#!/usr/bin/env python

# coding: utf-8

# ## Environment Setup

# In[3]:

# import relevant modules

get\_ipython().run\_line\_magic('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

# Ignore warnings

import warnings

warnings.filterwarnings('ignore')

# Settings

pd.set\_option('display.max\_columns', None)

np.set\_printoptions(threshold=np.nan)

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\_\_))

# ## Load Data

# In[5]:

# 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\_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", "last\_flag"]

# Load NSL\_KDD train dataset

dfkdd\_train = pd.read\_table("train\_data.txt", sep=",", names=datacols) # change path to where the dataset is located.

dfkdd\_train = dfkdd\_train.iloc[:,:-1] # removes an unwanted extra field

# Load NSL\_KDD test dataset

dfkdd\_test = pd.read\_table("test\_data.txt", sep=",", names=datacols)

dfkdd\_test = dfkdd\_test.iloc[:,:-1]

# ### Train dataset

# In[7]:

# View train data

dfkdd\_train.head(3)

# train set dimension

print('Train set dimension: {} rows, {} columns'.format(dfkdd\_train.shape[0], dfkdd\_train.shape[1]))

# ### Test dataset

# In[8]:

# View test data

dfkdd\_test.head(3)

# test set dimension

print('Test set dimension: {} rows, {} columns'.format(dfkdd\_test.shape[0], dfkdd\_test.shape[1]))

# ## Data Preprocessing

# ### Map attack field to attack class

# NSL-KDD dataset has 42 attributes for each connection record including class label containing attack types. The attack types are categorized into four attack classes as described by Mahbod Tavallaee et al. in [\_A Detailed analysis of the KDD CUP 99 Data Set\_](http://www.ee.ryerson.ca/~bagheri/papers/cisda.pdf) as:

# 1. \*\*Denial of Service (DoS)\*\*: is an attack in which an adversary directed a deluge of traffic requests to a system in order to make the computing or memory resource too busy or too full to handle legitimate requests and in the process, denies legitimate users access to a machine.

# 2. \*\*Probing Attack (Probe)\*\*: probing network of computers to gather information to be used to compromise its security controls.

# 3. \*\*User to Root Attack (U2R)\*\*: a class of exploit in which the adversary starts out with access to a normal user account on the system (gained either by sniffing passwords, a dictionary attack, or social engineering) and is able to exploit some vulnerability to gain root access to the system.

# 4. \*\*Remote to Local Attack (R2L)\*\*: occurs when an attacker who has the ability to send packets to a machine over a network but who does not have an account on that machine exploits some vulnerability to gain local access as a user of that machine.

# In[9]:

mapping = {'ipsweep': 'Probe','satan': 'Probe','nmap': 'Probe','portsweep': 'Probe','saint': 'Probe','mscan': 'Probe',

'teardrop': 'DoS','pod': 'DoS','land': 'DoS','back': 'DoS','neptune': 'DoS','smurf': 'DoS','mailbomb': 'DoS',

'udpstorm': 'DoS','apache2': 'DoS','processtable': 'DoS',

'perl': 'U2R','loadmodule': 'U2R','rootkit': 'U2R','buffer\_overflow': 'U2R','xterm': 'U2R','ps': 'U2R',

'sqlattack': 'U2R','httptunnel': 'U2R',

'ftp\_write': 'R2L','phf': 'R2L','guess\_passwd': 'R2L','warezmaster': 'R2L','warezclient': 'R2L','imap': 'R2L',

'spy': 'R2L','multihop': 'R2L','named': 'R2L','snmpguess': 'R2L','worm': 'R2L','snmpgetattack': 'R2L',

'xsnoop': 'R2L','xlock': 'R2L','sendmail': 'R2L',

'normal': 'Normal'

}

# In[10]:

# Apply attack class mappings to the dataset

dfkdd\_train['attack\_class'] = dfkdd\_train['attack'].apply(lambda v: mapping[v])

dfkdd\_test['attack\_class'] = dfkdd\_test['attack'].apply(lambda v: mapping[v])

# In[11]:

# 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[12]:

# View top 3 train data

dfkdd\_train.head(3)

# ### Exploratory Data Analysis

# In[13]:

# Descriptive statistics

dfkdd\_train.describe()

# In[14]:

dfkdd\_train['num\_outbound\_cmds'].value\_counts()

dfkdd\_test['num\_outbound\_cmds'].value\_counts()

# In[15]:

# 'num\_outbound\_cmds' field has all 0 values. Hence, it will be removed from both train and test dataset since it is a redundant field.

dfkdd\_train.drop(['num\_outbound\_cmds'], axis=1, inplace=True)

dfkdd\_test.drop(['num\_outbound\_cmds'], axis=1, inplace=True)

# In[17]:

# Attack Class Distribution

attack\_class\_freq\_train = dfkdd\_train[['attack\_class']].apply(lambda x: x.value\_counts())

attack\_class\_freq\_test = dfkdd\_test[['attack\_class']].apply(lambda x: x.value\_counts())

attack\_class\_freq\_train['frequency\_percent\_train'] = round((100 \* attack\_class\_freq\_train / attack\_class\_freq\_train.sum()),2)

attack\_class\_freq\_test['frequency\_percent\_test'] = round((100 \* attack\_class\_freq\_test / attack\_class\_freq\_test.sum()),2)

attack\_class\_dist = pd.concat([attack\_class\_freq\_train,attack\_class\_freq\_test], axis=1)

attack\_class\_dist

# In[18]:

# Attack class bar plot

plot = attack\_class\_dist[['frequency\_percent\_train', 'frequency\_percent\_test']].plot(kind="bar");

plot.set\_title("Attack Class Distribution", fontsize=20);

plot.grid(color='lightgray', alpha=0.5);

