Analysis of Intrusion detection system (IDS) by Machine learning

Presented by : Pratik Kage
Github link :

Abstract

The world has experienced a radical change due to the internet. As a matter of fact, it assists people in maintaining their social networks and links them to other members of their social networks when they require assistance. In effect sharing professional and personal data comes with several risks to individuals and organizations. Internet became a crucial element in our daily life, therefore, the security of our DATA could be threatened at any time. For this reason, Intrusion detection system plays a major role in protecting internet users against any malicious network attacks. (IDS) Intrusion Detection System is a system that monitors network traffic for suspicious activity and issues alerts when such activity is discovered. In this paper, the focus will be on two different classifications; starting by machine learning, algorithms Logistic regression, SVM, KNN, Guassian naive bayes, Decision tree, Random forest, XGboost, Neural networks. These algorithms will be used to define the best accuracy by means of the training dataset. Based on the result of the training dataset, the testing dataset is used to process our database with the most efficient algorithm. Two different datasets will be operated in our experiments to evaluate the model performance. KDD train, KDD test datasets are used to measure the performance of the proposed approach in order to guarantee its efficiency

Introduction

The number of computing devices has grown at a rapid rate. Laptops and desktop computers, as well as smartphones and tablets, have become nearly vital tools in everyday life, and many people use them on a regular basis. The main issue comes here the data which we get through internet has to be secured; This security of data over network is done by Intrusion Detection System (IDS). An intrusion detection system (IDS) is a software application or device that monitors system or network activity for policy violations or malicious behavior, and generates reports for the management system. The need for an intrusion detection system is undeniable; thus, an accurate model must be developed. In this field, machine learning has proven to be an effective investigation device that can detect any irregular event taking place in any system's traffic. To build a good IDS it well be able to detect malicious traffics with a high efficacy; the accuracy of algorithms of classification well decide that efficacy. In this work, we propose an IDS approach for detecting malicious network traffic with more efficiency and higher accuracy at the first a presentation of our DATASET that will be trained by 3 different algorithms of classification; the next section represents ours second DATASET but this time it well be trained by the higher accuracy of the tree algorithms bellows; the last section is a conclusion as well as some issues which have been highlighted for future research.

Problem Statement

Intrusions/attacks are a set of events that can compromise the principles of a computer network such as integrity, availability and confidentiality. These cyber-attacks are in headline almost every day as the use of Computers, Mobile devices, and IoT devices have become an integral part of our lives. Modern attacks' environment cannot be detected by firewalls so that NIDS is designed to achieve high protection from cyber attacks.

