**#importing libraries**

**import pandas as pd**

**import numpy as np**

**import matplotlib**

**import matplotlib.pyplot as plt**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.preprocessing import StandardScaler**

**from sklearn.decomposition import PCA**

**from sklearn.pipeline import Pipeline**

**from sklearn.linear\_model import LogisticRegression**

**from sklearn.tree import DecisionTreeClassifier**

**from sklearn.ensemble import RandomForestClassifier**

**#import the dataset**

**df = pd.read\_csv('/content/drive/My Drive/kddcup99\_csv.csv')**

**df.head()**

**df.describe()**

**#preprocessing**

**1.label encode the numerical value**

**from sklearn.preprocessing import LabelEncoder, OneHotEncoder**

**from sklearn.compose import ColumnTransformer**

**labelencoder\_x\_1 = LabelEncoder()**

**labelencoder\_x\_2 = LabelEncoder()**

**labelencoder\_x\_3 = LabelEncoder()**

**x[:, 1] = labelencoder\_x\_1.fit\_transform(x[:, 1])**

**x[:, 2] = labelencoder\_x\_2.fit\_transform(x[:, 2])**

**x[:, 3] = labelencoder\_x\_3.fit\_transform(x[:, 3])**

**2.To check the missing values**

**df.isnull()**

**3.convert into binary class**

**dataset['label'] = dataset['label'].replace(['back', 'buffer\_overflow', 'ftp\_write', 'guess\_passwd', 'imap', 'ipsweep', 'land', 'loadmodule', 'multihop', 'neptune', 'nmap', 'perl', 'phf', 'pod', 'portsweep', 'rootkit', 'satan', 'smurf', 'spy', 'teardrop', 'warezclient', 'warezmaster'], 'attack') dataset['label'] = dataset['label'].replace(['back', 'buffer\_overflow', 'ftp\_write', 'guess\_passwd', 'imap', 'ipsweep', 'land', 'loadmodule', 'multihop', 'neptune', 'nmap', 'perl', 'phf', 'pod', 'portsweep', 'rootkit', 'satan', 'smurf', 'spy', 'teardrop', 'warezclient', 'warezmaster'], 'attack')**

