# Importing Libraries

import matplotlib

import matplotlib.pyplot as plt

import pandas as pd  #Data manipulation and analysis

import numpy as np   #Performs high level manipulation

import sklearn   # provides efficient tools for predictive data analysis

# Preprocessing purpose

from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import LabelEncoder

# For getting the importances

from sklearn.ensemble import RandomForestClassifier

# Feature Extraction

from sklearn.decomposition import PCA

# Splitting Data

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

# For accuracy,Classification Report, Confusion Matrix

from sklearn import metrics

# For training different ML models

from sklearn import tree

from sklearn.linear\_model import LogisticRegression

from sklearn.naive\_bayes import BernoulliNB

from sklearn.linear\_model import LinearRegression

from sklearn.svm import SVC

from sklearn.ensemble import GradientBoostingClassifier

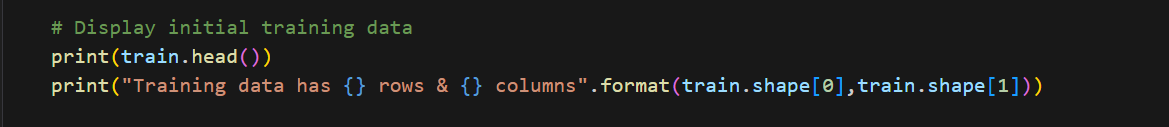
# Ignore warnings

import warnings

warnings.filterwarnings('ignore')

The code is importing necessary libraries and modules for performing machine learning tasks on a dataset. Here is a detailed explanation of each line:

1. **import matplotlib**: This is a library used for data visualization in Python.
2. **import matplotlib.pyplot as plt**: This is a module of the matplotlib library used for creating plots and charts.
3. **import pandas as pd**: This library provides data manipulation and analysis tools for Python.
4. **import numpy as np**: This library provides a collection of functions and tools for performing high-level mathematical operations on arrays and matrices..
5. **import sklearn**: This library provides efficient tools for predictive data analysis, such as machine learning algorithms, data preprocessing tools, and model selection and evaluation tools.
6. **from sklearn.preprocessing import StandardScaler**: This module provides a tool for standardizing the scale of features in a dataset by removing the mean and scaling to unit variance.
7. **from sklearn.preprocessing import LabelEncoder**: This module provides a tool for encoding categorical variables in a dataset into numerical values.
8. **from sklearn.ensemble import RandomForestClassifier**: This module provides an implementation of the random forest algorithm for classification tasks.
9. **from sklearn.decomposition import PCA**: This module provides a tool for feature extraction by reducing the dimensionality of a dataset while preserving the most important information.
10. **from sklearn.model\_selection import train\_test\_split**: This module provides a tool for splitting a dataset into training and testing sets for model training and evaluation.
11. **from sklearn import metrics**: This module provides tools for computing metrics to evaluate the performance of machine learning models, such as accuracy, classification report, and confusion matrix.
12. **from sklearn import tree**: This module provides an implementation of the decision tree algorithm for classification and regression tasks.
13. **from sklearn.linear\_model import LogisticRegression**: This module provides an implementation of the logistic regression algorithm for classification tasks.
14. **from sklearn.naive\_bayes import BernoulliNB**: This module provides an implementation of the Naive Bayes algorithm for binary classification tasks.
15. **from sklearn.linear\_model import LinearRegression**: This module provides an implementation of the linear regression algorithm for regression tasks.
16. **from sklearn.svm import SVC**: This module provides an implementation of the support vector machine algorithm for classification and regression tasks.
17. **from sklearn.ensemble import GradientBoostingClassifier**: This module provides an implementation of the gradient boosting algorithm for classification tasks.
18. **import warnings**: This is a Python module used for suppressing warnings.
19. **warnings.filterwarnings('ignore')**: This line is used to suppress all warnings in the code to avoid cluttering the output.



This code is used to display the first few rows of the training dataset and the dimensions of the dataset. Here is a detailed explanation of each line:

1. **print(train.head())**: This line prints the first few rows of the 'train' DataFrame using the 'head()' method. By default, this method prints the first 5 rows of the DataFrame.
2. **print("Training data has {} rows & {} columns".format(train.shape[0],train.shape[1]))**: This line prints the dimensions of the 'train' DataFrame using the 'shape' attribute. The 'shape' attribute returns a tuple containing the number of rows and columns in the DataFrame, which are accessed using the indices [0] and [1], respectively. The 'format()' method is used to insert these values into the string that is being printed.

#Display initial Testing Data

print(test.head())

print("Testing data has {} rows & {} columns".format(test.shape[0],test.shape[1]))

This code is used to display the first few rows of the testing dataset and the dimensions of the dataset. Here is a detailed explanation of each line:

1. **print(test.head())**: This line prints the first few rows of the 'test' DataFrame using the 'head()' method. By default, this method prints the first 5 rows of the DataFrame.
2. **print("Testing data has {} rows & {} columns".format(test.shape[0],test.shape[1]))**: This line prints the dimensions of the 'test' DataFrame using the 'shape' attribute. The 'shape' attribute returns a tuple containing the number of rows and columns in the DataFrame, which are accessed using the indices [0] and [1], respectively. The 'format()' method is used to insert these values into the string that is being printed.

train.describe()

The **train.describe()** method generates descriptive statistics that summarize the central tendency, dispersion, and shape of the numerical features in the training dataset. Here is a detailed explanation of this code:

1. **train.describe()**: This line calls the **describe()** method on the **train** DataFrame to generate the summary statistics. The **describe()** method computes the count, mean, standard deviation, minimum, 25th percentile, median, 75th percentile, and maximum values for each numerical feature in the dataset.