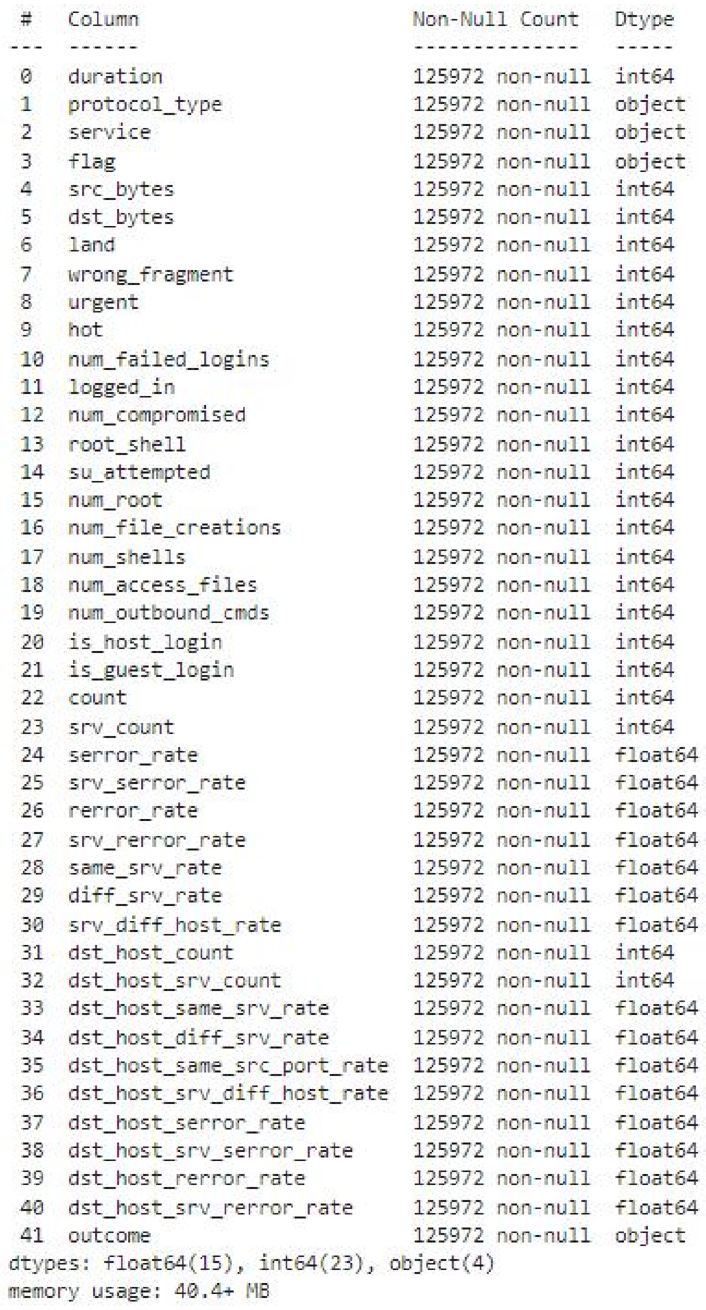
A Network Intrusion Detection System (NIDS) monitors network traffic flow to detect malicious activity. NIDS is classified basically into two types viz. Signature based and Anomaly based detection system. Signature-based IDS detects possible threats by looking for specific patterns, such as byte sequences in network traffic, or known malicious instruction sequences used by malware. Although signature-based IDS can easily detect known attacks, it is impossible to detect new attacks, for which no pattern is available. Anomaly based detection system is designed to detect unknown attacks. This detection method uses machine learning to create a defined model of trustworthy activity, and then compare new behavior against this model. Anomaly based detection systems are used nowadays due to its superiority in detecting unknown attacks.

Machine learning problem statement

Our problem is to detect malicious traffic from incoming traffic. So we will be classifying given incoming network traffic datapoint as normal or malicious. Since we will be classifying incoming network traffic into two categories, this will be a binary classification machine learning problem.