**#feature selection**

**1.Correlating the feature using Heat map to find out the feature selection**

**import seaborn as sns**

**import matplotlib.pyplot as plt**

**#get correlations of each features in dataset**

**corrmat = df.corr()**

**top\_corr\_features = corrmat.index**

**plt.figure(figsize=(20,20))**

**#plot heat map**

**g=sns.heatmap(df[top\_corr\_features].corr(),annot=True,cmap="RdYlGn")**

**2.feature selection by using variance method**

**print(dataset.var()['src\_bytes'])**

**print(dataset.var()['dst\_bytes'])**

**print(dataset.var()['land'])**

**print(dataset.var()['wrong\_fragment'])**

**print(dataset.var()['urgent'])**

**print(dataset.var()['hot'])**

**print(dataset.var()['num\_failed\_logins'])**

**print(dataset.var()['logged\_in'])**

**print(dataset.var()['lnum\_compromised'])**

**print(dataset.var()['lroot\_shell'])**

**print(dataset.var()['lsu\_attempted'])**

**print(dataset.var()['lnum\_root'])**

**print(dataset.var()['lnum\_file\_creations'])**

**print(dataset.var()['lnum\_shells'])**

**print(dataset.var()['lnum\_access\_files'])**

**print(dataset.var()['lnum\_outbound\_cmds'])**

**print(dataset.var()['is\_host\_login'])**

**print(dataset.var()['is\_guest\_login'])**

**print(dataset.var()['count'])**

**print(dataset.var()['srv\_count'])**

**print(dataset.var()['serror\_rate'])**

**print(dataset.var()['srv\_serror\_rate'])**

**print(dataset.var()['rerror\_rate'])**

**print(dataset.var()['srv\_rerror\_rate'])**

**print(dataset.var()['same\_srv\_rate'])**

**print(dataset.var()['diff\_srv\_rate'])**

**print(dataset.var()['srv\_diff\_host\_rate'])**

**print(dataset.var()['dst\_host\_count'])**

**print(dataset.var()['dst\_host\_srv\_count'])**

**print(dataset.var()['dst\_host\_same\_srv\_rate'])**

**print(dataset.var()['dst\_host\_diff\_srv\_rate'])**

**print(dataset.var()['dst\_host\_same\_src\_port\_rate'])**

**print(dataset.var()['dst\_host\_srv\_diff\_host\_rate'])**

**print(dataset.var()['dst\_host\_serror\_rate'])**

**print(dataset.var()['dst\_host\_srv\_serror\_rate'])**

**print(dataset.var()['dst\_host\_rerror\_rate'])**

**print(dataset.var()['dst\_host\_srv\_rerror\_rate'])**

**3.Removethe redundant features**

**f['lnum\_outbound\_cmds'].value\_counts()**

**df.drop('lnum\_outbound\_cmds', axis=1, inplace=True)**

**df['is\_host\_login'].value\_counts()**

**df.drop('is\_host\_login', axis=1, inplace=True)**

**df['wrong\_fragment'].value\_counts()**

**df.drop('wrong\_fragment', axis=1, inplace=True)**

**df['hot'].value\_counts()**

**df.drop('hot', axis=1, inplace=True)**

**df['num\_failed\_logins'].value\_counts()**

**df.drop('num\_failed\_logins', axis=1, inplace=True)**

**df['logged\_in'].value\_counts()**

**df.drop('logged\_in', axis=1, inplace=True)**

**df['lroot\_shell'].value\_counts()**

**df.drop('lroot\_shell', axis=1, inplace=True)**

**df['lnum\_file\_creations'].value\_counts()**

**df.drop('lnum\_file\_creations', axis=1, inplace=True)**

**df['lnum\_shells'].value\_counts()**

**df.drop('lnum\_shells', axis=1, inplace=True)**

**df['lnum\_access\_files'].value\_counts()**

**df.drop('lnum\_access\_files', axis=1, inplace=True)**

