AI-Project (Intrusion Detection System with ML&DL)

Group Members:

Qasim Akbar (231300)

Waqar ul Hasan (231270)

M.Azeem Khan (231294)

Ahmed Mustafa (231306)

Date of Submission: 30-Dec-2025

Intrusion Detection System with ML&DL

PREFACE:

With the rapid growth of computer networks and the increasing number of applications running on them, network security has become a major concern. Most systems today suffer from security vulnerabilities, which can be exploited by attackers and may lead to serious economic and data losses. Therefore, detecting malicious activities in network traffic accurately and in real time is extremely important.

In this project, we build an AI-based Intrusion Detection System (IDS) using machine learning techniques. The system is trained on the NSL-KDD dataset, which is a widely used benchmark dataset for network intrusion detection. The goal of this project is to classify network connections as either normal or attack.

Different machine learning models such as Logistic Regression, K-Nearest Neighbors, Naive Bayes, Support Vector Machines, Decision Trees, Random Forests, XGBoost, and Neural Networks are implemented and evaluated. Additionally, Principal Component Analysis (PCA) is used to reduce dimensionality and improve efficiency.

The performance of each model is analyzed using metrics such as accuracy, precision, recall, and confusion matrices. This project demonstrates how artificial intelligence can be effectively used to detect malicious network behavior and improve overall network security.

Double-click (or enter) to edit

```
!unzip -q "archive (6).zip" -d dataset
```

Problem Statement

With the rapid growth of computer networks and the increasing number of applications running on them, network security has become a major concern. Most systems today suffer from security vulnerabilities that can be exploited by attackers, leading to serious economic and data losses. Therefore, accurate and timely detection of malicious activities in network traffic is essential.

In this project, an AI-based Intrusion Detection System (IDS) is developed to identify whether a network connection is normal or malicious. A machine learning model using a Support Vector Machine (SVM) classifier is trained to analyze network traffic and detect intrusion attempts.

Intrusion Detection System (IDS)

An Intrusion Detection System (IDS) monitors network or system activities to identify suspicious behavior and security policy violations. It detects malicious actions and generates alerts, which can be further analyzed using security management systems such as SIEM.

IDS can be classified into:

 # Host-Based IDS (HIDS): Monitors activities on a single host, including processes, logs, and local network traffic. It provides deep visibility into the host but limited network-wide context.**

Network-Based IDS (NIDS):

Monitors network traffic across an entire network, analyzing packet data to detect attacks. It offers a broader view of network activity but limited insight into individual hosts.

Detection Methods in IDS

Signature-Based Detection: Detects attacks by matching known patterns or signatures of previously identified malware. While effective for known threats, it cannot detect new or unknown attacks.

Anomaly-Based Detection: Uses machine learning to model normal network behavior and flags any significant deviation as suspicious. This method is effective in detecting unknown and emerging attacks and is more adaptable than signature-based approaches.

Project Objectives

To study the problem of network security and malware intrusion in computer networks.

To analyze the NSL-KDD dataset for detecting malicious network traffic.

To design and develop an AI-based Intrusion Detection System (IDS).

To classify network connections as normal or attack using machine learning techniques.

To implement and compare multiple machine learning and deep learning models.

To evaluate the performance of models using metrics such as accuracy, precision, recall, and confusion matrix.

To identify the most effective model for malware and intrusion detection.

Importing necessary libraries

```
import numpy as np
import pandas as pd
import warnings
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras import regularizers
import xgboost as xgb
from sklearn.decomposition import PCA
from sklearn import tree
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import RobustScaler
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn import metrics
pd.set_option('display.max_columns',None)
warnings.filterwarnings('ignore')
%matplotlib inline
```

Exploring the dataset

```
import pandas as pd

# Adjust the path based on your folder structure
data_train = pd.read_csv("dataset/KDDTrain+.txt", header=None)
data_test = pd.read_csv("dataset/KDDTest+.txt", header=None)

# Quick check
print(data_train.shape)
print(data_test.shape)
```

```
(125973, 43)
(22544, 43)
```

```
# Check data
data_train.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	2
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	0.0	0.0	0.
1	0	udp	other	SF	146	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13	1	0.0	0.0	0.
2	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	123	6	1.0	1.0	0.
3	0	tcp	http	SF	232	8153	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	5	5	0.2	0.2	0.
4	0	tcp	http	SF	199	420	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	30	32	0.0	0.0	0.

```
columns = (['duration','protocol_type','service','flag','src_bytes','dst_bytes','land','wrong_fragment','urgen
,'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','ser
,'srv_serror_rate','rerror_rate','srv_rerror_rate','same_srv_rate','diff_srv_rate','srv_diff_host_rate','dst_h
,'dst_host_same_srv_rate','dst_host_diff_srv_rate','dst_host_same_src_port_rate','dst_host_srv_diff_host_rate'
,'dst_host_srv_serror_rate','dst_host_rerror_rate','dst_host_srv_rerror_rate','outcome','level'])
```

```
# Assign name for columns
data_train.columns = columns
```

```
data_train.head()
```

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

```
data_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 125973 entries, 0 to 125972
Data columns (total 43 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   duration                     125973 non-null  int64
 1   protocol_type                125973 non-null  object
 2   service                      125973 non-null  object
 3   flag                         125973 non-null  object
 4   src_bytes                    125973 non-null  int64
 5   dst_bytes                    125973 non-null  int64
 6   land                         125973 non-null  int64
 7   wrong_fragment               125973 non-null  int64
 8   urgent                       125973 non-null  int64
 9   hot                          125973 non-null  int64
 10  num_failed_logins            125973 non-null  int64
 11  logged_in                    125973 non-null  int64
 12  num_compromised              125973 non-null  int64
 13  root_shell                   125973 non-null  int64
 14  su_attempted                 125973 non-null  int64
 15  num_root                     125973 non-null  int64
 16  num_file_creations           125973 non-null  int64
 17  num_shells                   125973 non-null  int64
 18  num_access_files             125973 non-null  int64
 19  num_outbound_cmds            125973 non-null  int64
 20  is_host_login                125973 non-null  int64
 21  is_guest_login               125973 non-null  int64
 22  count                        125973 non-null  int64
 23  srv_count                    125973 non-null  int64
 24  serror_rate                  125973 non-null  float64
 25  srv_serror_rate              125973 non-null  float64
 26  rerror_rate                  125973 non-null  float64
 27  srv_rerror_rate              125973 non-null  float64
 28  same_srv_rate                125973 non-null  float64
 29  diff_srv_rate                125973 non-null  float64
 30  srv_diff_host_rate           125973 non-null  float64
 31  dst_host_count               125973 non-null  int64
 32  dst_host_srv_count           125973 non-null  int64
 33  dst_host_same_srv_rate       125973 non-null  float64
 34  dst_host_diff_srv_rate       125973 non-null  float64
 35  dst_host_same_src_port_rate  125973 non-null  float64
 36  dst_host_srv_diff_host_rate  125973 non-null  float64
 37  dst_host_serror_rate         125973 non-null  float64
 38  dst_host_srv_serror_rate     125973 non-null  float64
 39  dst_host_rerror_rate         125973 non-null  float64
 40  dst_host_srv_rerror_rate     125973 non-null  float64
 41  outcome                      125973 non-null  object
 42  level                        125973 non-null  int64
dtypes: float64(15), int64(24), object(4)
memory usage: 41.3+ MB
```

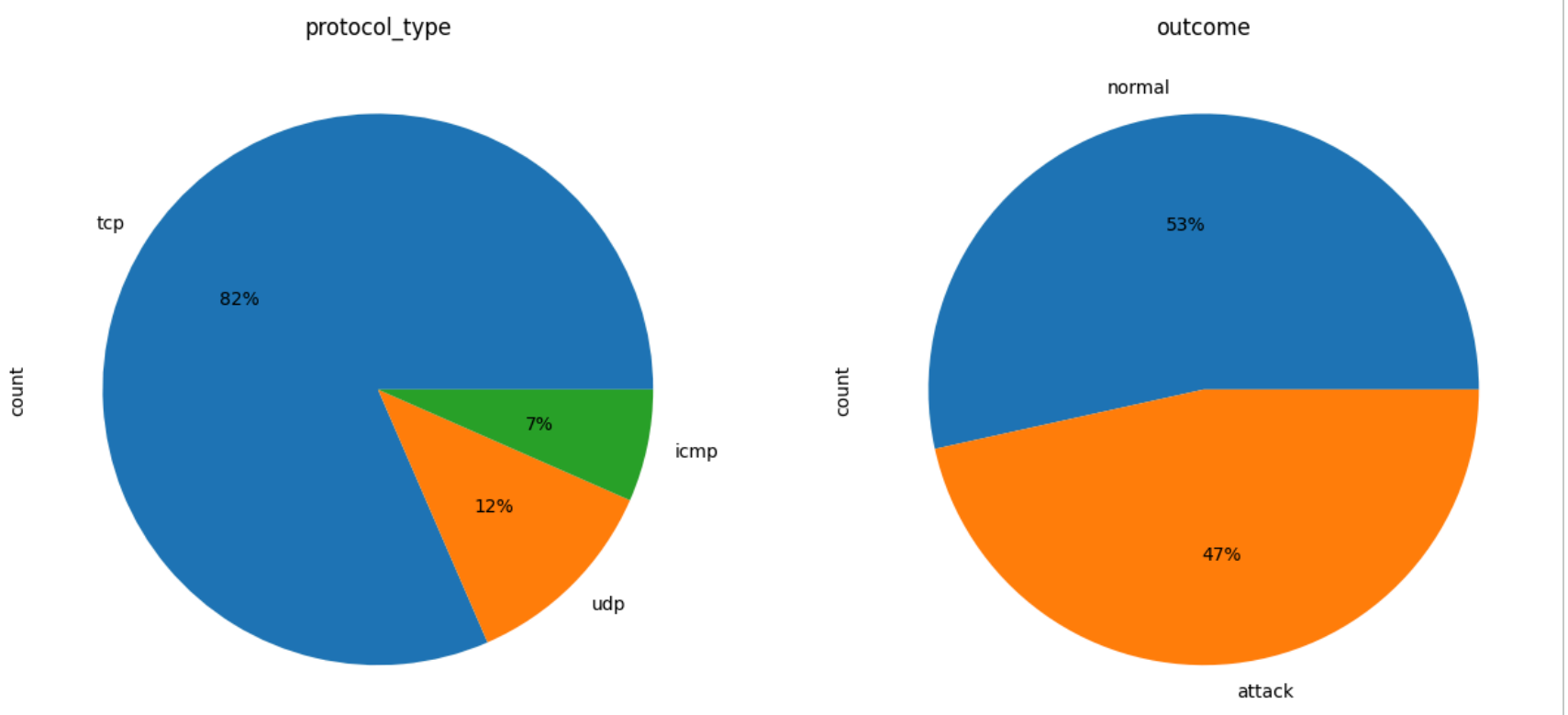
```
data_train.describe().style.background_gradient(cmap='Blues').set_properties(**{'font-family':'Segoe UI'})
```

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	h
count	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000(
mean	287.144650	45566.743000	19779.114421	0.000198	0.022687	0.000111	0.204
std	2604.515310	5870331.181891	4021269.151440	0.014086	0.253530	0.014366	2.149
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000(
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000(
50%	0.000000	44.000000	0.000000	0.000000	0.000000	0.000000	0.000(
75%	0.000000	276.000000	516.000000	0.000000	0.000000	0.000000	0.000(
max	42908.000000	1379963888.000000	1309937401.000000	1.000000	3.000000	3.000000	77.000(