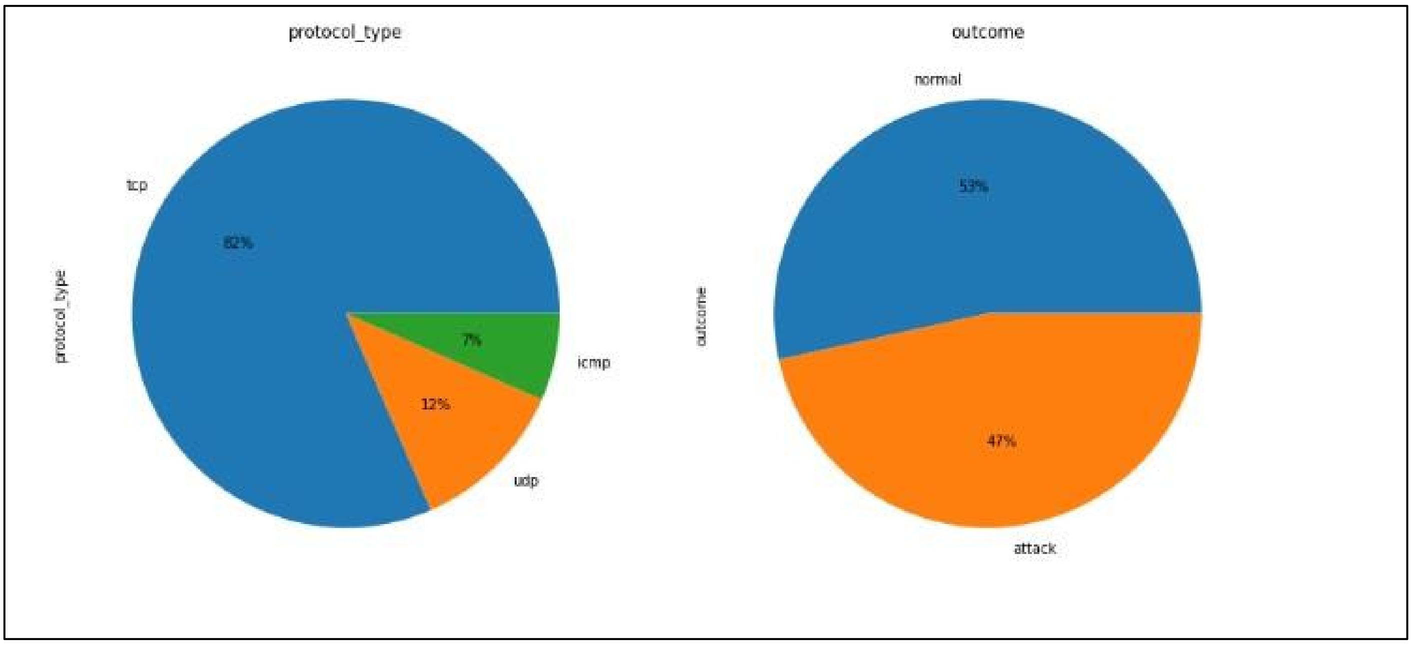
❖ Data-set Description

Many datasets are publicly available online for research purposes. According to an examination of the literature, some of them were created decades ago and may not be very useful in detecting recent threats. Following are the features of the dataset and last column “Outcome” is the label (Target) for the problem statement :



```
 #   Column                       Non-Null Count   Dtype  
---  ------                       --------------   -----  
 0   duration                     125972 non-null  int64  
 1   protocol_type                125972 non-null  object 
 2   service                      125972 non-null  object 
 3   flag                         125972 non-null  object 
 4   src_bytes                    125972 non-null  int64  
 5   dst_bytes                    125972 non-null  int64  
 6   land                         125972 non-null  int64  
 7   wrong_fragment               125972 non-null  int64  
 8   urgent                       125972 non-null  int64  
 9   hot                          125972 non-null  int64  
 10  num_failed_logins            125972 non-null  int64  
 11  logged_in                    125972 non-null  int64  
 12  num_compromised              125972 non-null  int64  
 13  root_shell                   125972 non-null  int64  
 14  su_attempted                 125972 non-null  int64  
 15  num_root                     125972 non-null  int64  
 16  num_file_creations           125972 non-null  int64  
 17  num_shells                   125972 non-null  int64  
 18  num_access_files             125972 non-null  int64  
 19  num_outbound_cmds            125972 non-null  int64  
 20  is_host_login                125972 non-null  int64  
 21  is_guest_login               125972 non-null  int64  
 22  count                        125972 non-null  int64  
 23  srv_count                    125972 non-null  int64  
 24  serror_rate                  125972 non-null  float64
 25  srv_serror_rate              125972 non-null  float64
 26  rerror_rate                  125972 non-null  float64
 27  srv_rerror_rate              125972 non-null  float64
 28  same_srv_rate                125972 non-null  float64
 29  diff_srv_rate                125972 non-null  float64
 30  srv_diff_host_rate           125972 non-null  float64
 31  dst_host_count               125972 non-null  int64  
 32  dst_host_srv_count           125972 non-null  int64  
 33  dst_host_same_srv_rate       125972 non-null  float64
 34  dst_host_diff_srv_rate       125972 non-null  float64
 35  dst_host_same_src_port_rate  125972 non-null  float64
 36  dst_host_srv_diff_host_rate  125972 non-null  float64
 37  dst_host_serror_rate         125972 non-null  float64
 38  dst_host_srv_serror_rate     125972 non-null  float64
 39  dst_host_rerror_rate         125972 non-null  float64
 40  dst_host_srv_rerror_rate     125972 non-null  float64
 41  outcome                      125972 non-null  object 
dtypes: float64(15), int64(23), object(4)
memory usage: 40.4+ MB
```

❖ Distribution of protocol_type and Outcome



❖ **Protocol Type Distribution** :

TCP (Transmission Control Protocol): 82% of the data instances are using the TCP protocol.
UDP (User Datagram Protocol): 12% of the data instances are associated with UDP.
ICMP (Internet Control Message Protocol): 7% of the data instances are related to the ICMP protocol.