**df['is\_guest\_login'].value\_counts()**

**df.drop('is\_guest\_login', axis=1, inplace=True)**

**df['serror\_rate'].value\_counts()**

**df.drop('serror\_rate', axis=1, inplace=True)**

**df['srv\_serror\_rate'].value\_counts()**

**df.drop('srv\_serror\_rate', axis=1, inplace=True)**

**df['rerror\_rate'].value\_counts()**

**df.drop('rerror\_rate', axis=1, inplace=True)**

**df['srv\_rerror\_rate'].value\_counts()**

**df.drop('srv\_rerror\_rate', axis=1, inplace=True)**

**df['same\_srv\_rate'].value\_counts()**

**df.drop('same\_srv\_rate', axis=1, inplace=True)**

**df['diff\_srv\_rate'].value\_counts()**

**df.drop('diff\_srv\_rate', axis=1, inplace=True)**

**df['srv\_diff\_host\_rate'].value\_counts()**

**df.drop('srv\_diff\_host\_rate', axis=1, inplace=True)**

**df['dst\_host\_same\_srv\_rate'].value\_counts()**

**df.drop('dst\_host\_same\_srv\_rate', axis=1, inplace=True)**

**df['dst\_host\_diff\_srv\_rate'].value\_counts()**

**df.drop('dst\_host\_diff\_srv\_rate', axis=1, inplace=True)**

**df['dst\_host\_same\_src\_port\_rate'].value\_counts()**

**df.drop('dst\_host\_same\_src\_port\_rate', axis=1, inplace=True)**

**df['dst\_host\_srv\_diff\_host\_rate'].value\_counts()**

**df.drop('dst\_host\_srv\_diff\_host\_rate', axis=1, inplace=True)**

**df['dst\_host\_serror\_rate'].value\_counts()**

**df.drop('dst\_host\_serror\_rate', axis=1, inplace=True)**

**df['dst\_host\_srv\_serror\_rate'].value\_counts()**

**df.drop('dst\_host\_srv\_serror\_rate', axis=1, inplace=True)**

**df['dst\_host\_rerror\_rate'].value\_counts()**

**df.drop('dst\_host\_rerror\_rate', axis=1, inplace=True)**

**df['dst\_host\_srv\_rerror\_rate'].value\_counts()**

**df.drop('dst\_host\_srv\_rerror\_rate', axis=1, inplace=True)**

**df['lsu\_attempted'].value\_counts()**

**df.drop('lsu\_attempted', axis=1, inplace=True)**

**df['urgent'].value\_counts()**

**df.drop('urgent', axis=1, inplace=True)**

**4.scale the feature using min-max scalar**

**from sklearn.preprocessing import StandardScaler**

**sScaler = StandardScaler()**

**rescaleX = sScaler.fit\_transform(X)**

**rescaleX**

**df\_rescaled = pd.DataFrame(data=rescaleX)**

**df\_rescaled.hist(figsize=(50, 30), bins=20)**

**plt.show()**

**5.feature selection using normalization**

**from sklearn.preprocessing import Normalizer**

**norm = Normalizer()**

**xNormalize = norm.fit\_transform(X)**

**xNormalize**

**df\_Normalized = pd.DataFrame(data=xNormalize)**

**df\_Normalized.hist(figsize=(50, 30), bins=20)**

**plt.show()**

**6.dimensionality reduction using PCA**

**from sklearn.decomposition import PCA**

**pca = PCA(n\_components=3)**

**pca.fit(X,Y)**

**pca.components\_**

**pca.explained\_variance\_**

**pca.transform(X)**

**#Exploratory data analysis**

**1.Log-scaled distribution of attaks**

**plt.clf()**

**plt.figure(figsize=(12,8))**

**params = {'axes.titlesize':'18',**

**'xtick.labelsize':'14',**

**'ytick.labelsize':'14'}**

**matplotlib.rcParams.update(params)**

**plt.title('Distribution of attacks')**

**df.plot(kind='barh')**

**df['label'].value\_counts().apply(np.log).plot(kind='barh')**

**plt.show()**

**2.KDD