```
data_train.loc[data_train['outcome'] == "normal", "outcome"] = 'normal'
data_train.loc[data_train['outcome'] != 'normal', "outcome"] = 'attack'
```

```
def pie_plot(df, cols_list, rows, cols):
    fig, axes = plt.subplots(rows, cols)
    for ax, col in zip(axes.ravel(), cols_list):
        df[col].value_counts().plot(ax=ax, kind='pie', figsize=(15, 15), fontsize=10, autopct='%1.0f%%')
        ax.set_title(str(col), fontsize = 12)
    plt.show()
```

```
pie_plot(data_train, ['protocol_type', 'outcome'], 1, 2)
```



Preprocessing the data

```
def Scaling(df_num, cols):
    std_scaler = RobustScaler()
    std_scaler_temp = std_scaler.fit_transform(df_num)
    std_df = pd.DataFrame(std_scaler_temp, columns =cols)
    return std_df
```

```
cat_cols = ['is_host_login','protocol_type','service','flag','land', 'logged_in','is_guest_login', 'level', 'c
def preprocess(dataframe):
    df_num = dataframe.drop(cat_cols, axis=1)
    num_cols = df_num.columns
    scaled_df = Scaling(df_num, num_cols)

    dataframe.drop(labels=num_cols, axis="columns", inplace=True)
    dataframe[num_cols] = scaled_df[num_cols]

    dataframe.loc[dataframe['outcome'] == "normal", "outcome"] = 0
    dataframe.loc[dataframe['outcome'] != 0, "outcome"] = 1

    dataframe = pd.get_dummies(dataframe, columns = ['protocol_type', 'service', 'flag'])
    return dataframe
```

```
scaled_train = preprocess(data_train)
```

Principal Component Analysis

Principal component analysis, or PCA, is a statistical technique to convert high dimensional data to low dimensional data by selecting the most important features that capture maximum information about the dataset. The features are selected on the basis of variance that they cause in the output. The feature that causes highest variance is the first principal

component. The feature that is responsible for second highest variance is considered the second principal component, and so on. It is important to mention that principal components do not have any correlation with each other.

Advantages of PCA

There are two main advantages of dimensionality reduction with PCA.

- The training time of the algorithms reduces significantly with less number of features.
- It is not always possible to analyze data in high dimensions. For instance if there are 100 features in a dataset. Total number of scatter plots required to visualize the data would be 100(100-1)2 = 4950. Practically it is not possible to analyze data this way.

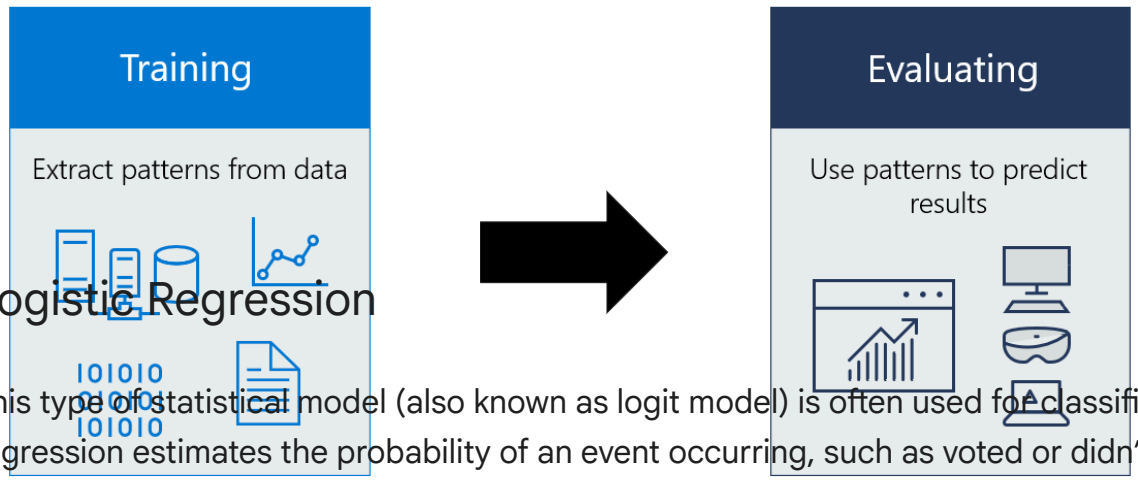
```
x = scaled_train.drop(['outcome', 'level'] , axis = 1).values
y = scaled_train['outcome'].values
y_reg = scaled_train['level'].values

x = x.astype(np.float32)
y = y.astype('int')

pca = PCA(n_components=20)
pca = pca.fit(x)
x_reduced = pca.transform(x)
print("Number of original features is {} and of reduced features is {}".format(x.shape[1], x_reduced.shape[1])

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
x_train_reduced, x_test_reduced, y_train_reduced, y_test_reduced = train_test_split(x_reduced, y, test_size=0.
x_train_reg, x_test_reg, y_train_reg, y_test_reg = train_test_split(x, y_reg, test_size=0.2, random_state=42)
```

```
Number of original features is 122 and of reduced features is 20
```

```
kernal_evals = dict()
def evaluate_classification(model, name, X_train, X_test, y_train, y_test):
    train_accuracy = metrics.accuracy_score(y_train, model.predict(X_train))
    test_accuracy = metrics.accuracy_score(y_test, model.predict(X_test))

    train_precision = metrics.precision_score(y_train, model.predict(X_train))
    test_precision = metrics.precision_score(y_test, model.predict(X_test))

    train_recall = metrics.recall_score(y_train, model.predict(X_train))
    test_recall = metrics.recall_score(y_test, model.predict(X_test))

    kernal_evals[str(name)] = [train_accuracy, test_accuracy, train_precision, test_precision, train_recall, t
    print("Training Accuracy " + str(name) + " {}  Test Accuracy ".format(train_accuracy*100) + str(name) + "
    print("Training Precesion " + str(name) + " {}  Test Precesion ".format(train_precision*100) + str(name) +
    print("Training Recall " + str(name) + " {}  Test Recall ".format(train_recall*100) + str(name) + " {}".fo

    actual = y_test
    predicted = model.predict(X_test)
    confusion_matrix = metrics.confusion_matrix(actual, predicted)
    cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = ['normal