The output of this code will be a table of summary statistics for each numerical feature in the training dataset.

test.describe()

The **test.describe()** method generates descriptive statistics that summarize the central tendency, dispersion, and shape of the numerical features in the testing dataset. Here is a detailed explanation of this code:

1. **test.describe()**: This line calls the **describe()** method on the **test** DataFrame to generate the summary statistics. The **describe()** method computes the count, mean, standard deviation, minimum, 25th percentile, median, 75th percentile, and maximum values for each numerical feature in the dataset.

The output of this code will be a table of summary statistics for each numerical feature in the testing dataset.

print('Total attack categories and count in training data')

print()

print(train['attack\_cat'].value\_counts())

Total attack categories and count in training data

Normal 37000

Generic 18871

Exploits 11132

Fuzzers 6062

DoS 4089

Reconnaissance 3496

Analysis 677

Backdoor 583

Shellcode 378

Worms 44

Name: attack\_cat, dtype: int64

This code is used to print the total number of attack categories and their count in the training dataset. Here is a detailed explanation of each line:

1. **print('Total attack categories and count in training data')**: This line prints the string "Total attack categories and count in training data" to the console, indicating the purpose of the following output.
2. **print()**: This line prints a blank line to create a separation between the previous line and the next line of output.
3. **print(train['attack\_cat'].value\_counts())**: This line prints the count of each unique value in the 'attack\_cat' column of the 'train' DataFrame using the 'value\_counts()' method. The 'value\_counts()' method counts the occurrence of each unique value in the specified column and returns a Series object with the counts sorted in descending order.

The output of this code will be a list of the attack categories and their respective counts in the training dataset.

print('Total attack categories and count in testing data')

print()

print(test['attack\_cat'].value\_counts())

Total attack categories and count in testing data

Normal 56000

Generic 40000

Exploits 33393

Fuzzers 18184

DoS 12264

Reconnaissance 10491

Analysis 2000

Backdoor 1746

Shellcode 1133

Worms 130

Name: attack\_cat, dtype: int64

This code is used to print the total number of attack categories and their count in the testing dataset. Here is a detailed explanation of each line:

1. **print('Total attack categories and count in testing data')**: This line prints the string "Total attack categories and count in testing data" to the console, indicating the purpose of the following output.
2. **print()**: This line prints a blank line to create a separation between the previous line and the next line of output.
3. **print(test['attack\_cat'].value\_counts())**: This line prints the count of each unique value in the 'attack\_cat' column of the 'test' DataFrame using the 'value\_counts()' method. The 'value\_counts()' method counts the occurrence of each unique value in the specified column and returns a Series object with the counts sorted in descending order.

The output of this code will be a list of the attack categories and their respective counts in the testing dataset.

Top of Form

#id column is of no use so drop it

train.drop(['id'],axis=1,inplace=True)

test.drop(['id'],axis=1,inplace=True)

print(list(train.columns))

print()

print(list(test.columns))

This code is used to drop the 'id' column from both the training and testing datasets, and print the remaining columns in each dataset. Here is a detailed explanation of each line:

1. **train.drop(['id'],axis=1,inplace=True)**: This line drops the 'id' column from the 'train' DataFrame using the 'drop()' method with the 'axis=1' parameter. The 'inplace=True' parameter specifies that the DataFrame should be modified in place, rather than returning a new DataFrame with the 'id' column removed.
2. **test.drop(['id'],axis=1,inplace=True)**: This line drops the 'id' column from the 'test' DataFrame using the 'drop()' method with the 'axis=1' parameter. The 'inplace=True' parameter specifies that the DataFrame should be modified in place, rather than returning a new DataFrame with the 'id' column removed.
3. **print(list(train.columns))**: This line prints a list of the column names in the 'train' DataFrame using the 'columns' attribute.
4. **print()**: This line prints a blank line to create a separation between the previous line and the next line of output.
5. **print(list(test.columns))**: This line prints a list of the column names in the 'test' DataFrame using the 'columns' attribute.

The output of this code will be a list of the remaining columns in each dataset, with the 'id' column removed.

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

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

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

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

sc\_test = scaler.fit\_transform(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)

This code is used to scale the numerical attributes of both the training and testing datasets using the StandardScaler() method from scikit-learn's preprocessing module. Here is a detailed explanation of each line:

1. **from sklearn.preprocessing import StandardScaler**: This line imports the StandardScaler class from the sklearn.preprocessing module. The StandardScaler class is used to scale data to have zero mean and unit variance.
2. **scaler = StandardScaler()**: This line creates a StandardScaler object called 'scaler' which will be used to scale the numerical data.
3. **cols = train.select\_dtypes(include=['float64','int64']).columns**: This line selects the column names of numerical data types in the 'train' DataFrame using the 'select\_dtypes()' method with the 'include' parameter set to a list of float and integer data types. The 'columns' attribute is then used to extract the column names.
4. **sc\_train = scaler.fit\_transform(train.select\_dtypes(include=['float64','int64']))**: This line applies the fit\_transform() method of the 'scaler' object to the numerical data in the 'train' DataFrame, which scales the data to have zero mean and unit variance. The 'fit\_transform()' method is used to first fit the scaling parameters on the data and then transform the data based on those parameters. The 'select\_dtypes()' method is used to select only the columns with numerical data types.
5. **sc\_test = scaler.fit\_transform(test.select\_dtypes(include=['float64','int64']))**: This line applies the fit\_transform() method of the 'scaler' object to the numerical data in the 'test' DataFrame, which scales the data to have zero mean and unit variance. The 'fit\_transform()' method is used to first fit the scaling parameters on the data and then transform the data based on those parameters. The 'select\_dtypes()' method is used to select only the columns with numerical data types.
6. **sc\_traindf = pd.DataFrame(sc\_train, columns = cols)**: This line converts the scaled numerical data of the 'train' DataFrame to a new DataFrame called 'sc\_traindf' using the 'pd.DataFrame()' constructor. The 'sc\_train' array contains the scaled numerical data, and the 'cols' list contains the names of the columns. These are passed to the 'pd.DataFrame()' constructor using the 'data' and 'columns' parameters, respectively.
7. **sc\_testdf = pd.DataFrame(sc\_test, columns = cols)**: This line converts the scaled numerical data of the 'test' DataFrame to a new DataFrame called 'sc\_testdf' using the 'pd.DataFrame()' constructor. The 'sc\_test' array contains the scaled numerical data, and the 'cols' list contains the names of the columns. These are passed to the 'pd.DataFrame()' constructor using the 'data' and 'columns' parameters, respectively.