❖ **Outcome Distribution** :

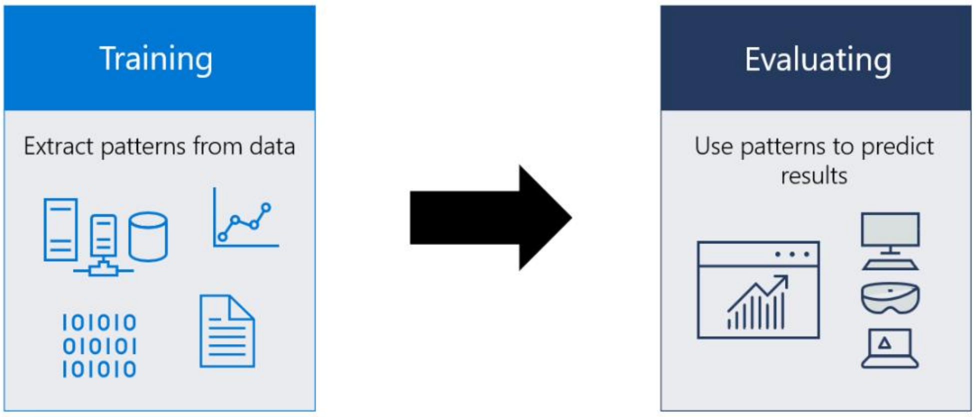
Normal: 53% of the instances in the dataset are classified as normal traffic.

Attack: 47% of the instances are classified as attacks.

The data distribution here likely comes from a network intrusion detection system (IDS). The traffic is mostly comprised of TCP packets (82%), followed by UDP (12%) and ICMP (7%). In terms of the outcome, the dataset shows a balanced distribution between normal traffic (53%) and attack traffic (47%), emphasizing the relevance of detecting threats within the network.