    fig, ax = plt.subplots(figsize=(10,10))
    ax.grid(False)
    cm_display.plot(ax=ax)
```

Modeling

The process of modeling means training a machine learning algorithm to predict the labels from the features, tuning it for the business need, and validating it on holdout data. The output from modeling is a trained model that can be used for inference, making predictions on new data points.



Logistic Regression

This type of statistical model (also known as logit model) is often used for classification and predictive analytics. Logistic regression estimates the probability of an event occurring, such as voted or didn't vote, based on a given dataset of independent variables. Since the outcome is a probability, the dependent variable is bounded between 0 and 1. In logistic regression, a logit transformation is applied on the odds – that is, the probability of success divided by the probability of failure. This is also commonly known as the log odds, or the natural logarithm of odds, and this logistic function is represented by the following formulas:

A machine learning model itself is a file that has been trained to recognize certain types of patterns. You train a model over a set of data, providing it an algorithm that it can use to reason over and learn from those data. Once you have trained the model, you can use it to reason over data that it hasn't seen before, and make predictions about those data.

n recognize a user's emotions based on their facial
s of faces that are each tagged with a certain emotion, and
ıize any user's emotion

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}},$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

In this logistic regression equation, h is the dependent or response variable and x is the independent variable. The beta parameter, or coefficient, in this model is commonly estimated via maximum likelihood estimation (MLE). This method tests different values of beta through multiple iterations to optimize for the best fit of log odds. All of these iterations produce the log likelihood function, and logistic regression seeks to maximize this function to find the best parameter estimate. Once the optimal coefficient (or coefficients if there is more than one independent variable) is found, the conditional probabilities for each observation can be calculated, logged, and summed together to yield a predicted probability. For binary classification, a probability less than .5 will predict 0 while a probability greater than 0 will predict 1. After the model has been computed, it's best practice to evaluate the how well the model predicts the dependent variable, which is called goodness of fit.

Binary logistic regression:

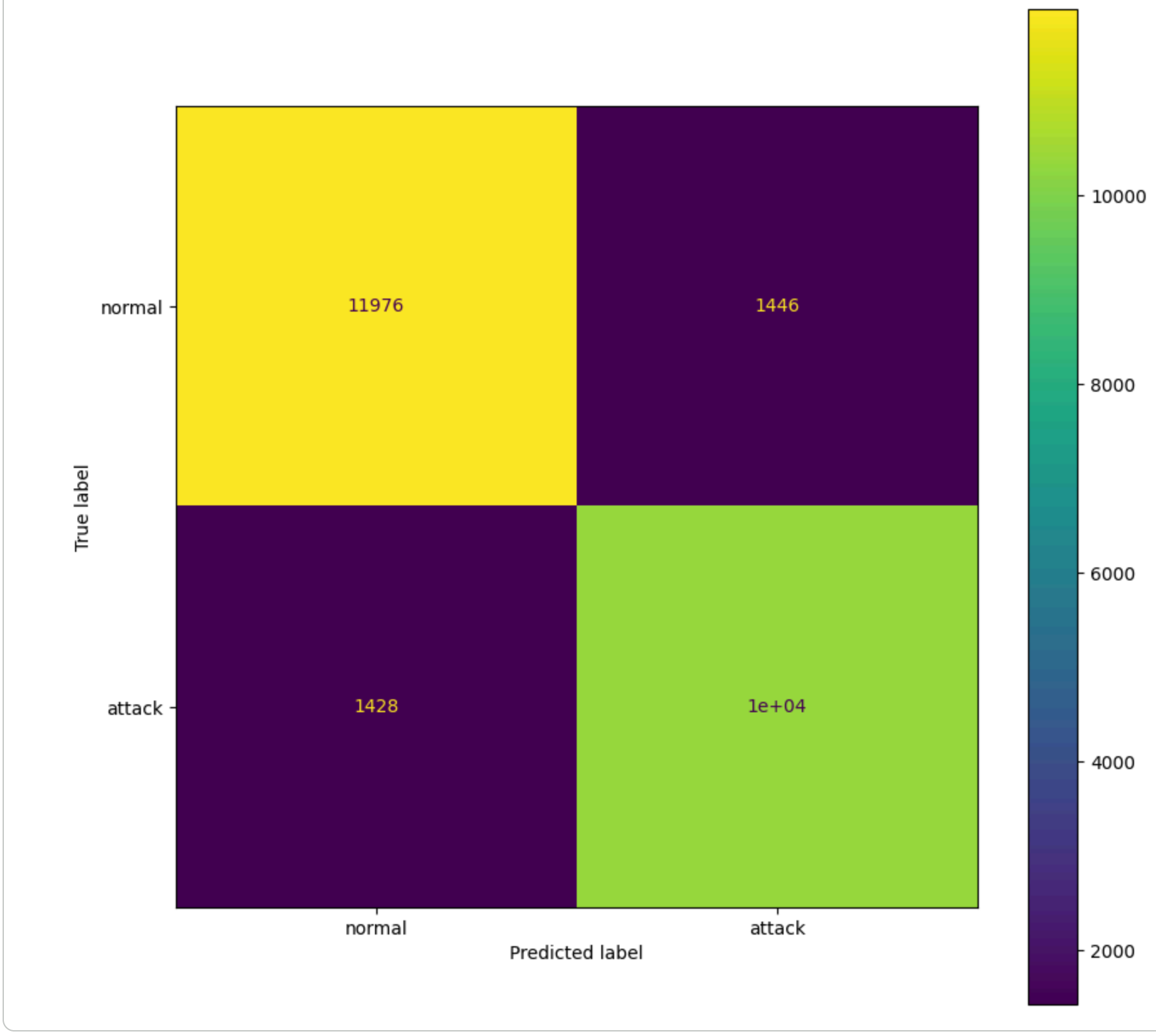
In this approach, the response or dependent variable is dichotomous in nature—i.e. it has only two possible outcomes (e.g. 0 or 1). Some popular examples of its use include predicting if an e-mail is spam or not spam or if a tumor is malignant or not malignant. Within logistic regression, this is the most commonly used approach, and more generally, it is one of the most common classifiers for binary classification.

Multinomial logistic regression:

In this type of logistic regression model, the dependent variable has three or more possible outcomes; however, these values have no specified order. For example, movie studios want to predict what genre of film a moviegoer is likely to see to market films more effectively. A multinomial logistic regression model can help the studio to determine the strength of influence a person's age, gender, and dating status may have on the type of film that they prefer. The studio can then orient an advertising campaign of a specific movie toward a group of people likely to go see it.

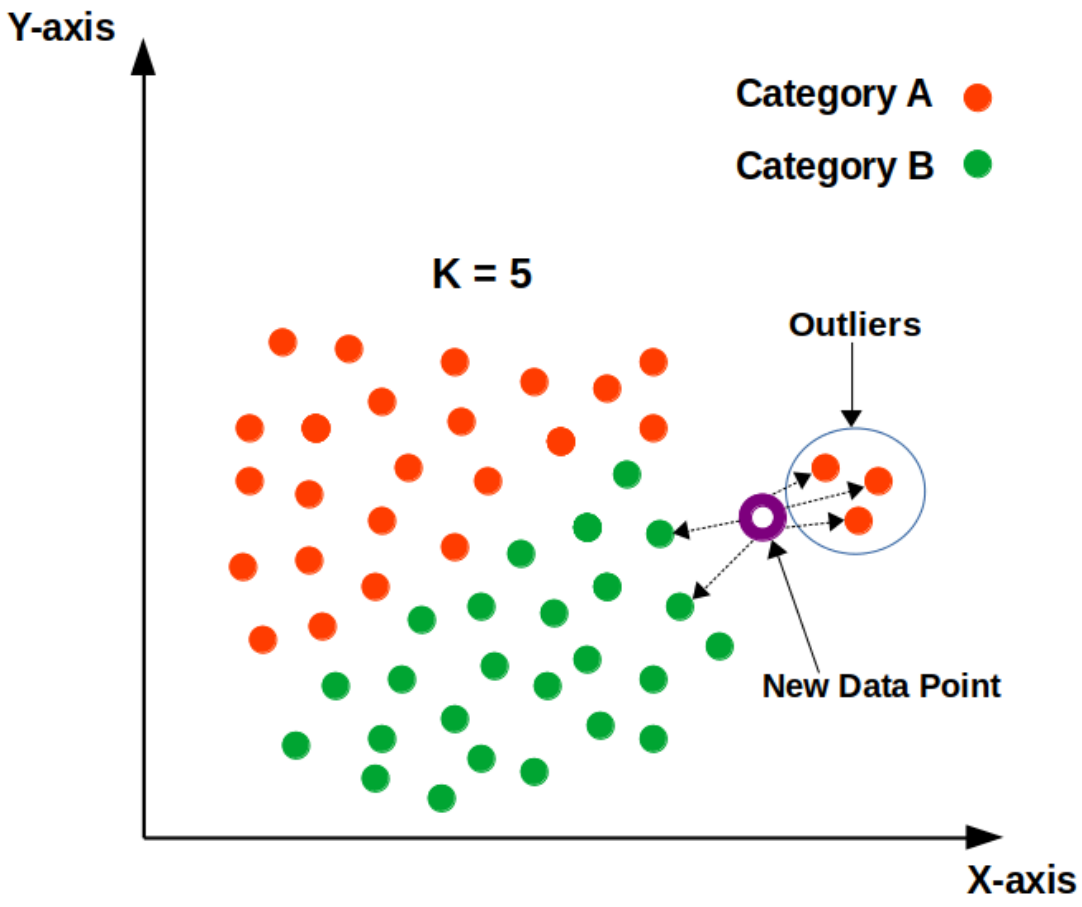
```
lr = LogisticRegression().fit(x_train, y_train)
evaluate_classification(lr, "Logistic Regression", x_train, x_test, y_train, y_test)
```

```
Training Accuracy Logistic Regression 89.07698108714203  Test Accuracy Logistic Regression 88.59297479658662
Training Precesion Logistic Regression 88.201444981991  Test Precesion Logistic Regression 87.73640912560427
Training Recall Logistic Regression 88.3219156155964  Test Recall Logistic Regression 87.87055126136075
```



k-nearest neighbors

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. While it can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another.



Determine your distance metrics

In order to determine which data points are closest to a given query point, the distance between the query point and the other data points will need to be calculated. These distance metrics help to form decision boundaries, which partitions query points into different regions. You commonly will see decision boundaries visualized with Voronoi diagram

```
knn = KNeighborsClassifier(n_neighbors=20).fit(x_train, y_train)
evaluate_classification(knn, "KNeighborsClassifier", x_train, x_test, y_train, y_test)
```

```
Training Accuracy KNeighborsClassifier 99.02260413979242  Test Accuracy KNeighborsClassifier 98.8727922206787
Training Precesion KNeighborsClassifier 99.18089029933957  Test Precesion KNeighborsClassifier 99.0186876013311
Training Recall KNeighborsClassifier 98.71310583263974  Test Recall KNeighborsClassifier 98.56451201902658
```

Given two points A and B in d dimensional space such that $A = [a_1, a_2, \ldots, a_d]$ and $B = [b_1, b_2, \ldots, b_d]$, the

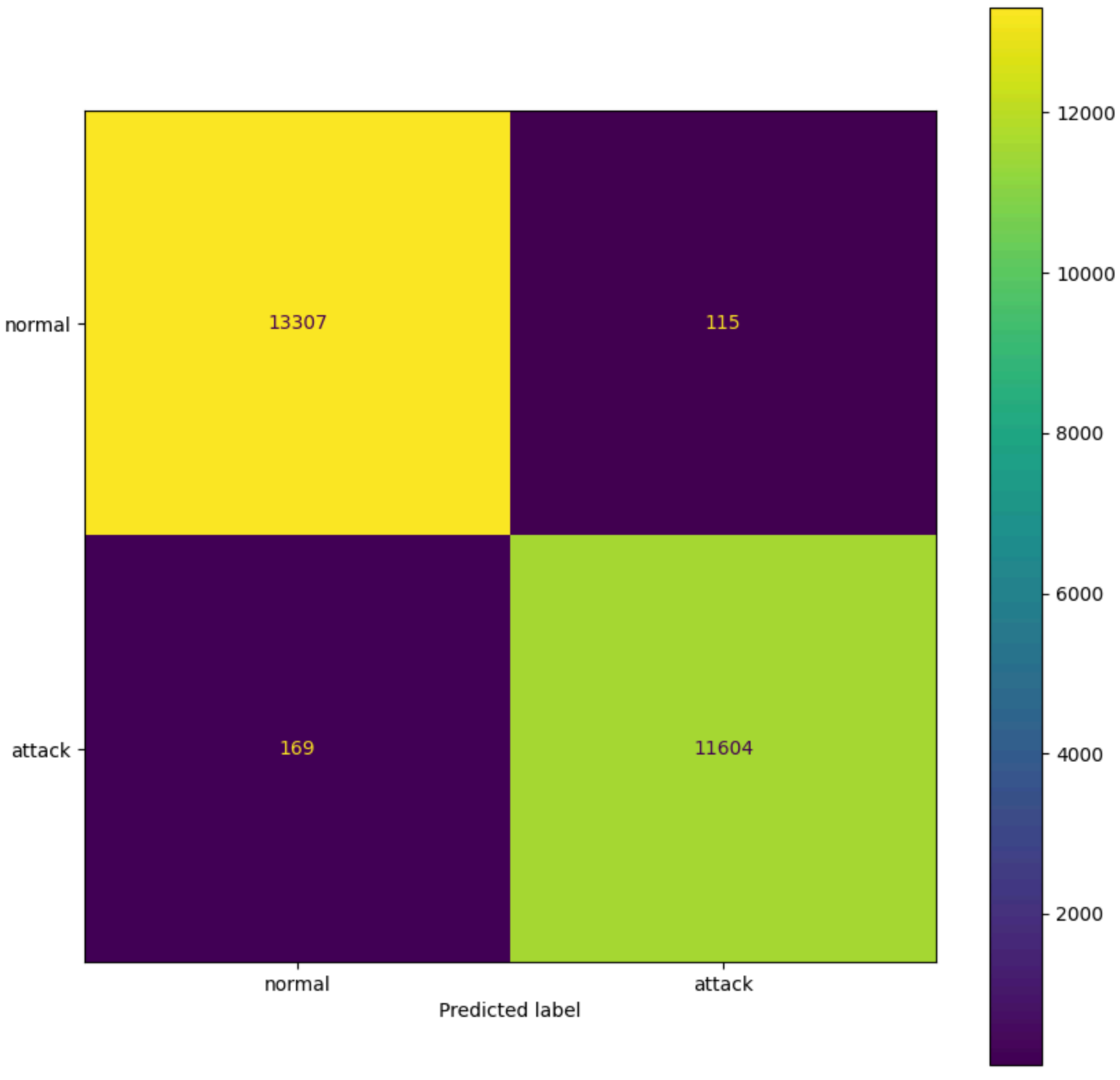
Euclidean distance between A and B is defined as:

The
dista

Man

Give
, the

The
dista



Naive Bayes

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle. Every pair of features being classified is independent of each other. The assumptions made by Naive Bayes are not generally correct in real-world situations. In-fact, the independence assumption is never correct but often works well in practice.

Now, it is important to know about Bayes' theorem.

Bayes' Theorem

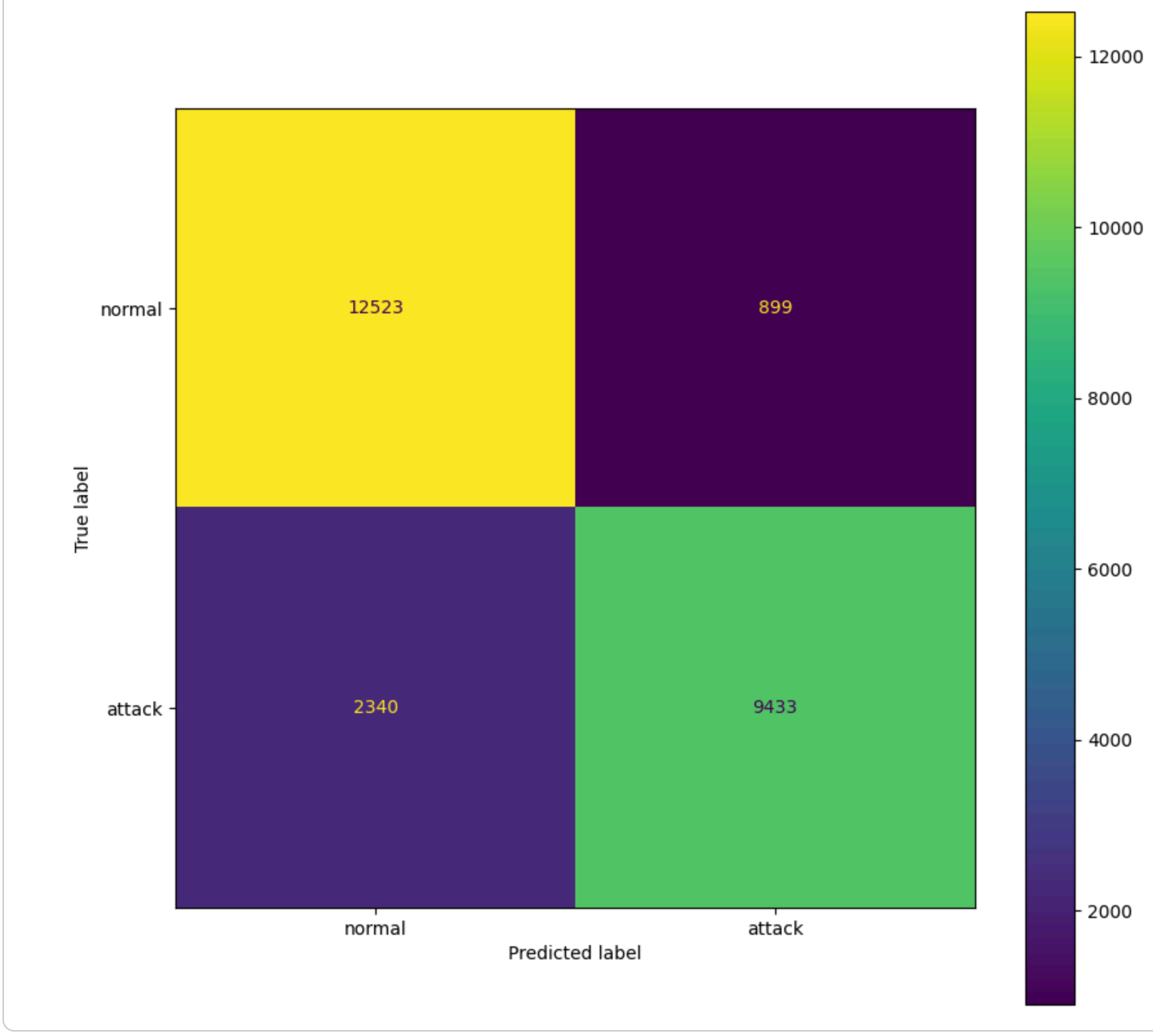
Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is stated mathematically as the following equation:

where A and B are events and P(B) ≠ 0.

- Basically, we are trying to find probability of event A, given the event B is true. Event B is also termed as evidence.
- P(A) is the priori of A (the prior probability, i.e. Probability of event before evidence is seen). The evidence is an attribute value of an unknown instance(here, it is event B).
- P(A|B) is a posteriori probability of B, i.e. probability of event after evidence is seen.