The output of this code will be two new DataFrames called 'sc\_traindf' and 'sc\_testdf', which contain the scaled numerical data of the original 'train' and 'test' DataFrames.

sc\_traindf.describe()

This code will display the summary statistics of the **sc\_traindf** dataframe, which contains the standardized numerical attributes of the training data. The summary statistics include count, mean, standard deviation, minimum, 25th percentile, 50th percentile (median), 75th percentile, and maximum for each column in the dataframe.

By looking at the summary statistics, we can get an idea of the distribution of the data and whether there are any outliers or extreme values. We can also check if the scaling process has worked correctly, as the mean should be approximately 0 and the standard deviation should be approximately 1 for each column.

sc\_testdf.describe()

This code will display the summary statistics of the **sc\_testdf** dataframe, which contains the standardized numerical attributes of the testing data. The summary statistics include count, mean, standard deviation, minimum, 25th percentile, 50th percentile (median), 75th percentile, and maximum for each column in the dataframe.

By looking at the summary statistics, we can get an idea of the distribution of the data and whether there are any outliers or extreme values. We can also check if the scaling process has worked correctly, as the mean should be approximately 0 and the standard deviation should be approximately 1 for each column.

LE = LabelEncoder()

# extract categorical attributes from both training and test sets

obj\_train = train.select\_dtypes(include=['object']).copy()

obj\_test = test.select\_dtypes(include=['object']).copy()

#print(obj\_train)

#print(obj\_pred)

# encode the categorical attributes

LE\_obj\_train = obj\_train.apply(LE.fit\_transform)

LE\_obj\_test = obj\_test.apply(LE.fit\_transform)

# separate target column from encoded data

enctrain = LE\_obj\_train.drop(['attack\_cat'], axis=1)

#print(enctrain)

enctest = LE\_obj\_test.drop(['attack\_cat'],axis=1)

#print(encpred)

test\_target = test['attack\_cat']

#lir\_tar\_train = LE\_obj\_train['attack\_cat']

This code performs label encoding on the categorical attributes of the training and testing data.

First, the **LabelEncoder()** function is imported from **sklearn.preprocessing**.

Next, the categorical attributes of the training and testing data are extracted using the **select\_dtypes** function with **include=['object']**. This returns a dataframe containing only the columns with object datatype.

Then, the **fit\_transform()** method of **LabelEncoder()** is applied to the categorical attributes of the training and testing data separately to encode the categorical values as integers. The **fit\_transform()** method learns the mapping of each unique value to a corresponding integer and applies this mapping to the original data.

Finally, the encoded data is stored in new dataframes **LE\_obj\_train** and **LE\_obj\_test**, and the **attack\_cat** column is dropped from these dataframes as it will be used as the target variable during training. The target variable of the testing data is stored separately in **test\_target**.

Top of Form

train\_x = pd.concat([sc\_traindf,enctrain],axis=1)

train\_y = train['attack\_cat']

train\_x.shape

(82332, 43)

This code combines the scaled numerical attributes and the encoded categorical attributes of the training data into a single dataframe, **train\_x**, which will be used as the input for the machine learning models. The target variable of the training data is stored separately in **train\_y**.

First, the **concat()** function from Pandas is used to concatenate the scaled numerical attributes in **sc\_traindf** and the encoded categorical attributes in **enctrain** along the columns axis (i.e., **axis=1**). This creates a new dataframe **train\_x** containing both the scaled numerical attributes and the encoded categorical attributes.

The target variable of the training data, **attack\_cat**, is stored separately in **train\_y**.

Finally, the shape of **train\_x** is printed to confirm the number of rows and columns.

test\_df = pd.concat([sc\_testdf,enctest],axis=1)

test\_df.shape

(175341, 43)

This code is similar to the previous code block. Here, the scaled numerical attributes and the encoded categorical attributes of the test data are concatenated into a single dataframe **test\_df**, which will be used as the input for prediction using the machine learning models.

First, the **concat()** function from Pandas is used to concatenate the scaled numerical attributes in **sc\_testdf** and the encoded categorical attributes in **enctest** along the columns axis (i.e., **axis=1**). This creates a new dataframe **test\_df** containing both the scaled numerical attributes and the encoded categorical attributes.