skewness and kurtosis**

**df.skew()**

**df.kurtosis()**

**3.Univariate histograms**

**import matplotlib.pyplot as plt**

**import matplotlib**

**params = {'axes.titlesize':'28',**

**'xtick.labelsize':'24',**

**'ytick.labelsize':'24'}**

**matplotlib.rcParams.update(params)**

**df.hist(figsize=(50, 30), bins=20)**

**plt.show()**

**#Build the model with classification algorithms**

**x = dataset.iloc[:, :-1].values**

**y = dataset.iloc[:, 13].values**

**x.shape**

**y.shape**

**from sklearn.model\_selection import train\_test\_split**

**x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.3, random\_state = 0)**

**1. naive\_bayes**

**from sklearn.naive\_bayes import GaussianNB**

**classifier = GaussianNB()**

**classifier.fit(x\_train, y\_train)**

**#Predicting the Test set results**

**y\_pred = classifier.predict(x\_test)**

**from sklearn import metrics**

**#Model Accuracy, how often is the classifier correct**

**print("Accuracy nb:",metrics.accuracy\_score(y\_test, y\_pred))**

**2.** **GradientBoostingClassifier**

**from sklearn.ensemble import GradientBoostingClassifier**

**from sklearn.metrics import accuracy\_score**

**gbt = GradientBoostingClassifier()**

**gbt1=gbt.fit(x\_train,y\_train)**

**predictions = gbt1.predict(x\_test)**

**print("accuracy for gbt:",accuracy\_score(y\_test, predictions)\*100)**

**3. SGDClassifier**

**from sklearn.linear\_model import SGDClassifier**

**sgb = SGDClassifier(loss="hinge", penalty="l1")**

**sgb1=sgb.fit(x\_train, y\_train)**

**predictions = sgb1.predict(x\_test)**

**print("accuracy for SGD:",accuracy\_score(y\_test, predictions)\*100)**

**4. AdaBoostClassifier**

**from sklearn.ensemble import AdaBoostClassifier**

**abt = AdaBoostClassifier(n\_estimators=100)**

**abt1=abt.fit(x\_train, y\_train)**

**predictions = abt1.predict(x\_test)**

**print("accuracy:",accuracy\_score(y\_test, predictions)\*100)**

**5. DecisionTreeClassifier**

**from sklearn.tree import DecisionTreeClassifier**

**clf=DecisionTreeClassifier(max\_leaf\_nodes=15,criterion='gini')**

**clf1=clf.fit(x\_train,y\_train)**

**predictions = clf1.predict(x\_test)**

**print("accuracy for decision tree:",accuracy\_score(y\_test, predictions)\*100)**

**6. RandomForestClassifier**

**from sklearn.ensemble import RandomForestClassifier**

**clf2 = RandomForestClassifier(n\_estimators=1000,max\_leaf\_nodes=15)**

**#abc6=AdaBoostClassifier(n\_estimators=100,base\_estimator=clf2,learning\_rate=0.01)**

**clf5=clf2.fit(x\_train,y\_train)**

**predictions = clf5.predict(x\_test)**

**print("accuracy for RFC:",accuracy\_score(y\_test, predictions)\*100)**

**7. RandomForestClassifier**

**from sklearn.ensemble import RandomForestClassifier**

**clf2 = RandomForestClassifier(n\_estimators=1000,max\_leaf\_nodes=15)**

**clf5=clf2.fit(x\_train,y\_train)**

**predictions = clf5.predict(x\_test)**

**print("accuracy for RFC:",accuracy\_score(y\_test, predictions)\*100)**

**8. MLPClassifier**

**from sklearn.neural\_network import MLPClassifier**

**mlp = MLPClassifier(hidden\_layer\_sizes=(100, 100, 12), alpha=1e-4, solver='sgd', random\_state=42, learning\_rate\_init=.1)**