```
gnb = GaussianNB().fit(x_train, y_train)
evaluate_classification(gnb, "GaussianNB", x_train, x_test, y_train, y_test)
```

```
Training Accuracy GaussianNB 87.40697374426958  Test Accuracy GaussianNB 87.14427465767017
Training Precesion GaussianNB 91.7942946616431  Test Precesion GaussianNB 91.29887727448704
Training Recall GaussianNB 80.07341485797212  Test Recall GaussianNB 80.12401257113734
```

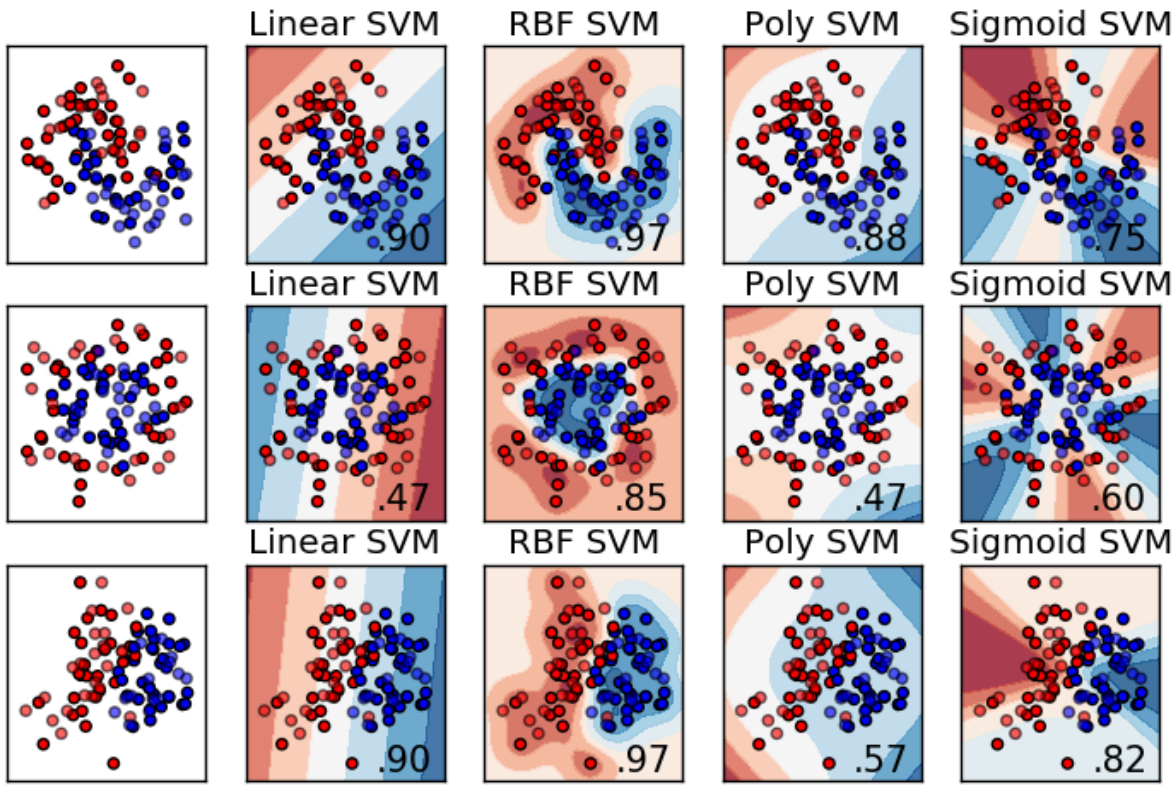


Support Vector Machines

Support Vector Machine (SVM) is a relatively simple Supervised Machine Learning Algorithm used for classification and/or regression. It is more preferred for classification but is sometimes very useful for regression as well. Basically, SVM finds a hyper-plane that creates a boundary between the types of data. In 2-dimensional space, this hyper-plane is nothing but a line. In SVM, we plot each data item in the dataset in an N-dimensional space, where N is the number of

features/attributes in the data. Next, find the optimal hyperplane to separate the data. So by this, you must have understood that inherently, SVM can only perform binary classification (i.e., choose between two classes). However, there are various techniques to use for multi-class problems. Support Vector Machine for Multi-CLass Problems To perform SVM on multi-class problems, we can create a binary classifier for each class of the data. The two results of each classifier will be :

- The data point belongs to that class OR
- The data point does not belong to that class.



For example, in a class of fruits, to perform multi-class classification, we can create a binary classifier for each fruit. For say, the 'mango' class, there will be a binary classifier to predict if it IS a mango OR it is NOT a mango. The classifier with the highest score is chosen as the output of the SVM. SVM for complex (Non Linearly Separable) SVM works very well without any modifications for linearly separable data. Linearly Separable Data is any data that can be plotted in a graph and can be separated into classes using a straight line.

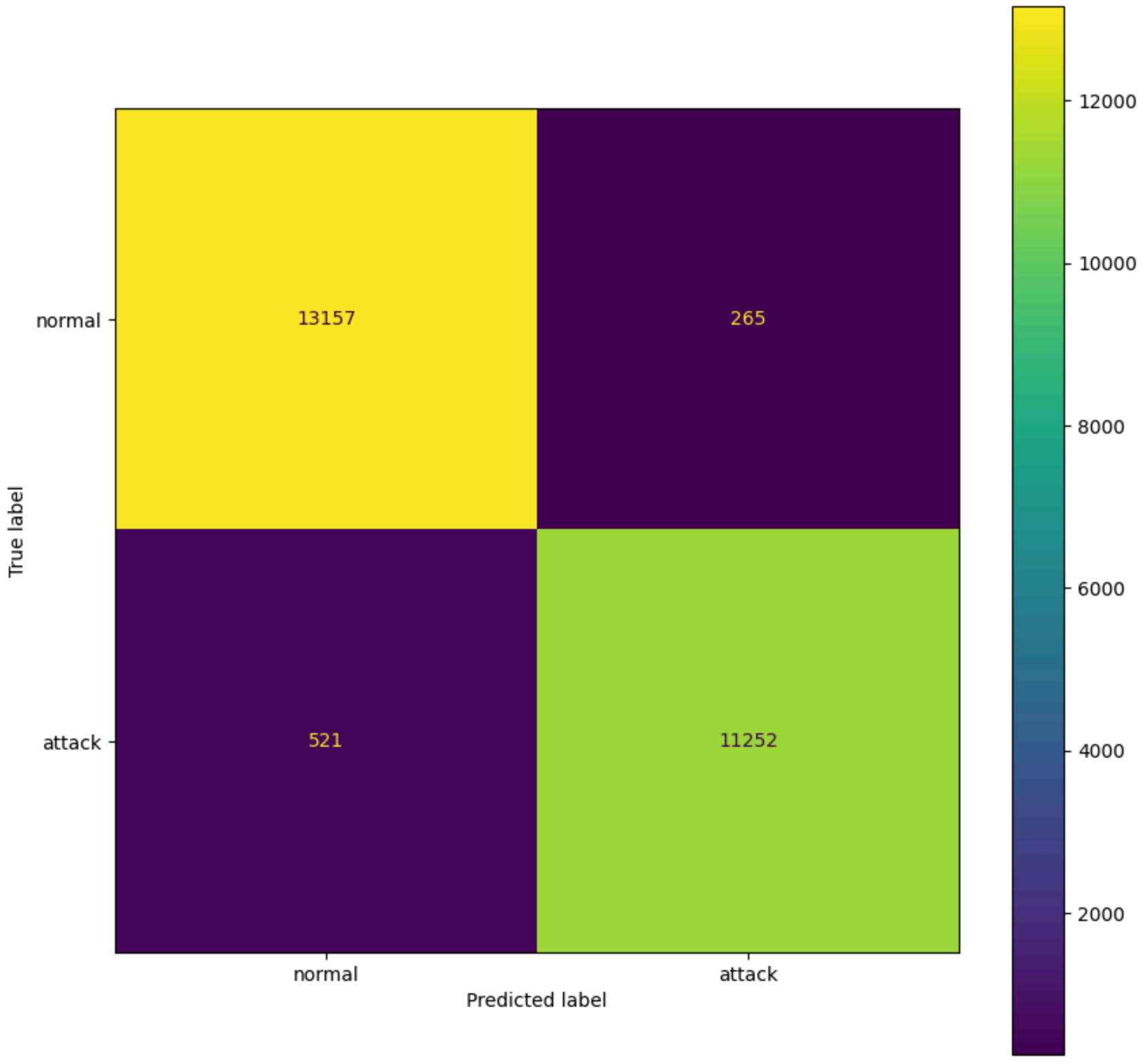
We use Kernelized SVM for non-linearly separable data. Say, we have some non-linearly separable data in one dimension. We can transform this data into two dimensions and the data will become linearly separable in two dimensions. This is done by mapping each 1-D data point to a corresponding 2-D ordered pair. So for any non-linearly separable data in any dimension, we can just map the data to a higher dimension and then make it linearly separable. This is a very powerful and general transformation. A kernel is nothing but a measure of similarity between data points. The kernel function in a kernelized SVM tells you, that given two data points in the original feature space, what the similarity is between the points in the newly transformed feature space. There are various kernel functions available, but two are very popular :

- Radial Basis Function Kernel (RBF): The similarity between two points in the transformed feature space is an exponentially decaying function of the distance between the vectors and the original input space as shown below. RBF is the default kernel used in SVM.
- Polynomial Kernel: The Polynomial kernel takes an additional parameter, 'degree' that controls the model's complexity and computational cost of the transformation