Finally, the shape of **test\_df** is printed to confirm the number of rows and columns.

rfc = RandomForestClassifier();

# fit random forest classifier on the training set

rfc.fit(train\_x, train\_y);

# extract important features

score = np.round(rfc.feature\_importances\_,3)

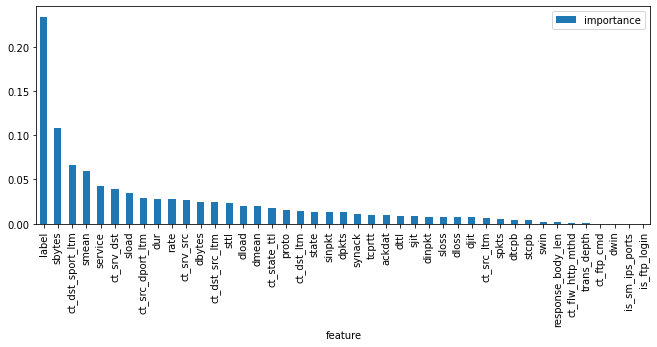
importances = pd.DataFrame({'feature':train\_x.columns,'importance':score})

importances = importances.sort\_values('importance',ascending=False).set\_index('feature')

# plot importances

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

importances.plot.bar();



In this code block, a **RandomForestClassifier** is created with default parameters, and it is fitted on the training set. The feature importances are then extracted from the classifier, and a bar plot is created to show the importance of each feature in descending order.

The **RandomForestClassifier** is an ensemble classifier that fits a number of decision tree classifiers on various sub-samples of the dataset and averages their predictions to improve the accuracy and control over-fitting. The **feature\_importances\_** attribute of the classifier returns an array of feature importances, which is then used to create a Pandas DataFrame with features as the index and their importance scores as the values. Finally, the DataFrame is sorted in descending order of importance, and a horizontal bar plot is created to visualize the relative importance of each feature.

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=15)

rfe = rfe.fit(train\_x, train\_y)

# summarize the selection of the attributes

feature\_map = [(i, v) for i, v in itertools.zip\_longest(rfe.get\_support(), train\_x.columns)]

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

selected\_train = train\_x.loc[:, selected\_features]

#print()

selected\_test = test\_df.loc[:, selected\_features]

selected\_features

['dur', 'sbytes', 'dbytes', 'rate', 'sttl', 'sload', 'dload', 'synack', 'smean', 'dmean', 'ct\_srv\_src', 'ct\_dst\_sport\_ltm', 'ct\_srv\_dst', 'label', 'service']

The code above performs feature selection using the Recursive Feature Elimination (RFE) method. RFE selects the top **n\_features\_to\_select** features by recursively considering smaller and smaller sets of features, and ranking their importance using a specified machine learning model. In this case, we use a random forest classifier (**rfc**) as the model. The selected features are stored in **selected\_features**. We then extract these features from both the training and test datasets, and store them in **selected\_train** and **selected\_test**, respectively. This step is important for reducing the dimensionality of the data and improving the accuracy of the machine learning model.

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(selected\_train,train\_y,train\_size=0.70, random\_state=2)

Good choice, using **train\_test\_split()** function from scikit-learn package to split the data into training and validation sets. This function splits the data randomly into training and validation sets based on the size of the validation set specified.

Here, you have specified that 70% of the data should be used for training the model, and the remaining 30% should be used for validation. You have also set the random state to 2, which ensures that the random split is reproducible.

Now, you can use these splits to train and evaluate your model.

#Training different Machine Learning models for comapritive analysis

DTC\_Classifier = tree.DecisionTreeClassifier(criterion='entropy', random\_state=0) #Decision Tree Classifier

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

BNB\_Classifier = BernoulliNB() #Naive Bayes Algorithm

#LIR\_Classifier = LinearRegression() #Multi Linear Regression

gradient\_booster = GradientBoostingClassifier(learning\_rate=0.1)

SVM\_Classifier = SVC(kernel = 'poly',C = 75) #Support Vector Machine

The code above is importing various machine learning models from the scikit-learn library. These models will be used for comparative analysis to determine which model performs best for the given dataset. The models imported are:

1. DecisionTreeClassifier - a model that uses a decision tree to make predictions by learning simple decision rules from the data features.
2. LogisticRegression - a model that uses logistic function to estimate the probability of a binary response based on one or more predictor (or independent) variables.
3. BernoulliNB - a model that is based on the Bayes theorem and assumes that all the features are binary-valued (i.e., take only two values 0 or 1).
4. GradientBoostingClassifier - a model that combines multiple decision trees to improve predictive accuracy. Each tree is built sequentially with each new tree learning from the mistakes of the previous tree.
5. Support Vector Machine - a model that constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks. In this case, a polynomial kernel is used.

These models will be trained on the selected training data and used to predict the target variable for the testing data. The model with the best performance will be selected based on evaluation metrics such as accuracy, precision, recall, and F1 score.

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

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

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

#LIR\_Classifier.fit(mlrx\_train, mlry\_train)

gradient\_booster.fit(X\_train,Y\_train)

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

SVC

SVC(C=75, kernel='poly')

In the above code, we are fitting different machine learning models on the training data to train them for making predictions.

* **DTC\_Classifier** is a Decision Tree Classifier, which we initialize with a criterion of 'entropy' and a random state of 0. We then fit it on the training data using the **fit** method.
* **LGR\_Classifier** is a Logistic Regression model, which we initialize with a maximum number of iterations of -1 and a random state of 0. We then fit it on the training data using the **fit** method.
* **BNB\_Classifier** is a Bernoulli Naive Bayes algorithm, which we initialize and then fit it on the training data using the **fit** method.
* **gradient\_booster** is a Gradient Boosting Classifier, which we initialize with a learning rate of 0.1. We then fit it on the training data using the **fit** method.
* **SVM\_Classifier** is a Support Vector Machine classifier, which we initialize with a kernel of 'poly' and a C value of 75. We then fit it on the training data using the **fit** method.

models = []

models.append(('Decision Tree Classifier', DTC\_Classifier))

models.append(('Logistic Regression', LGR\_Classifier))

models.append(('Naive Baye Classifier', BNB\_Classifier))

models.append(('Gradient Booster Classifier', gradient\_booster))

models.append(('Support Vector Machine', SVM\_Classifier))

This code block is creating a list of models to train and compare for the classification problem. It is appending the models with their names as tuples in a list called **models**.

