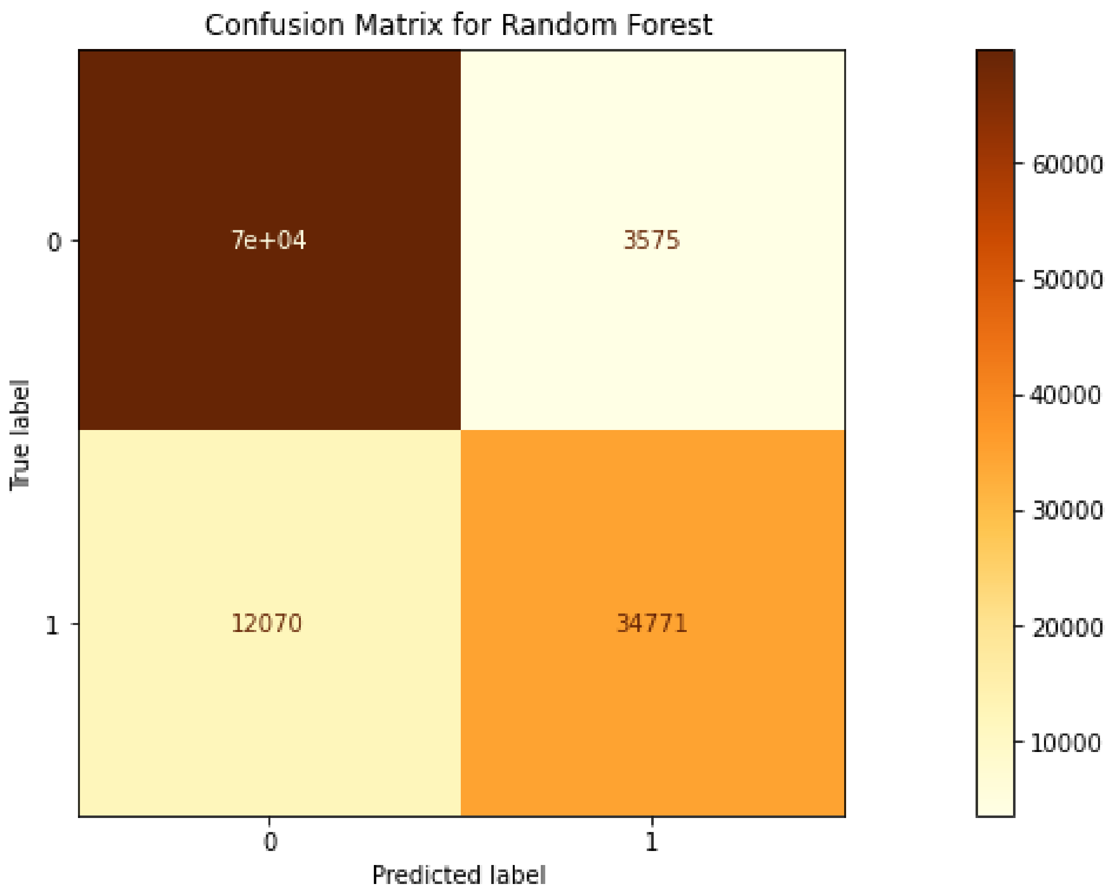
[85]:
```
# Accuracy
x = 'RF','NB','KNN','CART','LR','LDA'
y = 87,70,83,80,69,68
acc_df = pd.DataFrame({'Algorithms':x, 'Accuracy':y})

plt.figure(figsize=(12,6))
plt.xticks(rotation=45)
sns.barplot(x = 'Algorithms' ,y = 'Accuracy' , data = acc_df,palette = 'crest'␣
 ↪, order=acc_df.sort_values('Accuracy').Algorithms)
plt.title('Accuracy scores across all classifiers')
plt.grid()
plt.ylim((65,90))
```

[85]: (65.0, 90.0)