```
lin_svc = svm.LinearSVC().fit(x_train, y_train)
```

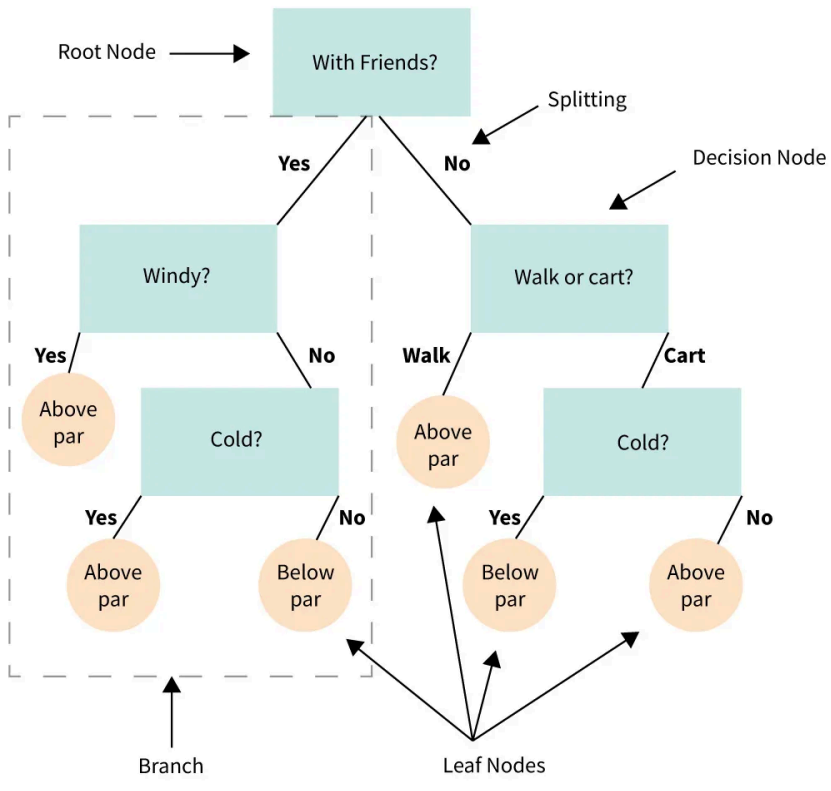
```
evaluate_classification(lin_svc, "Linear SVC(LBasedImpl)", x_train, x_test, y_train, y_test)
```

```
Training Accuracy Linear SVC(LBasedImpl) 97.1184187024946  Test Accuracy Linear SVC(LBasedImpl) 96.880333399484
Training Precesion Linear SVC(LBasedImpl) 97.91771144495563  Test Precesion Linear SVC(LBasedImpl) 97.699053572
Training Recall Linear SVC(LBasedImpl) 95.84053609919543  Test Recall Linear SVC(LBasedImpl) 95.57461989297545
```



Decision Tree

Decision Tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart-like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

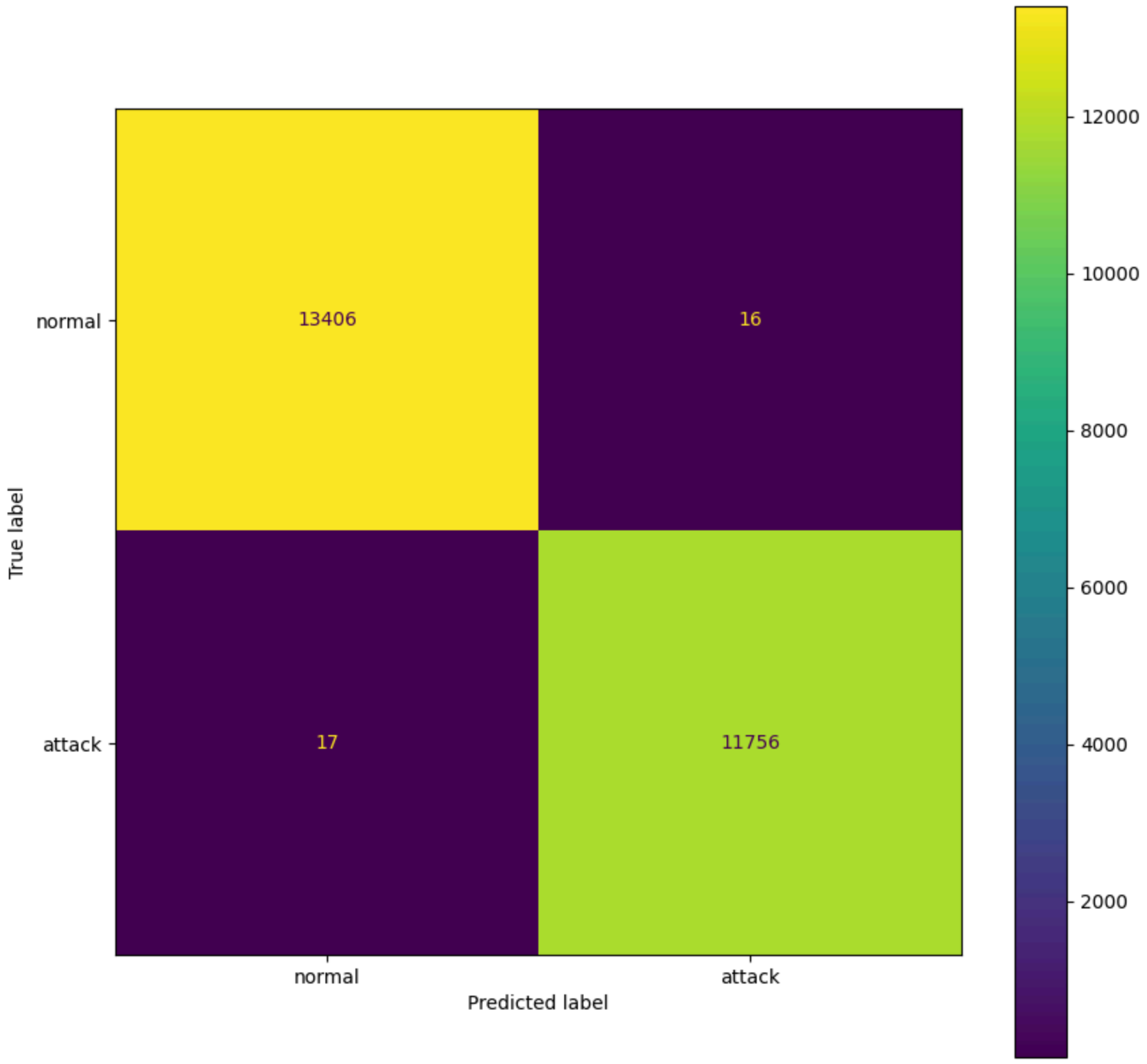


A tree can be “learned” by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at

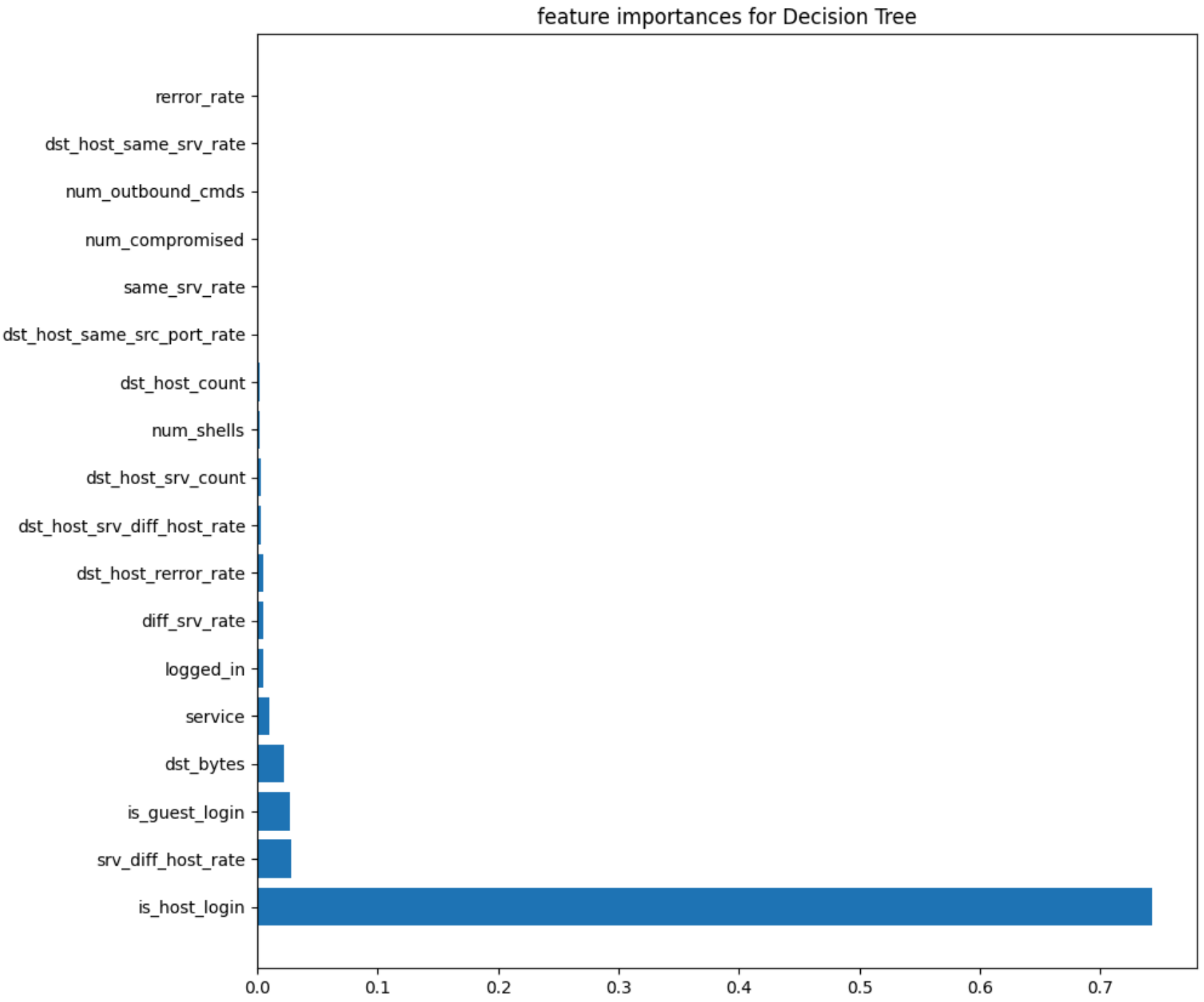
a node all has the same value of the target variable, or when splitting no longer adds value to the predictions.Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute as shown in the above figure. This process is then repeated for the subtree rooted at the new node. The decision tree in above figure classifies a particular morning according to whether it is suitable for playing tennis and returning the classification associated with the particular leaf.(in this case Yes or No).