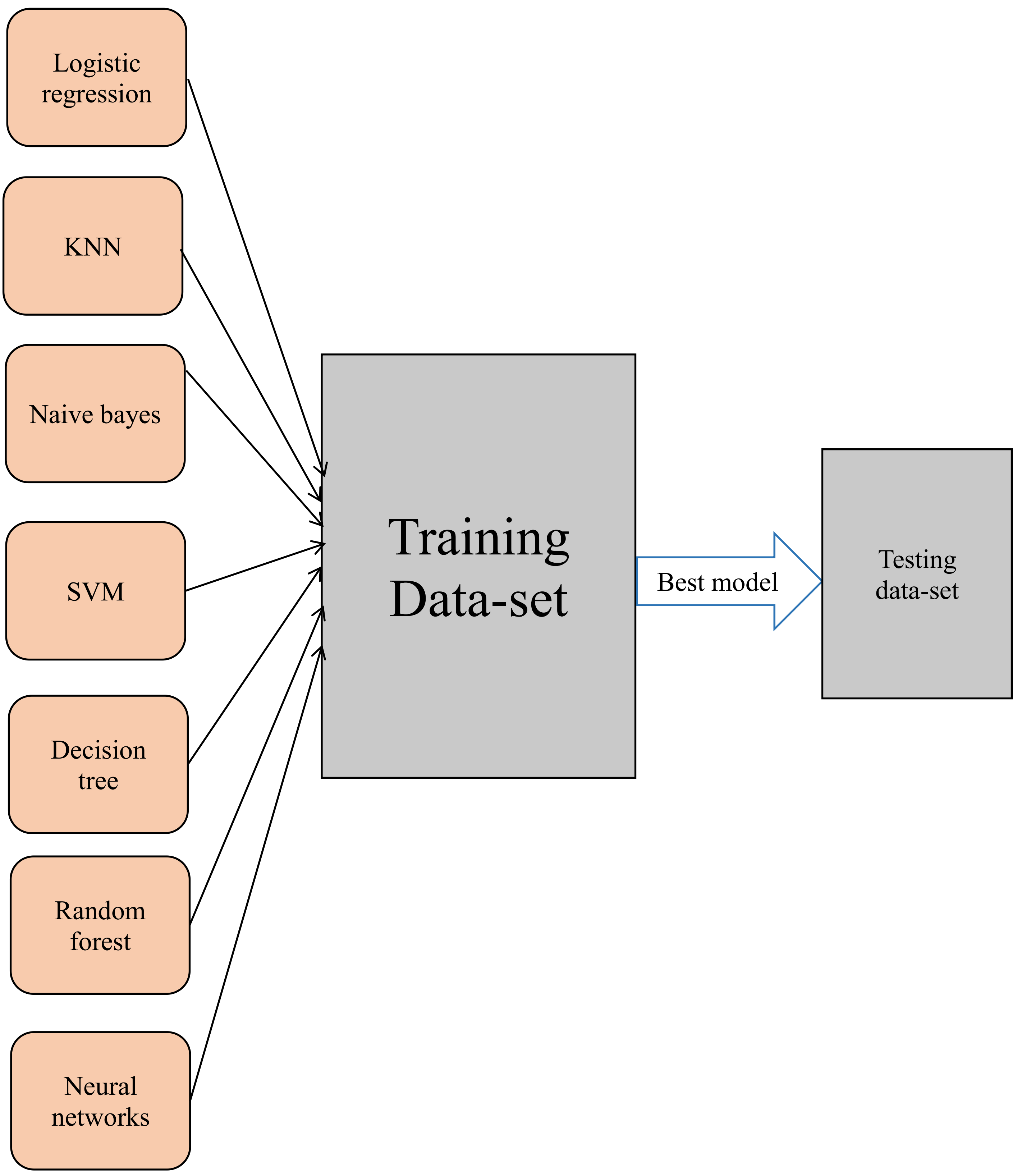
Machine learning Modelling

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.



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. For example, let's say you want to build an application that can recognize a user's emotions based on their facial expressions. You can train a model by providing it with images of faces that are each tagged with a certain emotion, and then you can use that model in an application that can recognize any user's emotion

Methodology



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:

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

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

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.

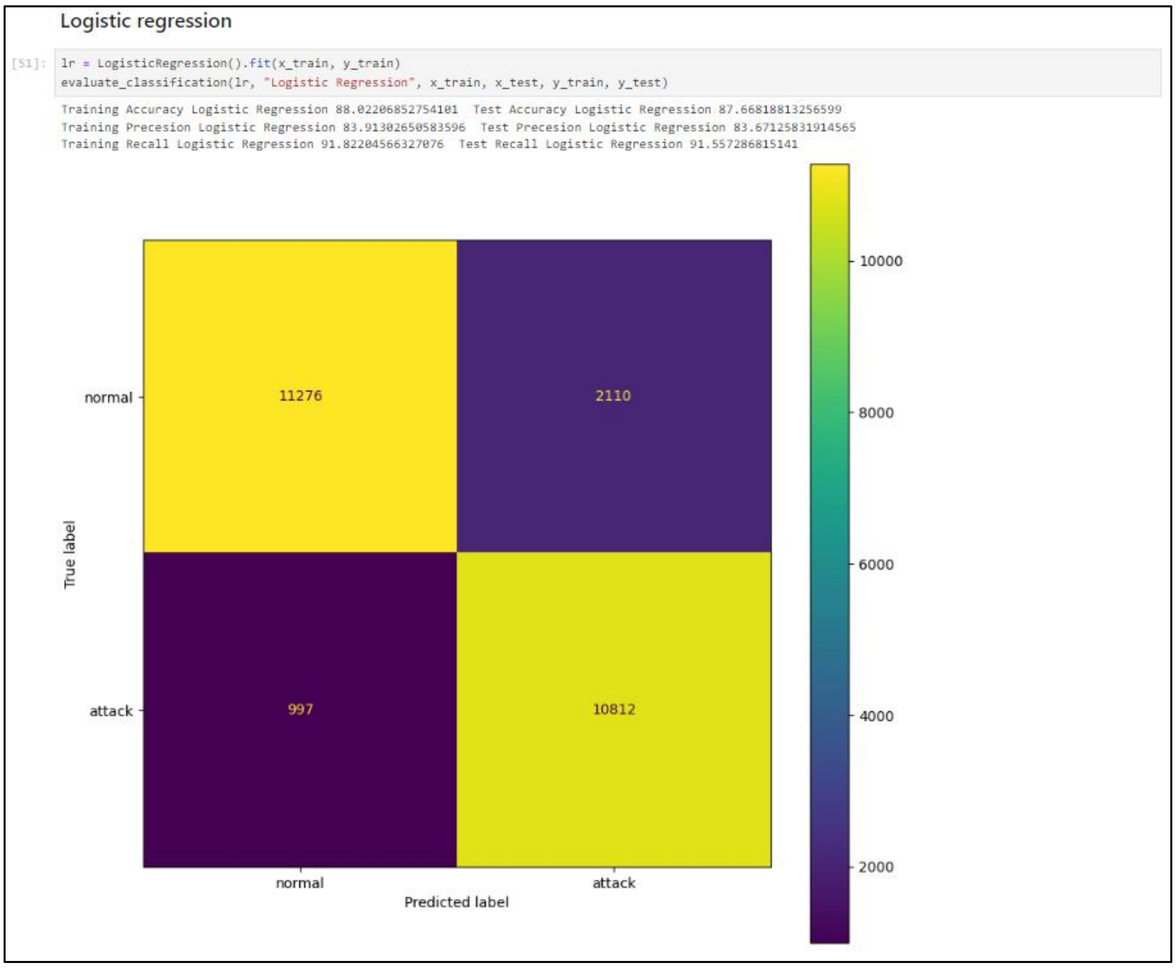
Binary logistic regression:

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.

- Confusion matrix for LR:



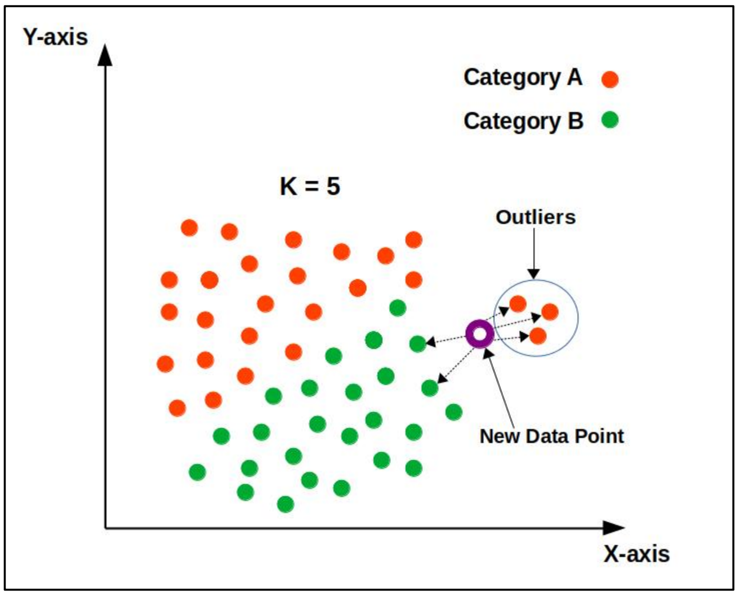
Logistic regression

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

```
Training Accuracy Logistic Regression 88.02206852754101  Test Accuracy Logistic Regression 87.66818813256599
Training Precesion Logistic Regression 83.91302650583596  Test Precesion Logistic Regression 83.67125831914565
Training Recall Logistic Regression 91.82204566327076  Test Recall Logistic Regression 91.557286815141
```