# In[19]:

dfkdd\_train.head()

# ### Scaling Numerical Attributes

# In[20]:

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# extract numerical attributes and scale it to have zero mean and unit variance

cols = dfkdd\_train.select\_dtypes(include=['float64','int64']).columns

sc\_train = scaler.fit\_transform(dfkdd\_train.select\_dtypes(include=['float64','int64']))

sc\_test = scaler.fit\_transform(dfkdd\_test.select\_dtypes(include=['float64','int64']))

# turn the result back to a dataframe

sc\_traindf = pd.DataFrame(sc\_train, columns = cols)

sc\_testdf = pd.DataFrame(sc\_test, columns = cols)

# ### Encoding of Categorical Attributes

# In[21]:

from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

# extract categorical attributes from both training and test sets

cattrain = dfkdd\_train.select\_dtypes(include=['object']).copy()

cattest = dfkdd\_test.select\_dtypes(include=['object']).copy()

# encode the categorical attributes

traincat = cattrain.apply(encoder.fit\_transform)

testcat = cattest.apply(encoder.fit\_transform)

# separate target column from encoded data

enctrain = traincat.drop(['attack\_class'], axis=1)

enctest = testcat.drop(['attack\_class'], axis=1)

cat\_Ytrain = traincat[['attack\_class']].copy()

cat\_Ytest = testcat[['attack\_class']].copy()

# ### Data Sampling

# In[22]:

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\_sample(X, y)

print('Original dataset shape {}'.format(Counter(y)))

print('Resampled dataset shape {}'.format(Counter(y\_res)))

# ### Feature Selection

# In[23]:

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\_index('feature')

# plot importances

plt.rcParams['figure.figsize'] = (11, 4)

importances.plot.bar();

# In[24]:

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(), refclasscol)]

selected\_features = [v for i, v in feature\_map if i==True]

# In[25]:

selected\_features

# ### Dataset Partition

# In[26]:

# 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

# In[27]:

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 labels'''

for j, k in normalclass:

for i, v in attacklist:

restrain\_set = res\_df.loc[(res\_df['attack\_class'] == k) | (res\_df['attack\_class'] == v)]

classdict[j +'\_' + i].append(restrain\_set)

# test labels

reftest\_set = reftest.loc[(reftest['attack\_class'] == k) | (reftest['attack\_class'] == v)]

classdict[j +'\_' + i].append(reftest\_set)

create\_classdict()

# In[28]:

for k, v in classdict.items():

k

# In[29]:

pretrain = classdict['Normal\_DoS'][0]

pretest = classdict['Normal\_DoS'][1]

grpclass = 'Normal\_DoS'

# ### Finalize data preprocessing for training

# In[30]:

from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder()

Xresdf = pretrain

newtest = pretest

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[31]:

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 LogisticRegression Model

LGR\_Classifier = LogisticRegression(n\_jobs=-1, random\_state=0)

LGR\_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', random\_state=0)

DTC\_Classifier.fit(X\_train, Y\_train);

# Train RandomForestClassifier Model

#RF\_Classifier = RandomForestClassifier(criterion='entropy', n\_jobs=-1, random\_state=0)

#RF\_Classifier.fit(X\_train, Y\_train);

# Train SVM Model

SVC\_Classifier = SVC(random\_state=0)

SVC\_Classifier.fit(X\_train, Y\_train)

## Train Ensemble Model (This method combines all the individual models above except RandomForest)

#combined\_model = [('Naive Baye Classifier', BNB\_Classifier),

# ('Decision Tree Classifier', DTC\_Classifier),

# ('KNeighborsClassifier', KNN\_Classifier),

# ('LogisticRegression', LGR\_Classifier)

# ]

#VotingClassifier = VotingClassifier(estimators = combined\_model,voting = 'soft', n\_jobs=-1)

#VotingClassifier.fit(X\_train, Y\_train);

# ## Evaluate Models

# In[32]:

from sklearn import metrics

models = []

#models.append(('SVM Classifier', SVC\_Classifier))

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(('LogisticRegression', LGR\_Classifier))

models.append(('SVC\_Classifier', SVC\_Classifier))

#models.append(('VotingClassifier', VotingClassifier))

for i, v in models:

scores = cross\_val\_score(v, X\_train, Y\_train, cv=10)

accuracy = metrics.accuracy\_score(Y\_train, v.predict(X\_train))

confusion\_matrix = metrics.confusion\_matrix(Y\_train, v.predict(X\_train))

classification = metrics.classification\_report(Y\_train, v.predict(X\_train))

print()

print('============================== {} {} Model Evaluation =============================='.format(grpclass, i))

print()

print ("Cross Validation Mean Score:" "\n", scores.mean())

print()

print ("Model Accuracy:" "\n", accuracy)

print()

print("Confusion matrix:" "\n", confusion\_matrix)

print()

print("Classification report:" "\n", classification)

print()

# ## Test Models

# In[33]:

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 =============================='.format(grpclass, i))

print()

print ("Model Accuracy:" "\n", accuracy)

print()

print("Confusion matrix:" "\n", confusion\_matrix)

print()

print("Classification report:" "\n", classification)

print()