```
dt = DecisionTreeClassifier(max_depth=3).fit(x_train, y_train)
tdt = DecisionTreeClassifier().fit(x_train, y_train)
evaluate_classification(tdt, "DecisionTreeClassifier", x_train, x_test, y_train, y_test)
```

```
Training Accuracy DecisionTreeClassifier 99.99801543987775  Test Accuracy DecisionTreeClassifier 99.86902163127
Training Precesion DecisionTreeClassifier 100.0  Test Precesion DecisionTreeClassifier 99.864084267754
Training Recall DecisionTreeClassifier 99.99573169430394  Test Recall DecisionTreeClassifier 99.85560180073048
```

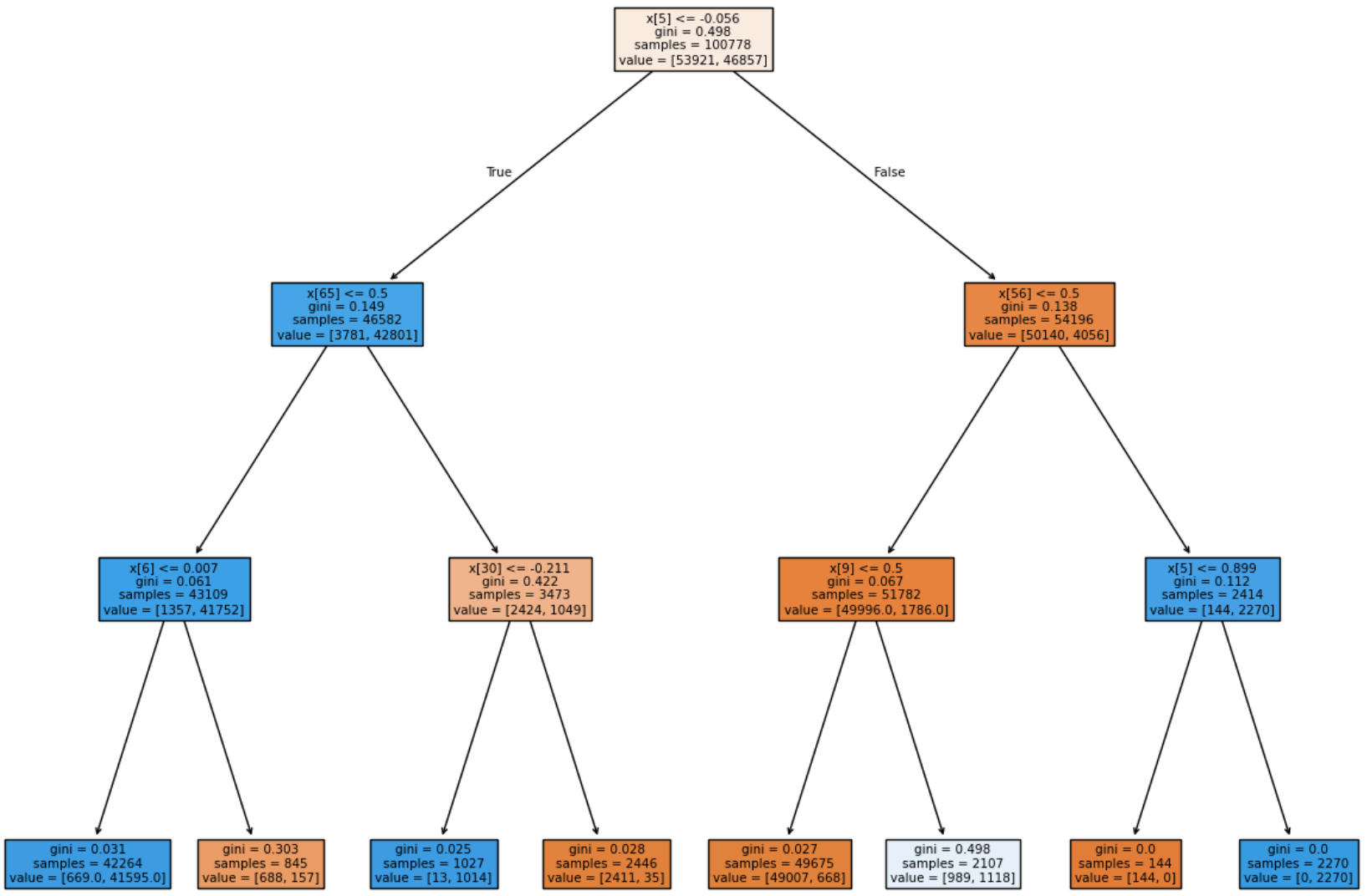



```
def f_importances(coef, names, top=-1):
    imp = coef
    imp, names = zip(*sorted(list(zip(imp, names))))

    # Show all features
    if top == -1:
        top = len(names)

    plt.figure(figsize=(10,10))
    plt.barh(range(top), imp[::-1][0:top], align='center')
    plt.yticks(range(top), names[::-1][0:top])
    plt.title('feature importances for Decision Tree')
    plt.show()

features_names = data_train.drop(['outcome', 'level'] , axis = 1)
f_importances(abs(tdt.feature_importances_), features_names, top=18)
```

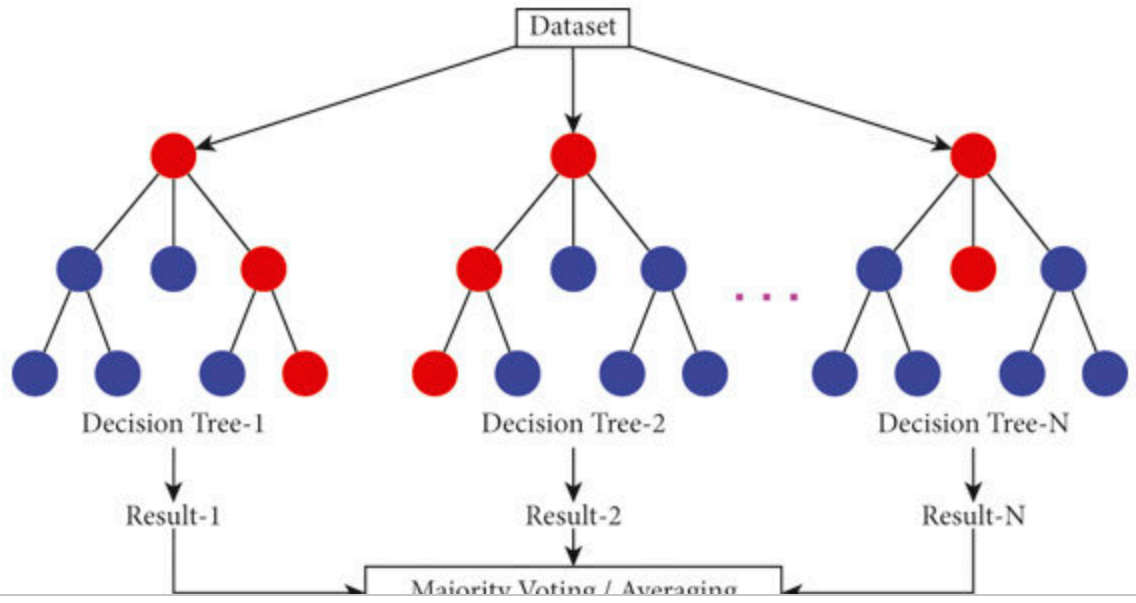