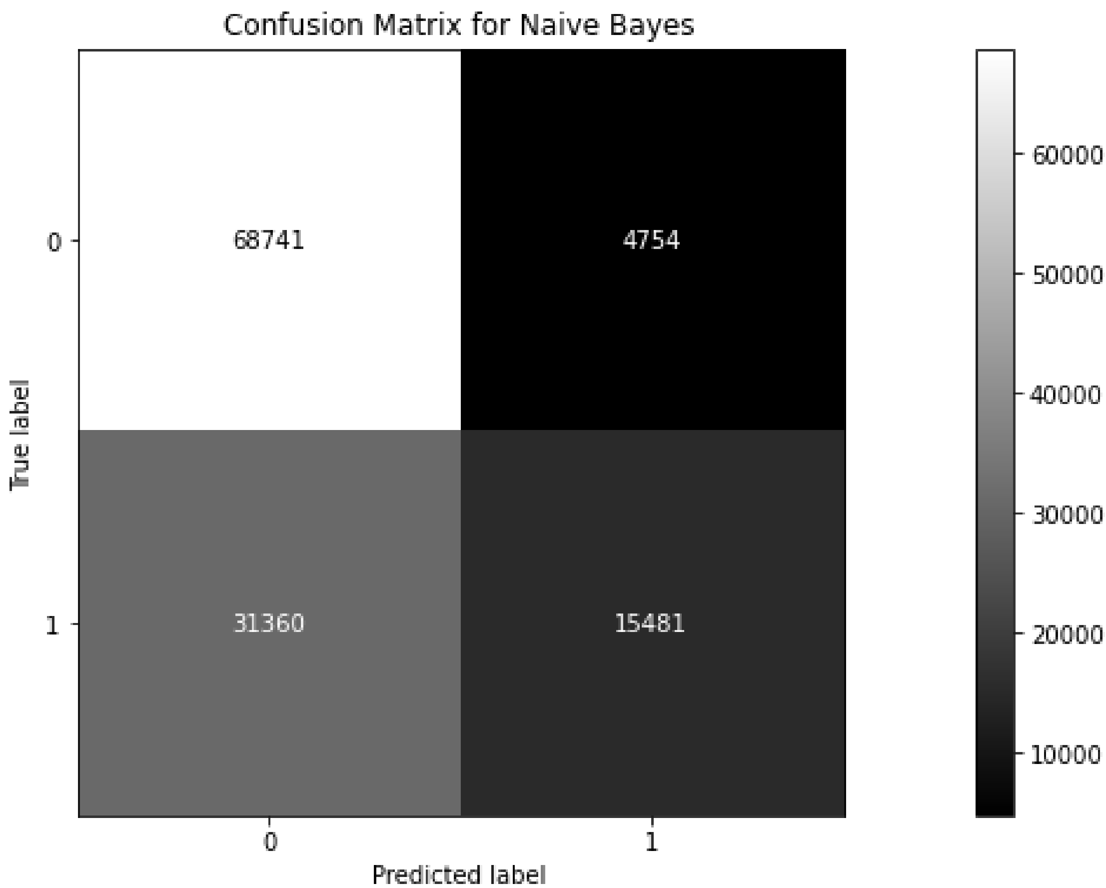
[66]:
```
plt.rcParams["figure.figsize"] = (20,6)
plot_confusion_matrix(clf, X_test, y_test, cmap = 'YlOrBr')
plt.title("Confusion Matrix for Random Forest")
```

[66]: Text(0.5, 1.0, 'Confusion Matrix for Random Forest')