Each tuple has two elements, the name of the model as a string and the trained model object.

* **DTC\_Classifier** is a decision tree classifier object created earlier
* **LGR\_Classifier** is a logistic regression classifier object created earlier
* **BNB\_Classifier** is a Bernoulli naive bayes classifier object created earlier
* **gradient\_booster** is a gradient boosting classifier object created earlier
* **SVM\_Classifier** is a support vector machine classifier object created earlier

These models will be trained on the selected features and compared based on their performance using evaluation metrics.

for i, v in models:

    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(i))

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

    print()

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

    print()

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

    print()

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

Model Accuracy:

0.9386799000555247

Confusion matrix:

[[ 62 2 86 215 107 0 0 0 0 0]

[ 17 53 25 216 98 0 0 0 0 0]

[ 12 9 2200 504 103 0 0 2 0 0]

[ 19 14 1057 6559 168 1 0 0 0 0]

[ 18 12 164 415 3621 0 0 2 0 0]

[ 0 0 21 9 0 13181 0 0 0 0]

[ 0 0 0 0 0 0 25902 0 0 0]

[ 0 1 155 81 0 0 0 2224 0 0]

[ 0 0 0 0 0 0 0 0 268 0]

[ 0 0 1 0 0 0 0 0 0 28]]

Classification report:

precision recall f1-score support

Analysis 0.48 0.13 0.21 472

Backdoor 0.58 0.13 0.21 409

DoS 0.59 0.78 0.67 2830

Exploits 0.82 0.84 0.83 7818

Fuzzers 0.88 0.86 0.87 4232

Generic 1.00 1.00 1.00 13211

Normal 1.00 1.00 1.00 25902

Reconnaissance 1.00 0.90 0.95 2461

Shellcode 1.00 1.00 1.00 268

Worms 1.00 0.97 0.98 29

accuracy 0.94 57632

macro avg 0.84 0.76 0.77 57632

weighted avg 0.94 0.94 0.94 57632

============================== Logistic Regression Model Evaluation ==============================

Model Accuracy:

0.8552193225985564

Confusion matrix:

[[ 2 0 35 151 126 145 0 13 0 0]

[ 1 0 16 83 130 141 0 38 0 0]

[ 2 0 363 1672 236 265 0 292 0 0]

[ 4 0 340 5660 738 574 0 502 0 0]

[ 2 0 64 547 2924 404 0 291 0 0]

[ 0 0 3 361 67 12713 0 67 0 0]

[ 0 0 0 0 0 0 25902 0 0 0]

[ 0 0 58 323 225 131 0 1724 0 0]

[ 0 0 2 53 35 0 0 178 0 0]

[ 0 0 0 19 3 6 0 1 0 0]]

Classification report:

precision recall f1-score support

Analysis 0.18 0.00 0.01 472

Backdoor 0.00 0.00 0.00 409

DoS 0.41 0.13 0.20 2830

Exploits 0.64 0.72 0.68 7818

Fuzzers 0.65 0.69 0.67 4232

Generic 0.88 0.96 0.92 13211

Normal 1.00 1.00 1.00 25902

Reconnaissance 0.56 0.70 0.62 2461

Shellcode 0.00 0.00 0.00 268

Worms 0.00 0.00 0.00 29

accuracy 0.86 57632

macro avg 0.43 0.42 0.41 57632

weighted avg 0.83 0.86 0.84 57632

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

Model Accuracy:

0.8332176568573015

Confusion matrix:

[[ 81 1 132 41 5 203 0 9 0 0]

[ 80 2 64 20 22 197 0 24 0 0]

[ 78 2 1422 725 147 269 0 187 0 0]

[ 165 3 1354 4887 649 465 0 295 0 0]

[ 210 3 549 379 2458 428 0 205 0 0]

[ 2 0 73 337 81 12691 0 27 0 0]

[ 0 0 0 0 0 0 25902 0 0 0]

[ 18 0 814 355 675 22 0 577 0 0]

[ 1 0 101 0 104 0 0 62 0 0]

[ 0 0 4 23 2 0 0 0 0 0]]

Classification report:

precision recall f1-score support

Analysis 0.13 0.17 0.15 472

Backdoor 0.18 0.00 0.01 409

DoS 0.32 0.50 0.39 2830

Exploits 0.72 0.63 0.67 7818

Fuzzers 0.59 0.58 0.59 4232

Generic 0.89 0.96 0.92 13211

Normal 1.00 1.00 1.00 25902

Reconnaissance 0.42 0.23 0.30 2461

Shellcode 0.00 0.00 0.00 268

Worms 0.00 0.00 0.00 29

accuracy 0.83 57632

macro avg 0.42 0.41 0.40 57632

weighted avg 0.83 0.83 0.83 57632

============================== Gradient Booster Classifier Model Evaluation ==============================

Model Accuracy:

0.8908418933925597

Confusion matrix:

[[ 37 0 116 299 19 0 0 1 0 0]

[ 0 20 54 304 25 0 0 2 4 0]

[ 1 0 1414 1226 104 15 3 40 26 1]