```
fig = plt.figure(figsize=(15,12))
tree.plot_tree(dt , filled=True)
```

```
[Text(0.5, 0.875, 'x[5] <= -0.056\ngini = 0.498\nsamples = 100778\nvalue = [53921, 46857]'),
 Text(0.25, 0.625, 'x[65] <= 0.5\ngini = 0.149\nsamples = 46582\nvalue = [3781, 42801]'),
 Text(0.375, 0.75, 'True  '),
 Text(0.125, 0.375, 'x[6] <= 0.007\ngini = 0.061\nsamples = 43109\nvalue = [1357, 41752]'),
 Text(0.0625, 0.125, 'gini = 0.031\nsamples = 42264\nvalue = [669.0, 41595.0]'),
 Text(0.1875, 0.125, 'gini = 0.303\nsamples = 845\nvalue = [688, 157]'),
 Text(0.375, 0.375, 'x[30] <= -0.211\ngini = 0.422\nsamples = 3473\nvalue = [2424, 1049]'),
 Text(0.3125, 0.125, 'gini = 0.025\nsamples = 1027\nvalue = [13, 1014]'),
 Text(0.4375, 0.125, 'gini = 0.028\nsamples = 2446\nvalue = [2411, 35]'),
 Text(0.75, 0.625, 'x[56] <= 0.5\ngini = 0.138\nsamples = 54196\nvalue = [50140, 4056]'),
 Text(0.625, 0.75, '  False'),
 Text(0.625, 0.375, 'x[9] <= 0.5\ngini = 0.067\nsamples = 51782\nvalue = [49996.0, 1786.0]'),
 Text(0.5625, 0.125, 'gini = 0.027\nsamples = 49675\nvalue = [49007, 668]'),
 Text(0.6875, 0.125, 'gini = 0.498\nsamples = 2107\nvalue = [989, 1118]'),
 Text(0.875, 0.375, 'x[5] <= 0.899\ngini = 0.112\nsamples = 2414\nvalue = [144, 2270]'),
 Text(0.8125, 0.125, 'gini = 0.0\nsamples = 144\nvalue = [144, 0]'),
 Text(0.9375, 0.125, 'gini = 0.0\nsamples = 2270\nvalue = [0, 2270]')]
```



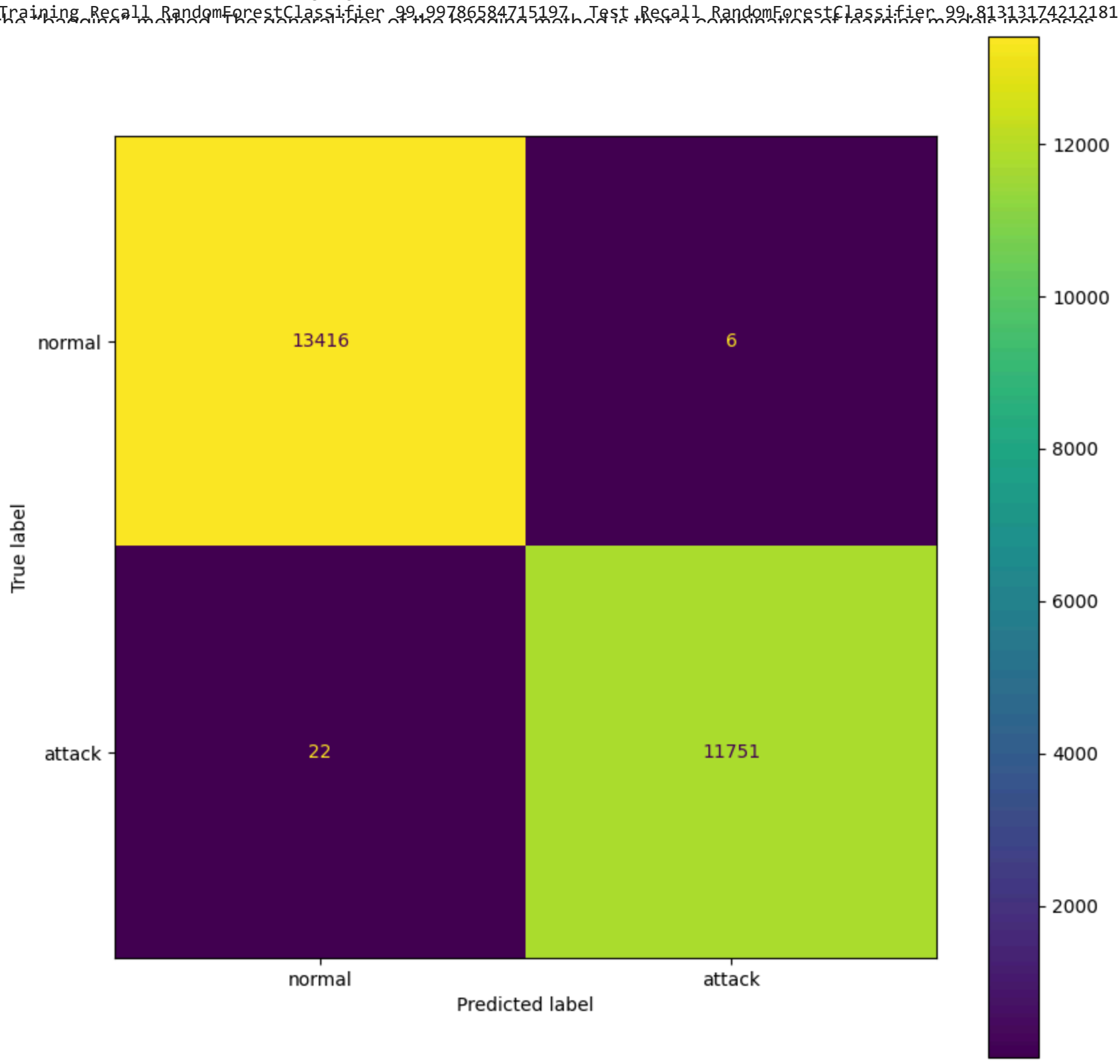
Random forest



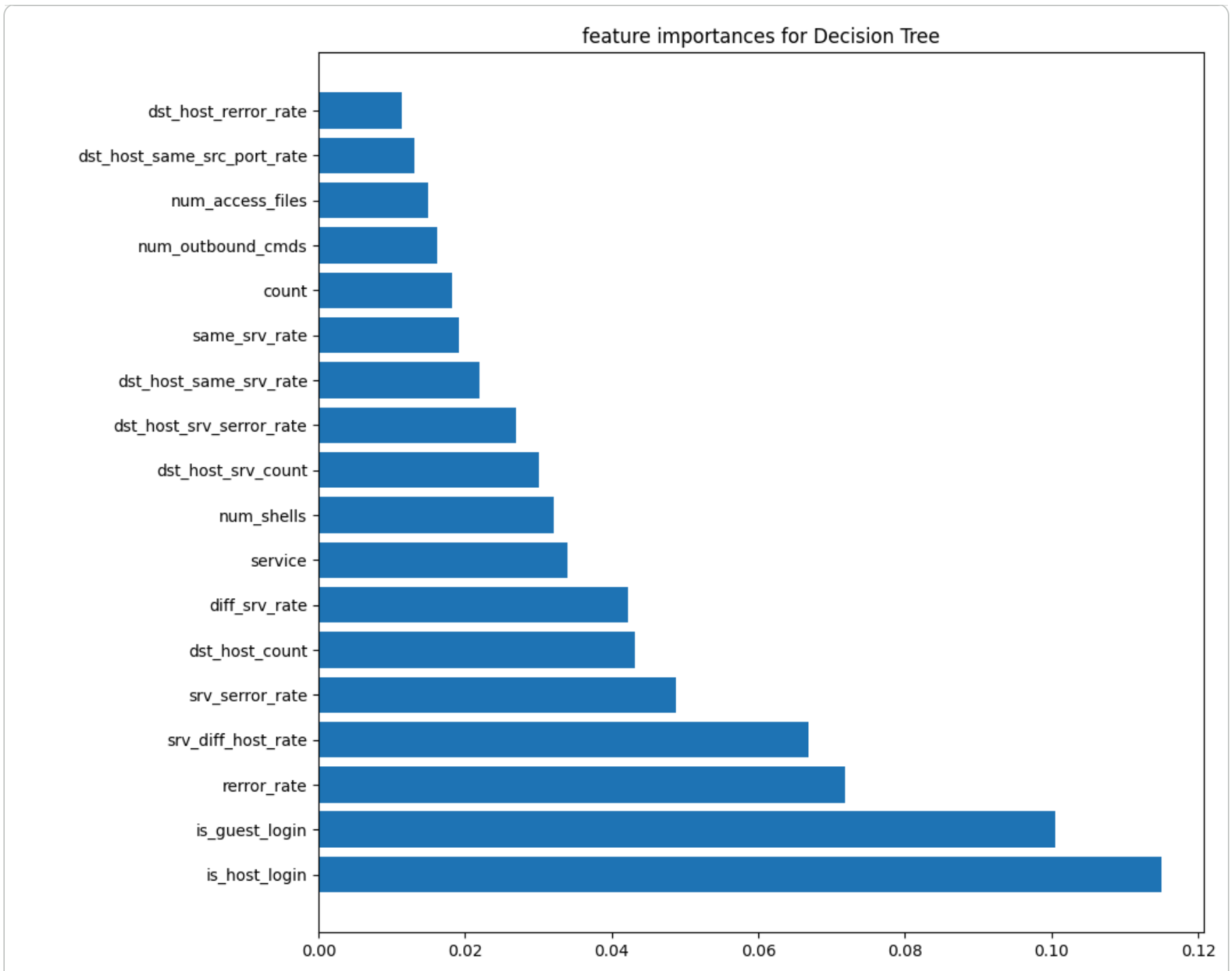

```
rf = RandomForestClassifier().fit(x_train, y_train)
evaluate_classification(rf, "RandomForestClassifier", x_train, x_test, y_train, y_test)
```

```
Training Accuracy RandomForestClassifier 99.99801543987775  Test Accuracy RandomForestClassifier 99.88886683865
Training Precision RandomForestClassifier 99.99658471971  Test Precision RandomForestClassifier 99.84806573
Training Recall RandomForestClassifier 99.99786584715197  Test Recall RandomForestClassifier 99.81313174212181
```

Random forest is a supervised learning algorithm. The "forest" it builds is an ensemble of decision trees, usually trained with the "bagging" method. The general idea of the bagging method is that a combination of learning models increases the o probl trees




```
f_importances(abs(rf.feature_importances_), features_names, top=18)
```



Building an XGBOOST REgressor regressor in order to predict threat level

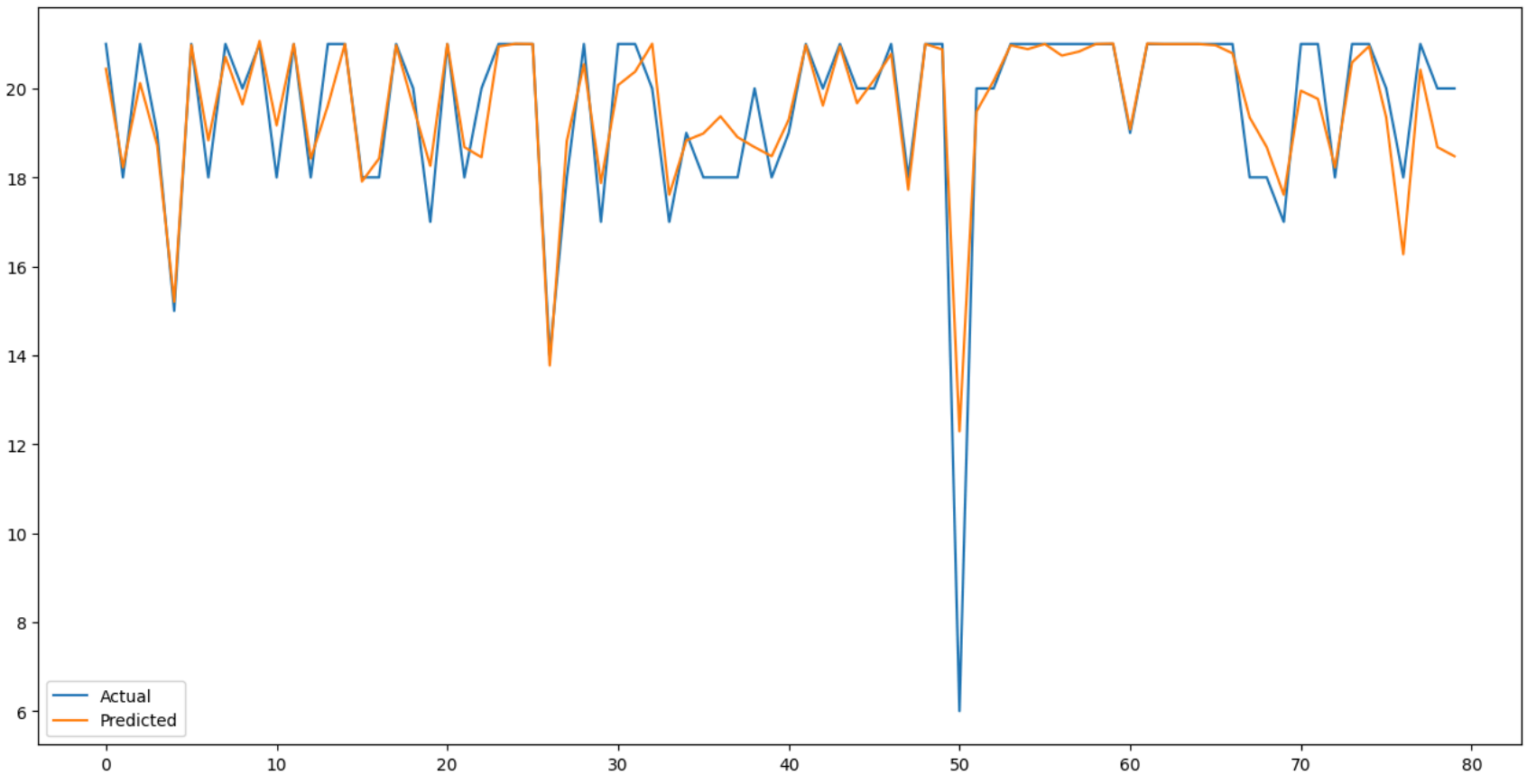
```
xg_r = xgb.XGBRegressor(objective ='reg:linear',n_estimators = 20).fit(x_train_reg, y_train_reg)
```