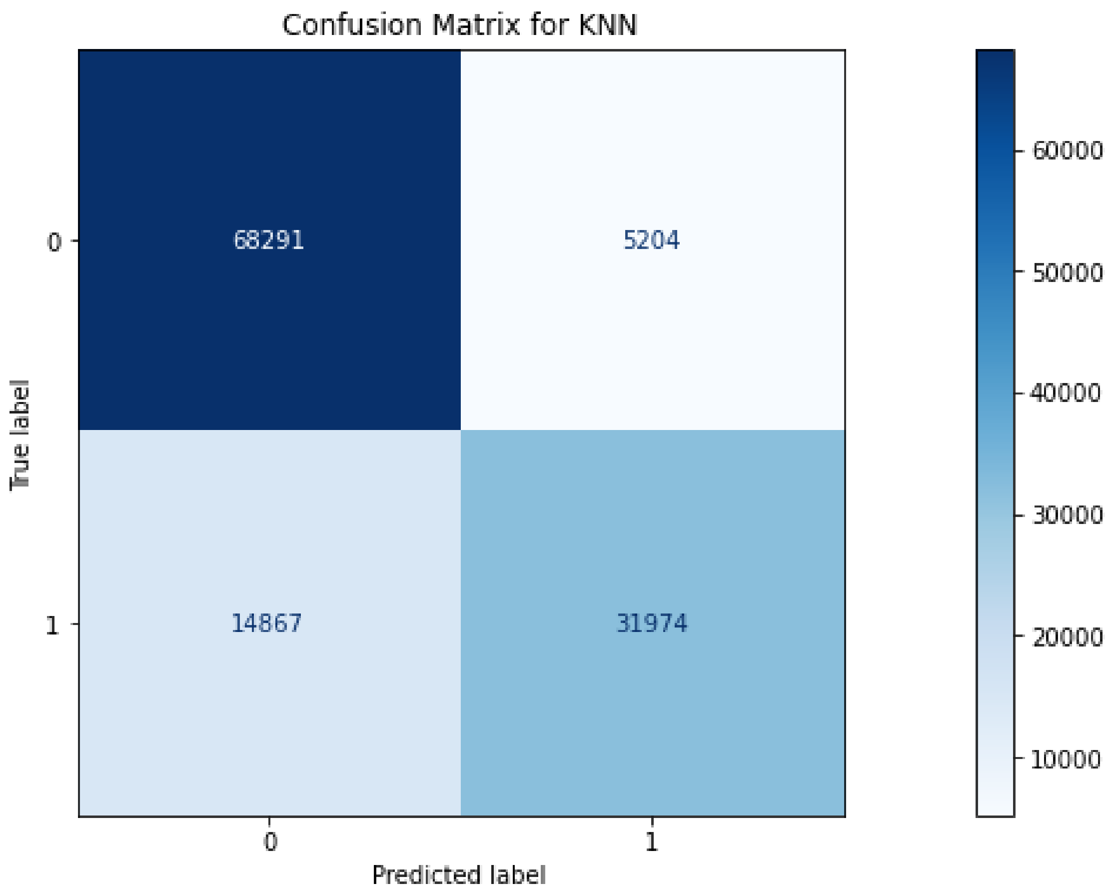

```
[67]: plt.rcParams["figure.figsize"] = (20,6)
      plot_confusion_matrix(NB_model, X_test, y_test, cmap = 'gray')
      plt.title("Confusion Matrix for Naive Bayes")
```

```
[67]: Text(0.5, 1.0, 'Confusion Matrix for Naive Bayes')
```



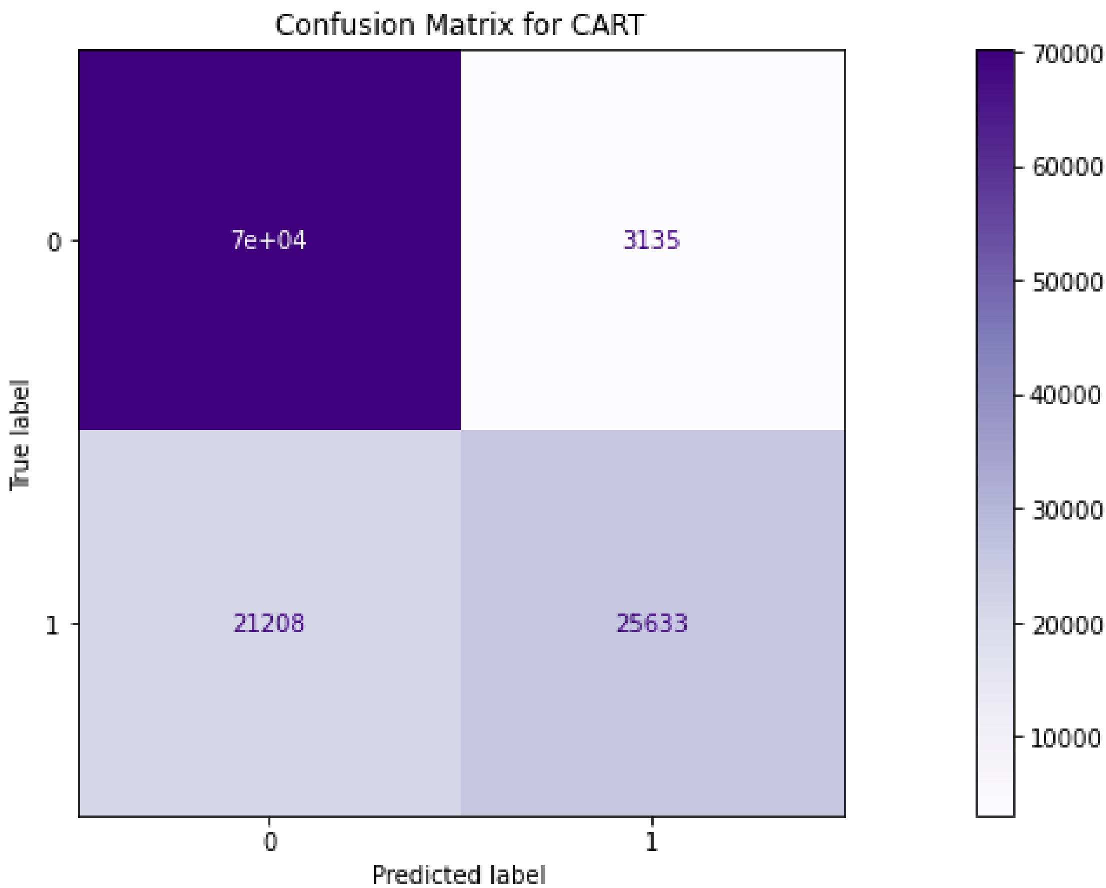
[36]:
```
plt.rcParams["figure.figsize"] = (20,6)
plot_confusion_matrix(knn_model, X_test, y_test, cmap = 'Blues')
plt.title("Confusion Matrix for KNN")
```