[ 1 1 1234 6094 274 40 10 138 24 2]

[ 0 0 260 709 3158 11 62 12 4 16]

[ 0 0 25 356 54 12770 1 1 4 0]

[ 0 0 1 73 13 145 25646 0 10 14]

[ 0 0 165 230 40 2 0 2021 3 0]

[ 0 0 3 44 30 3 8 23 157 0]

[ 0 0 0 3 0 0 2 0 0 24]]

Classification report:

precision recall f1-score support

Analysis 0.95 0.08 0.14 472

Backdoor 0.95 0.05 0.09 409

DoS 0.43 0.50 0.46 2830

Exploits 0.65 0.78 0.71 7818

Fuzzers 0.85 0.75 0.79 4232

Generic 0.98 0.97 0.97 13211

Normal 1.00 0.99 0.99 25902

Reconnaissance 0.90 0.82 0.86 2461

Shellcode 0.68 0.59 0.63 268

Worms 0.42 0.83 0.56 29

accuracy 0.89 57632

macro avg 0.78 0.63 0.62 57632

weighted avg 0.90 0.89 0.89 57632

============================== Support Vector Machine Model Evaluation ==============================

Model Accuracy:

0.8832766518600778

Confusion matrix:

[[ 10 0 42 366 52 0 0 2 0 0]

[ 0 0 17 294 70 0 0 28 0 0]

[ 1 0 504 1974 152 28 0 171 0 0]

[ 2 0 357 6650 440 27 0 341 1 0]

[ 0 0 88 835 3156 3 0 150 0 0]

[ 0 0 25 282 53 12821 0 29 0 1]

[ 0 0 0 0 0 0 25902 0 0 0]

[ 0 0 71 375 159 4 0 1852 0 0]

[ 0 0 15 63 22 3 0 158 7 0]

[ 0 0 1 23 1 0 0 1 0 3]]

Classification report:

precision recall f1-score support

Analysis 0.77 0.02 0.04 472

Backdoor 0.00 0.00 0.00 409

DoS 0.45 0.18 0.26 2830

Exploits 0.61 0.85 0.71 7818

Fuzzers 0.77 0.75 0.76 4232

Generic 0.99 0.97 0.98 13211

Normal 1.00 1.00 1.00 25902

Reconnaissance 0.68 0.75 0.71 2461

Shellcode 0.88 0.03 0.05 268

Worms 0.75 0.10 0.18 29

accuracy 0.88 57632

macro avg 0.69 0.46 0.47 57632

weighted avg 0.88 0.88 0.87 57632

The for loop is used to iterate through the list of models that were created earlier. For each model, it calculates its accuracy, confusion matrix, and classification report using the **metrics** module of the **sklearn** library.

* **accuracy = metrics.accuracy\_score(Y\_train, v.predict(X\_train))** calculates the accuracy of the model on the training set.
* **confusion\_matrix = metrics.confusion\_matrix(Y\_train, v.predict(X\_train))** calculates the confusion matrix of the model on the training set.
* **classification = metrics.classification\_report(Y\_train, v.predict(X\_train))** calculates the classification report of the model on the training set.

The results are then printed out in a formatted manner for each model, displaying the model name, accuracy, confusion matrix, and classification report.

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(i))

    print()

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

    print()

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

    print()

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

    print()

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

Model Accuracy:

0.8859919028340081

Confusion matrix:

[[ 17 8 44 100 36 0 0 0 0 0]

[ 4 4 15 97 50 0 0 2 2 0]

[ 10 4 676 471 61 19 0 9 8 1]

[ 27 12 717 2247 165 57 0 67 19 3]

[ 24 15 93 273 1406 6 0 7 6 0]

[ 0 3 26 63 17 5544 0 3 4 0]

[ 0 0 0 0 0 0 11098 0 0 0]

[ 0 2 79 106 8 0 0 833 7 0]

[ 0 1 13 18 13 8 0 2 55 0]

[ 0 0 4 3 2 2 0 0 0 4]]

Classification report:

precision recall f1-score support

Analysis 0.21 0.08 0.12 205

Backdoor 0.08 0.02 0.04 174

DoS 0.41 0.54 0.46 1259

Exploits 0.67 0.68 0.67 3314

Fuzzers 0.80 0.77 0.78 1830

Generic 0.98 0.98 0.98 5660

Normal 1.00 1.00 1.00 11098

Reconnaissance 0.90 0.80 0.85 1035

Shellcode 0.54 0.50 0.52 110

Worms 0.50 0.27 0.35 15

accuracy 0.89 24700

macro avg 0.61 0.56 0.58 24700

weighted avg 0.89 0.89 0.89 24700

============================== Logistic Regression Model Test Results ==============================

Model Accuracy:

0.8553036437246964

Confusion matrix:

[[ 0 0 21 75 52 53 0 4 0 0]

[ 1 0 6 36 61 58 0 12 0 0]

[ 1 0 140 751 121 110 0 136 0 0]

[ 1 0 136 2454 267 232 0 224 0 0]

[ 2 0 33 233 1263 170 0 129 0 0]

[ 0 0 0 159 25 5440 0 36 0 0]

[ 0 0 0 0 0 0 11098 0 0 0]

[ 0 0 27 139 80 58 0 731 0 0]

[ 0 0 0 26 16 0 0 68 0 0]

[ 0 0 0 8 4 3 0 0 0 0]]