**model\_3 = mlp.fit(x\_train, y\_train)**

**y\_pred = model\_3.predict(x\_test)**

**acc = metrics.accuracy\_score(y\_test, y\_pred)**

**print("This is Multi Layer Perceptron classifier \n\n")**

**print("Accuracy: {: .4f} %".format(acc\*100))**

**#classification report**

**from sklearn.metrics import confusion\_matrix**

**confusion\_matrix(y\_test, y\_pred)**

**from sklearn.model\_selection import cross\_val\_score**

**accuracies = cross\_val\_score(estimator = classifier, X = x\_train, y = y\_train, cv = 5)**

**accuracies.mean()**

**accuracies.std()**

**from sklearn import metrics**

**Model Accuracy, how often is the classifier correct**

**print(metrics.classification\_report(y\_test, y\_pred))**

**#Hyperparameter tuning - GridSearchCV using SVM**

**from sklearn.svm import SVC**

**classifier = SVC(kernel = 'linear', random\_state = 0)**

**classifier.fit(X\_train, y\_train)**

**Predicting the Test set results**

**y\_pred = classifier.predict(X\_test)**

**from sklearn import metrics**

**#Model Accuracy, how often is the classifier correct**

**print(metrics.classification\_report(y\_test, y\_pred))**

**Making the Confusion Matrix**

**from sklearn.metrics import confusion\_matrix**

**confusion\_matrix(y\_test, y\_pred)**

**from sklearn.model\_selection import GridSearchCV**

**parameters = [{'C': [1, 2], 'kernel': ['linear']},**

**{'C': [1, 2], 'kernel': ['rbf'], 'gamma': [0.1, 0.2]}]**

**grid\_search = GridSearchCV(estimator = classifier,**

**param\_grid = parameters,**

**scoring = 'accuracy',**

**cv = 2,**

**n\_jobs = -1)**

**grid\_search = grid\_search.fit(X\_train, y\_train)**

**accuracy = grid\_search.best\_score\_**

**accuracy\*100**

**#ML PIPLELINE**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.preprocessing import StandardScaler**

**from sklearn.decomposition import PCA**

**from sklearn.pipeline import Pipeline**

**from sklearn.linear\_model import LogisticRegression**

**from sklearn.tree import DecisionTreeClassifier**

**from sklearn.ensemble import RandomForestClassifier**

**from sklearn.model\_selection import train\_test\_split**

**x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.3, random\_state = 0)**

**pipeline\_lr=Pipeline([('scalar1',StandardScaler()),**

**('pca1',PCA(n\_components=2)),**

**('lr\_classifier',LogisticRegression(random\_state=0))])**

**pipeline\_dt=Pipeline([('scalar2',StandardScaler()),**

**('pca2',PCA(n\_components=2)),**

**('dt\_classifier',DecisionTreeClassifier())])**

**pipeline\_randomforest=Pipeline([('scalar3',StandardScaler()),**

**('pca3',PCA(n\_components=2)),**

**('rf\_classifier',RandomForestClassifier())])**

**## LEts make the list of pipelines**

**pipelines = [pipeline\_lr, pipeline\_dt, pipeline\_randomforest]**

**best\_accuracy=0.0**

**best\_classifier=0**

**best\_pipeline=""**

**# Dictionary of pipelines and classifier types for ease of reference**

**pipe\_dict = {0: 'Logistic Regression', 1: 'Decision Tree', 2: 'RandomForest'}**

**# Fit the pipelines**

**for pipe in pipelines:**

**pipe.fit(x\_train, y\_train)**

**for i,model in enumerate(pipelines):**

**print("{} Test Accuracy: {}".format(pipe\_dict[i],model.score(x\_test,y\_test)))**

**for i,model in enumerate(pipelines):**

**if model.score(x\_test,y\_test)>best\_accuracy:**

**best\_accuracy=model.score(x\_test,y\_test)**

**best\_pipeline=model**

**best\_classifier=i**

**print('Classifier with best accuracy:{}'.format(pipe\_dict[best\_classifier]))**

**"""Pipelines Perform Hyperparameter Tuning Using Grid SearchCV"""**

**from sklearn.model\_selection import GridSearchCV**

**# Create a pipeline**

**pipe = Pipeline([("classifier", RandomForestClassifier())])**

**# Create dictionary with candidate learning algorithms and their hyperparameters**

**grid\_param = [**

**{"classifier": [LogisticRegression()],**

**"classifier\_\_penalty": ['l2','l1'],**

**"classifier\_\_C": np.logspace(0, 4, 6)**

**},**

**{"classifier": [LogisticRegression()],**

**"classifier\_\_penalty": ['l2'],**

**"classifier\_\_C": np.logspace(0, 4, 6),**

**"classifier\_\_solver":['newton-cg','saga','sag','liblinear'] ##This solvers don't allow L1 penalty**

**},**

**{"classifier": [RandomForestClassifier()],**

**"classifier\_\_n\_estimators": [10, 15],**

**"classifier\_\_max\_depth":[5,8,15,25,30,None],**

**"classifier\_\_min\_samples\_leaf":[1,2,5,10,15,30],**

**"classifier\_\_max\_leaf\_nodes": [2, 5,10]}]**

**# create a gridsearch of the pipeline, the fit the best model**

**gridsearch = GridSearchCV(pipe, grid\_param, cv=3, verbose=0,n\_jobs=-1) # Fit grid search**

**best\_model = gridsearch.fit(x\_train,y\_train)**

**print(best\_model.best\_estimator\_)**

**print("The mean accuracy of the model is:",best\_model.score(x\_test,y\_test)\*100)**