[36]: Text(0.5, 1.0, 'Confusion Matrix for KNN')



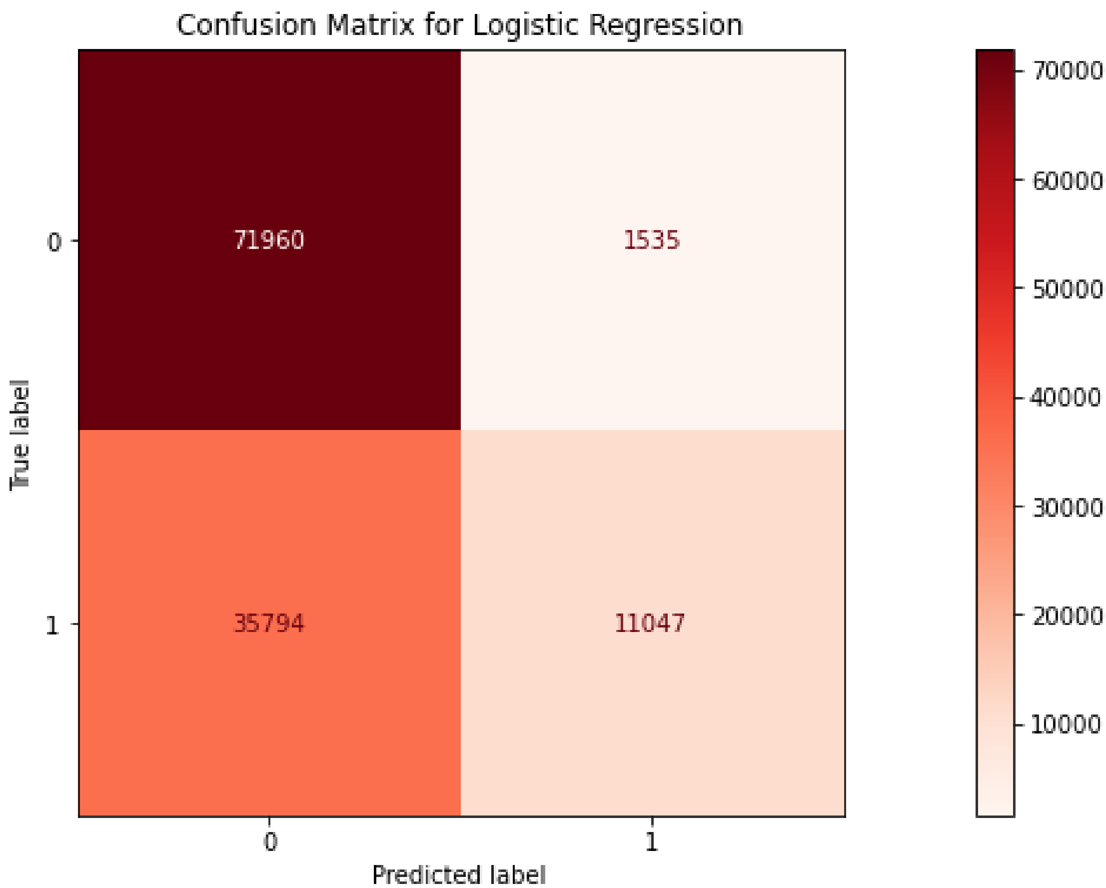
[68]:
```
plt.rcParams["figure.figsize"] = (20,6)
plot_confusion_matrix(cart_cv, X_test, y_test, cmap = 'Purples')
plt.title("Confusion Matrix for CART")
```