Classification report:

precision recall f1-score support

Analysis 0.00 0.00 0.00 205

Backdoor 0.00 0.00 0.00 174

DoS 0.39 0.11 0.17 1259

Exploits 0.63 0.74 0.68 3314

Fuzzers 0.67 0.69 0.68 1830

Generic 0.89 0.96 0.92 5660

Normal 1.00 1.00 1.00 11098

Reconnaissance 0.55 0.71 0.62 1035

Shellcode 0.00 0.00 0.00 110

Worms 0.00 0.00 0.00 15

accuracy 0.86 24700

macro avg 0.41 0.42 0.41 24700

weighted avg 0.83 0.86 0.84 24700

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

Model Accuracy:

0.8332388663967611

Confusion matrix:

[[ 25 1 73 17 2 85 0 2 0 0]

[ 26 0 25 13 9 91 0 10 0 0]

[ 34 0 648 307 64 119 0 87 0 0]

[ 51 1 592 2103 270 168 0 129 0 0]

[ 115 1 255 158 1048 176 0 77 0 0]

[ 1 0 36 137 37 5436 0 13 0 0]

[ 2 0 0 0 0 0 11096 0 0 0]

[ 6 0 377 152 260 15 0 225 0 0]

[ 1 0 48 0 41 0 0 20 0 0]

[ 0 0 2 11 2 0 0 0 0 0]]

Classification report:

precision recall f1-score support

Analysis 0.10 0.12 0.11 205

Backdoor 0.00 0.00 0.00 174

DoS 0.32 0.51 0.39 1259

Exploits 0.73 0.63 0.68 3314

Fuzzers 0.60 0.57 0.59 1830

Generic 0.89 0.96 0.93 5660

Normal 1.00 1.00 1.00 11098

Reconnaissance 0.40 0.22 0.28 1035

Shellcode 0.00 0.00 0.00 110

Worms 0.00 0.00 0.00 15

accuracy 0.83 24700

macro avg 0.40 0.40 0.40 24700

weighted avg 0.83 0.83 0.83 24700

============================== Gradient Booster Classifier Model Test Results ==============================

Model Accuracy:

0.8878947368421053

Confusion matrix:

[[ 15 0 69 115 6 0 0 0 0 0]

[ 0 5 26 129 14 0 0 0 0 0]

[ 0 0 634 524 57 9 2 18 14 1]

[ 0 0 541 2576 111 14 1 58 9 4]

[ 0 0 119 300 1366 3 29 4 4 5]

[ 1 0 4 154 27 5466 2 2 4 0]

[ 0 0 0 44 8 63 10969 0 6 8]

[ 0 0 74 106 14 0 0 840 1 0]

[ 0 0 2 21 18 2 3 12 52 0]

[ 0 0 0 5 1 1 0 0 0 8]]

Classification report:

precision recall f1-score support

Analysis 0.94 0.07 0.14 205

Backdoor 1.00 0.03 0.06 174

DoS 0.43 0.50 0.46 1259

Exploits 0.65 0.78 0.71 3314

Fuzzers 0.84 0.75 0.79 1830

Generic 0.98 0.97 0.97 5660

Normal 1.00 0.99 0.99 11098

Reconnaissance 0.90 0.81 0.85 1035

Shellcode 0.58 0.47 0.52 110

Worms 0.31 0.53 0.39 15

accuracy 0.89 24700

macro avg 0.76 0.59 0.59 24700

weighted avg 0.90 0.89 0.89 24700

============================== Support Vector Machine Model Test Results ==============================

Model Accuracy:

0.8780971659919028

Confusion matrix:

[[ 0 0 19 149 35 0 0 2 0 0]

[ 0 0 9 120 35 1 0 9 0 0]

[ 0 0 170 905 85 15 0 84 0 0]

[ 2 0 155 2826 169 9 0 153 0 0]

[ 0 0 56 374 1341 2 0 57 0 0]

[ 0 0 9 134 31 5472 0 14 0 0]

[ 0 0 0 0 0 0 11098 0 0 0]

[ 0 0 34 168 56 2 0 775 0 0]

[ 0 0 8 29 14 1 0 55 3 0]

[ 0 0 0 8 3 0 0 0 0 4]]

Classification report:

precision recall f1-score support

Analysis 0.00 0.00 0.00 205

Backdoor 0.00 0.00 0.00 174

DoS 0.37 0.14 0.20 1259

Exploits 0.60 0.85 0.70 3314

Fuzzers 0.76 0.73 0.75 1830

Generic 0.99 0.97 0.98 5660

Normal 1.00 1.00 1.00 11098

Reconnaissance 0.67 0.75 0.71 1035

Shellcode 1.00 0.03 0.05 110

Worms 1.00 0.27 0.42 15

accuracy 0.88 24700

macro avg 0.64 0.47 0.48 24700

weighted avg 0.87 0.88 0.86 24700

This code is evaluating the performance of each of the machine learning models on the test set.

The for loop iterates through each of the models stored in the **models** list, and for each model, it calculates the accuracy, confusion matrix, and classification report. The **metrics** module from **sklearn** library is used to calculate these metrics.

For each model, the accuracy is calculated using the **metrics.accuracy\_score()** function, which takes the true labels and predicted labels as input. The confusion matrix is calculated using the **metrics.confusion\_matrix()** function, which also takes the true labels and predicted labels as input. The classification report is calculated using the **metrics.classification\_report()** function, which takes the true labels and predicted labels as input.

Finally, the results are printed for each model in a clear and concise manner using the print statements. The title of each evaluation section includes the name of the model being evaluated, and the metrics such as accuracy, confusion matrix, and classification report are displayed in a readable format.