```
name = "XGBOOST"
train_error = np.sqrt(metrics.mean_squared_error(y_train_reg, xg_r.predict(x_train_reg)))
test_error = np.sqrt(metrics.mean_squared_error(y_test_reg, xg_r.predict(x_test_reg)))
print("Training Error " + str(name) + " {}  Test error ".format(train_error) + str(name) + " {}".format(test_e
```

```
Training Error XGBOOST 0.9228151344496821  Test error XGBOOST 1.038470278741186
```

```
y_pred = xg_r.predict(x_test_reg)
df = pd.DataFrame({"Y_test": y_test_reg , "Y_pred" : y_pred})
plt.figure(figsize=(16,8))
plt.plot(df[:80])
plt.legend(['Actual' , 'Predicted'])
```

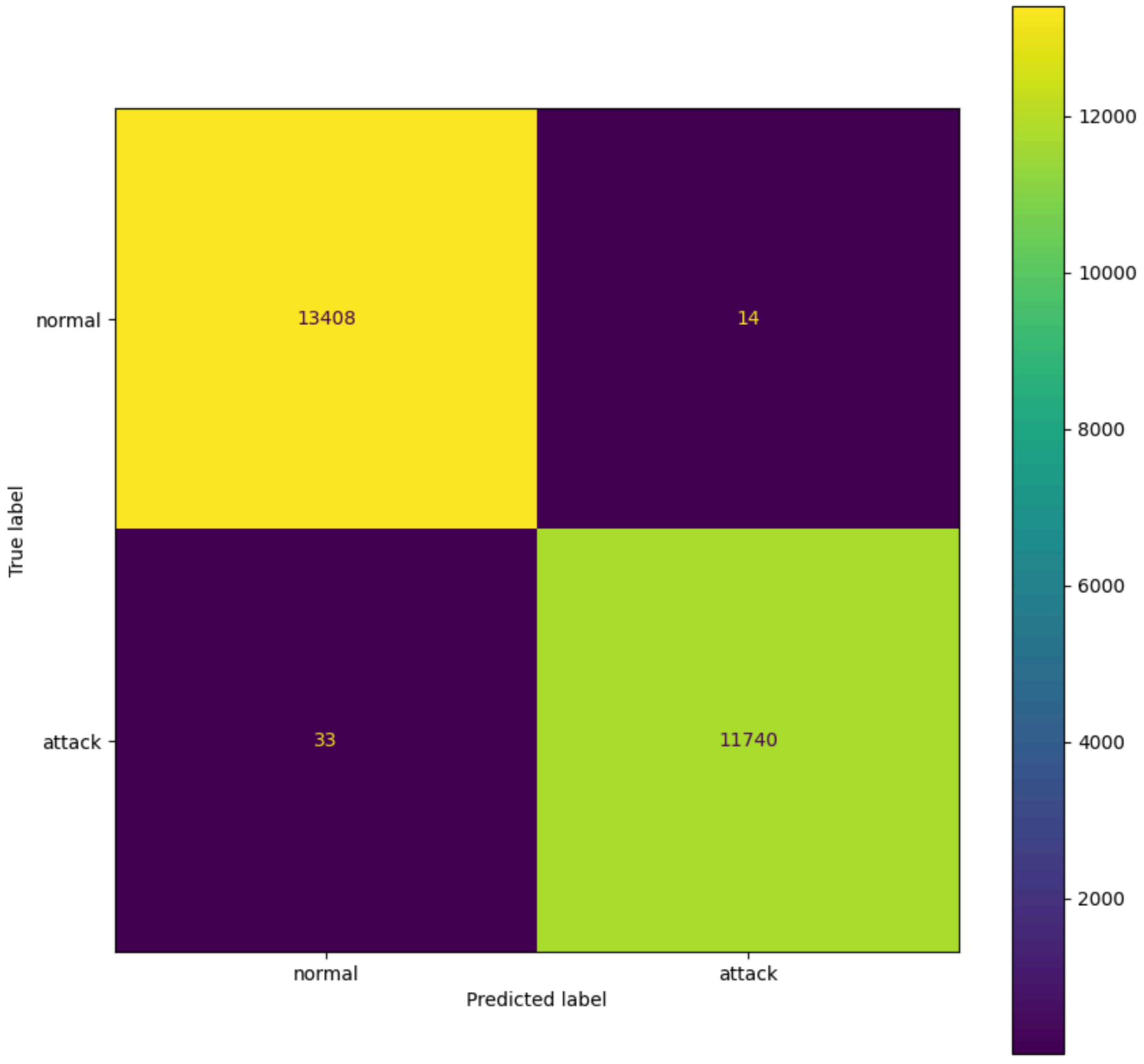
```
<matplotlib.legend.Legend at 0x7a21611f4a40>
```



Measuring effect of PCA

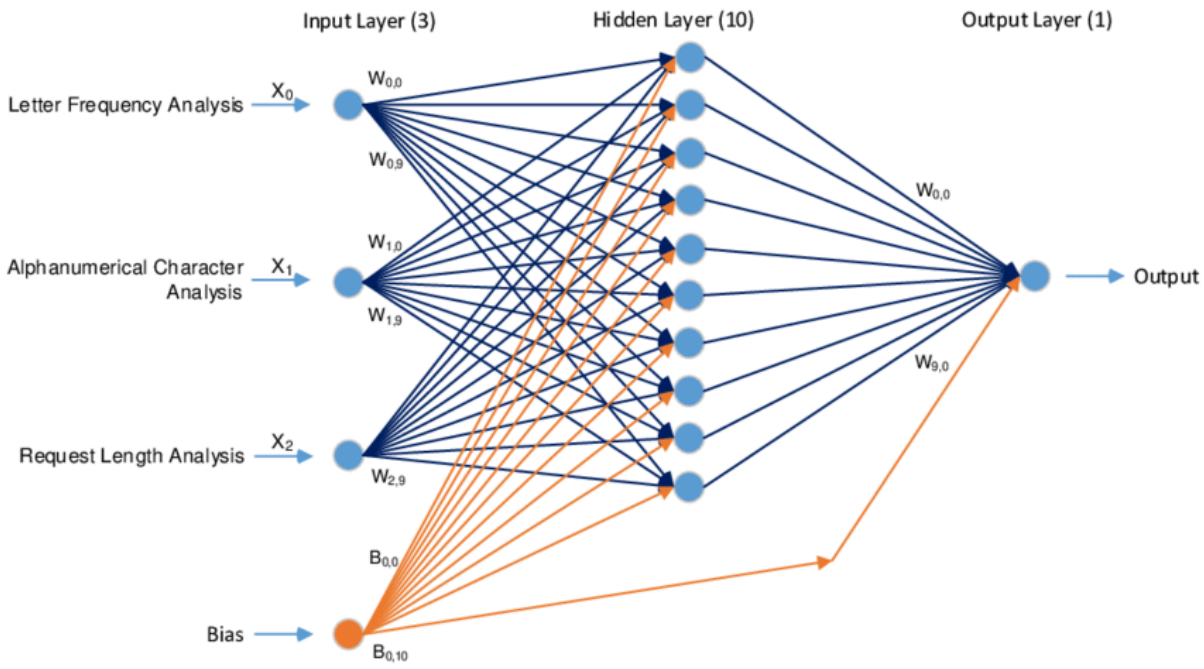
```
rrf = RandomForestClassifier().fit(x_train_reduced, y_train_reduced)
evaluate_classification(rrf, "PCA RandomForest", x_train_reduced, x_test_reduced, y_train_reduced, y_test_redu
```

```
Training Accuracy PCA RandomForest 99.99801543987775  Test Accuracy PCA RandomForest 99.81345505060528
Training Precesion PCA RandomForest 99.99786584715197  Test Precesion PCA RandomForest 99.88089161136634
Training Recall PCA RandomForest 99.99786584715197  Test Recall PCA RandomForest 99.7196976131827
```



Neural networks

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another. Artificial neural networks (ANNs) are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.



Neural networks rely on training data to learn and improve their accuracy over time. However, once these learning algorithms are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high velocity. Tasks in speech recognition or image recognition can take minutes versus

hours when compared to the manual identification by human experts. One of the most well-known neural networks is Google's search algorithm.

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=64, activation='relu', input_shape=(x_train.shape[1:]),
                          kernel_regularizer=regularizers.L1L2(l1=1e-5, l2=1e-4),
                          bias_regularizer=regularizers.L2(1e-4),
                          activity_regularizer=regularizers.L2(1e-5)),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(units=128, activation='relu',
                          kernel_regularizer=regularizers.L1L2(l1=1e-5, l2=1e-4),
                          bias_regularizer=regularizers.L2(1e-4),
                          activity_regularizer=regularizers.L2(1e-5)),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(units=512, activation='relu',
                          kernel_regularizer=regularizers.L1L2(l1=1e-5, l2=1e-4),
                          bias_regularizer=regularizers.L2(1e-4),
                          activity_regularizer=regularizers.L2(1e-5)),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(units=128, activation='relu',
                          kernel_regularizer=regularizers.L1L2(l1=1e-5, l2=1e-4),
                          bias_regularizer=regularizers.L2(1e-4),
                          activity_regularizer=regularizers.L2(1e-5)),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(units=1, activation='sigmoid'),
])
```

```
model.compile(optimizer='adam', loss=tf.keras.losses.BinaryCrossentropy(from_logits=True), metrics=['accuracy'
```

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	7,872
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 128)	8,320
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 512)	66,048
dropout_2 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 128)	65,664
dropout_3 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 1)	129

Total params: 148,033 (578.25 KB)
Trainable params: 148,033 (578.25 KB)
Non-trainable params: 0 (0.00 B)

```
!pip install pydot graphviz
from tensorflow.keras.utils import plot_model
plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```