[68]: Text(0.5, 1.0, 'Confusion Matrix for CART')



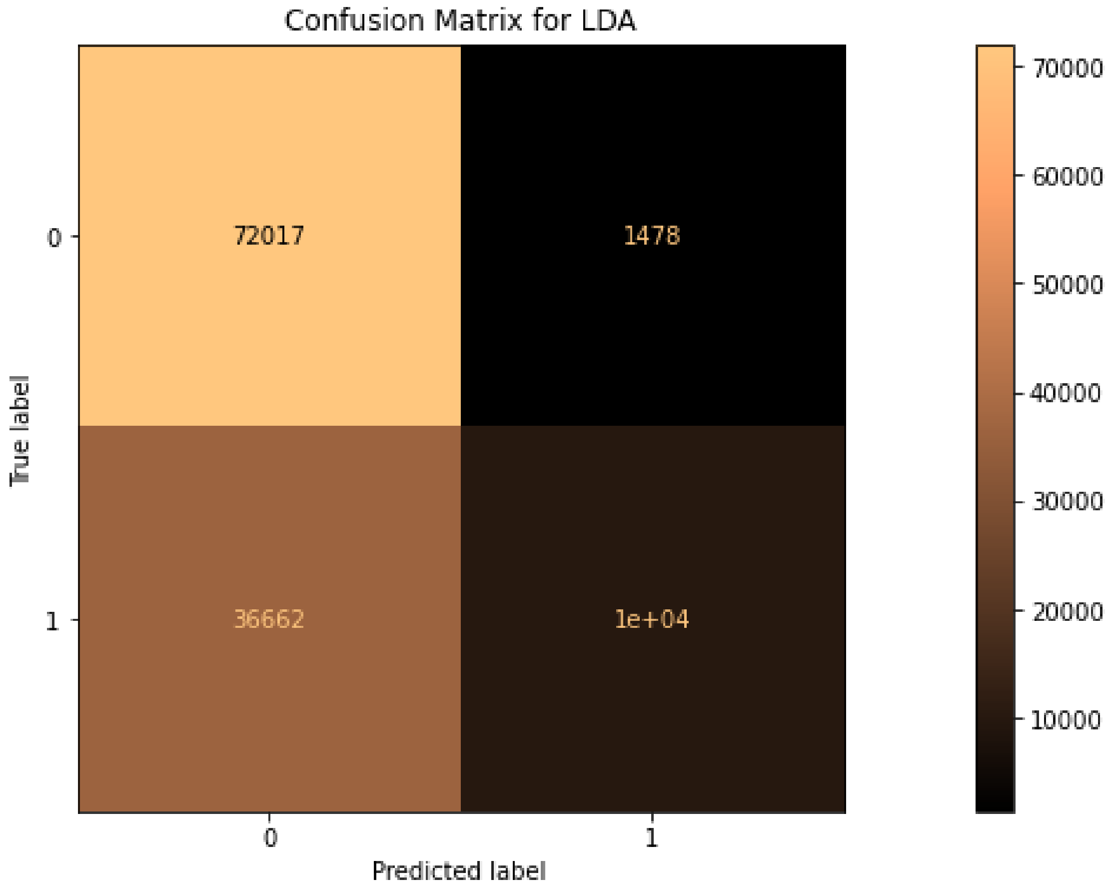

```
[69]: plt.rcParams["figure.figsize"] = (20,6)
      plot_confusion_matrix(LR_model, X_test, y_test, cmap = 'Reds')
      plt.title("Confusion Matrix for Logistic Regression")
```

```
[69]: Text(0.5, 1.0, 'Confusion Matrix for Logistic Regression')
```



[70]:
```
plt.rcParams["figure.figsize"] = (20,6)
plot_confusion_matrix(lda_model, X_test, y_test, cmap = 'copper')
plt.title("Confusion Matrix for LDA")
```

[70]: Text(0.5, 1.0, 'Confusion Matrix for LDA')


Confusion Matrix for LDA
72017
1478
36662
1e+04
0
1
True label
0
1
Predicted label
70000
60000
50000
40000
30000
20000
10000