!pip install colorama

import colorama

from colorama import Fore

**colorama** is a Python library for printing colored text on the command-line terminal. The **Fore** class in **colorama** provides a way to change the color of the text that follows it. The **!pip install colorama** command installs the library in the current Python environment, and **import colorama** and **from colorama import Fore** commands import the necessary classes for use in the code.

while(True):

    choice = input(Fore.BLUE+'Enter \'single\' for predicting single value\nEnter \'range\' to predict a range of values : ')

    if(choice=='range'):

        print()

        start,end = map(int,input(Fore.BLUE+'Enter the range for prediction between [0,175340]: ').split())

        prediction = selected\_test[start:end]

        tar = test\_target[start:end]

        break

    elif(choice=='single'):

        predict\_column = int(input(Fore.BLUE+'Enter the value between [0,175340]: '))

        prediction = selected\_test[predict\_column:predict\_column+1]

        tar = test\_target[predict\_column:predict\_column+1]

        break

    else:

        print(Fore.RED+'Enter correct choice')

tar = list(tar)

#print(tar)

prediction

Enter 'single' for predicting single value

Enter 'range' to predict a range of values : range

Enter the range for prediction between [0,175340]: 84420 84430

|  | **dur** | **sbytes** | **dbytes** | **rate** | **sttl** | **sload** | **dload** | **synack** | **smean** | **dmean** | **ct\_srv\_src** | **ct\_dst\_sport\_ltm** | **ct\_srv\_dst** | **label** | **service** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **84420** | -0.080050 | -0.048527 | -0.065497 | -0.576697 | -1.141901 | -0.389956 | -0.257682 | 1.550324 | -0.443391 | 1.656212 | -0.775991 | -0.554373 | -0.753074 | 0.685014 | 4 |
| **84421** | -0.056545 | -0.035847 | 0.186275 | -0.576472 | -1.141901 | -0.389868 | -0.142145 | 0.296720 | -0.037873 | 3.766028 | -0.775991 | -0.554373 | -0.660111 | 0.685014 | 5 |
| **84422** | -0.101807 | -0.047554 | -0.102057 | -0.576690 | 0.723268 | -0.389943 | -0.276150 | 0.530224 | -0.409190 | -0.306498 | -0.495729 | -0.554373 | -0.567147 | 0.685014 | 0 |
| **84423** | -0.142546 | -0.047486 | -0.102057 | -0.576611 | 0.723268 | -0.389925 | -0.275509 | 1.060505 | -0.399419 | -0.306498 | -0.122048 | -0.554373 | -0.102330 | 0.685014 | 0 |
| **84424** | -0.056388 | -0.047486 | -0.102057 | -0.576728 | 0.723268 | -0.389952 | -0.276464 | 1.844497 | -0.399419 | -0.306498 | -0.589150 | -0.554373 | -0.288257 | -1.459825 | 0 |
| **84425** | -0.106216 | -0.047325 | -0.102057 | -0.576684 | 0.723268 | -0.389940 | -0.276105 | 1.738552 | -0.389647 | -0.306498 | -0.589150 | -0.554373 | -0.474184 | -1.459825 | 0 |
| **84426** | 0.182604 | -0.039698 | -0.095695 | -0.576712 | 0.723268 | -0.389942 | -0.275741 | 1.312719 | -0.282161 | -0.271657 | -0.775991 | -0.554373 | -0.753074 | 0.685014 | 0 |
| **84427** | -0.081673 | -0.047463 | -0.101459 | -0.576695 | 0.723268 | -0.389947 | -0.275974 | 1.959590 | -0.399419 | -0.310369 | -0.589150 | -0.554373 | -0.474184 | -1.459825 | 0 |
| **84428** | -0.209774 | -0.049649 | -0.103923 | 0.178922 | 0.723268 | 0.055989 | -0.277208 | -0.484346 | -0.257732 | -0.480703 | -0.682570 | -0.554373 | -0.753074 | 0.685014 | 0 |
| **84429** | -0.029787 | -0.047486 | -0.101459 | -0.576731 | 0.723268 | -0.389955 | -0.276330 | 1.564564 | -0.399419 | -0.310369 | -0.589150 | -0.554373 | -0.288257 | -1.459825 | 0 |

This code block prompts the user to choose between two options, 'single' or 'range', for predicting values from the test dataset. If the user chooses 'range', they are prompted to enter a start and end index for the range of values they want to predict. If the user chooses 'single', they are prompted to enter a single index to predict a single value.

Once the user input is received, the corresponding portion of the test dataset is extracted and stored in the **prediction** variable. Additionally, the target values for the corresponding test data are extracted and stored in the **tar** variable. Both **prediction** and **tar** are printed for reference.

The **colorama** library is used to add color to the printed text. The **Fore.BLUE** and **Fore.RED** are used to print the prompts in blue and error messages in red, respectively.

DTC\_prediction\_result = DTC\_Classifier.predict(prediction)

for i in range(len(DTC\_prediction\_result)):

    print(tar[i]," ",DTC\_prediction\_result[i])

Exploits Exploits

Exploits Generic

Fuzzers DoS

Fuzzers DoS

Normal Normal

Normal Normal

Exploits Exploits

Normal Normal

Reconnaissance Exploits

Normal Normal

This code is using the trained decision tree classifier (**DTC\_Classifier**) to predict the target class for the given test data (**prediction**). It then loops through the predicted results (**DTC\_prediction\_result**) and prints out the corresponding actual target value (**tar[i]**) and predicted target value (**DTC\_prediction\_result[i]**) for each prediction.

So, the output will be a set of lines where each line contains the actual target value and the predicted target value separated by a space. This allows us to see how well the model is performing on the given test data.