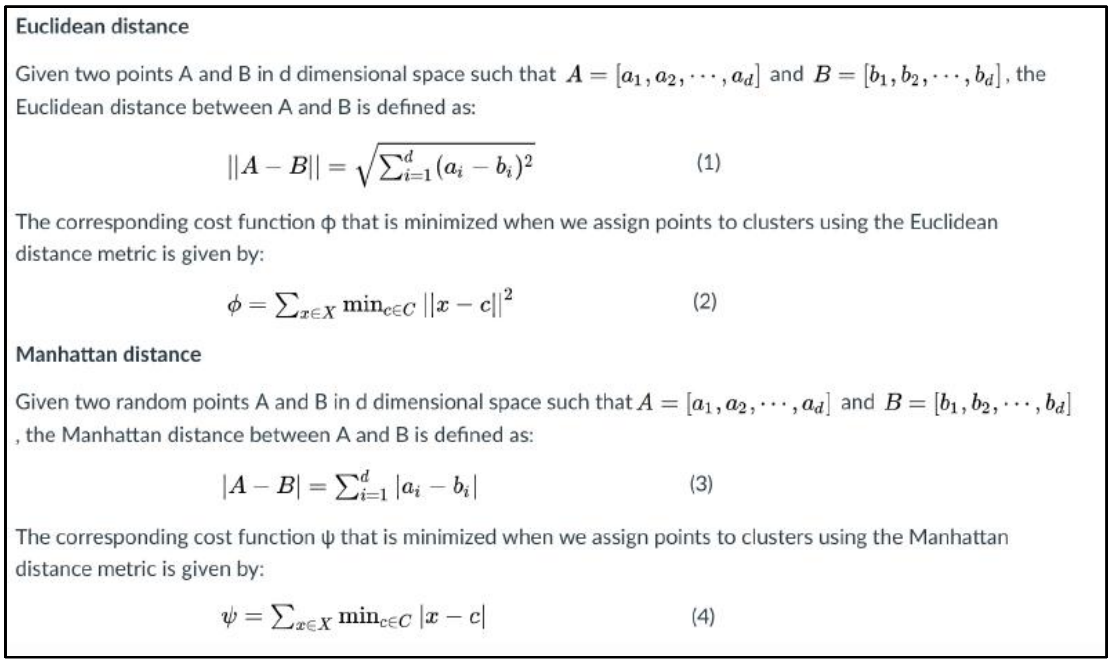
K-Nearest neighbours

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.



Euclidean distance

Given two points A and B in d dimensional space such that $A = [a_1, a_2, \cdots, a_d]$ and $B = [b_1, b_2, \cdots, b_d]$, the Euclidean distance between A and B is defined as:

$$||A - B|| = \sqrt{\sum_{i=1}^{d} (a_i - b_i)^2} \qquad (1)$$

The corresponding cost function ϕ that is minimized when we assign points to clusters using the Euclidean distance metric is given by:

$$\phi = \sum_{x \in X} \min_{c \in C} ||x - c||^2 \qquad (2)$$

Manhattan distance

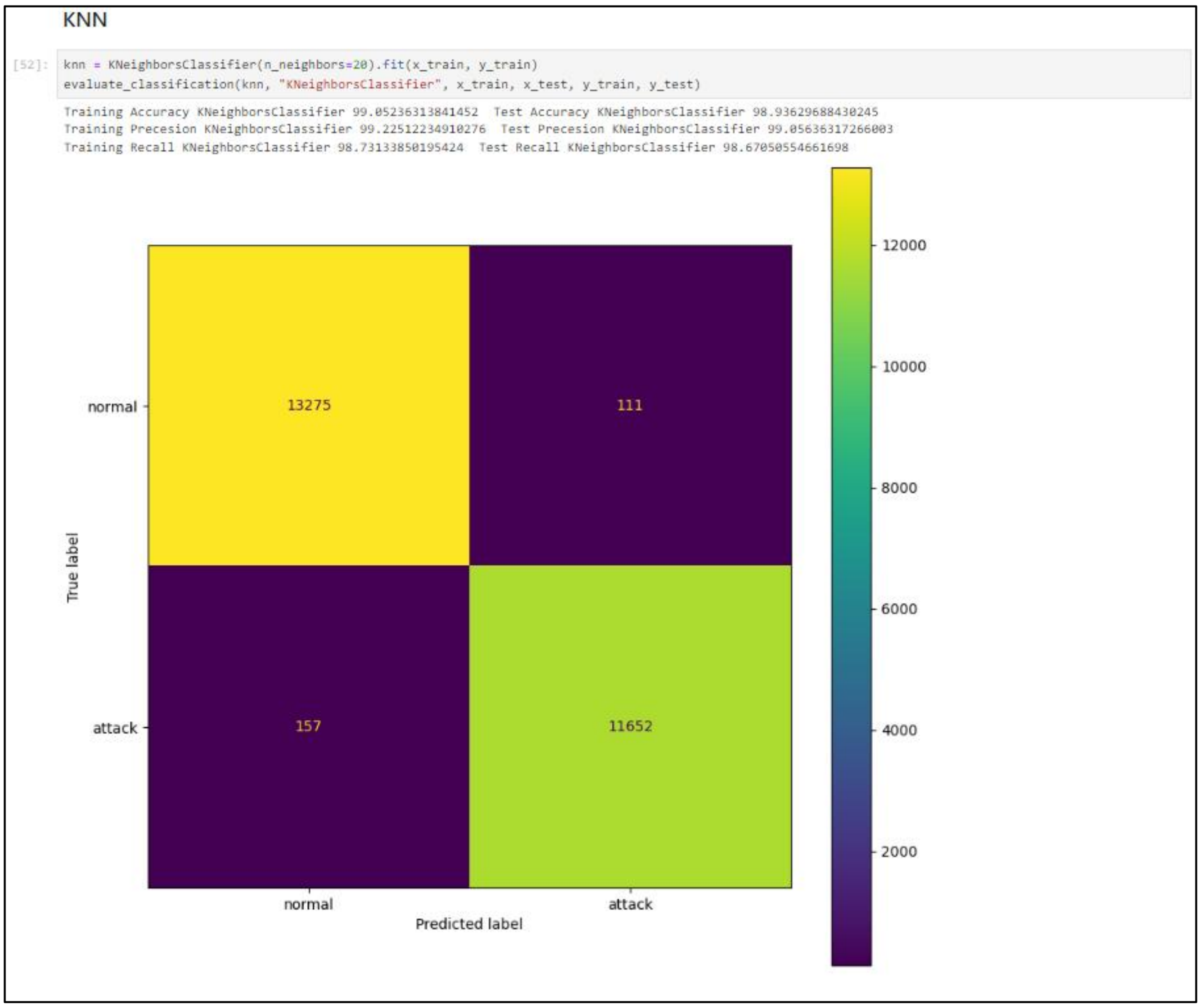
Given two random points A and B in d dimensional space such that $A = [a_1, a_2, \cdots, a_d]$ and $B = [b_1, b_2, \cdots, b_d]$, the Manhattan distance between A and B is defined as:

$$|A - B| = \sum_{i=1}^{d} |a_i - b_i| \qquad (3)$$

The corresponding cost function ψ that is minimized when we assign points to clusters using the Manhattan distance metric is given by:

$$\psi = \sum_{x \in X} \min_{c \in C} |x - c| \qquad (4)$$

Confusion matrix for KNN:



KNN

```
[52]: 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 99.05236313841452  Test Accuracy KNeighborsClassifier 98.93629688430245
Training Precesion KNeighborsClassifier 99.22512234910276  Test Precesion KNeighborsClassifier 99.05636317266003
Training Recall KNeighborsClassifier 98.73133850195424  Test Recall KNeighborsClassifier 98.67050554661698
```

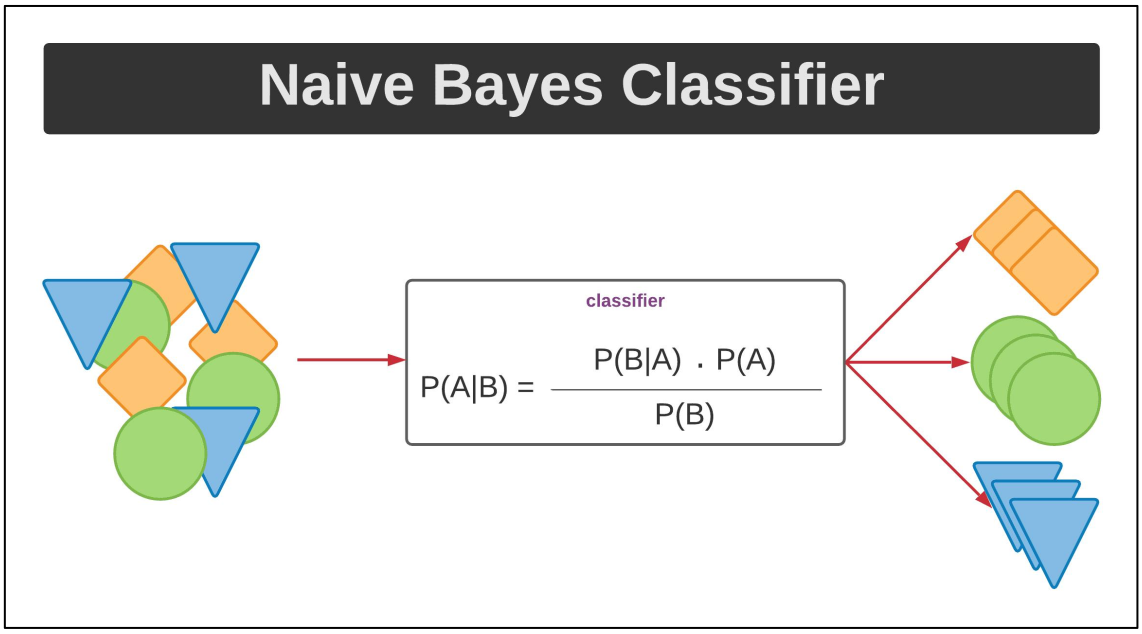
Naive Bayes¶

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:



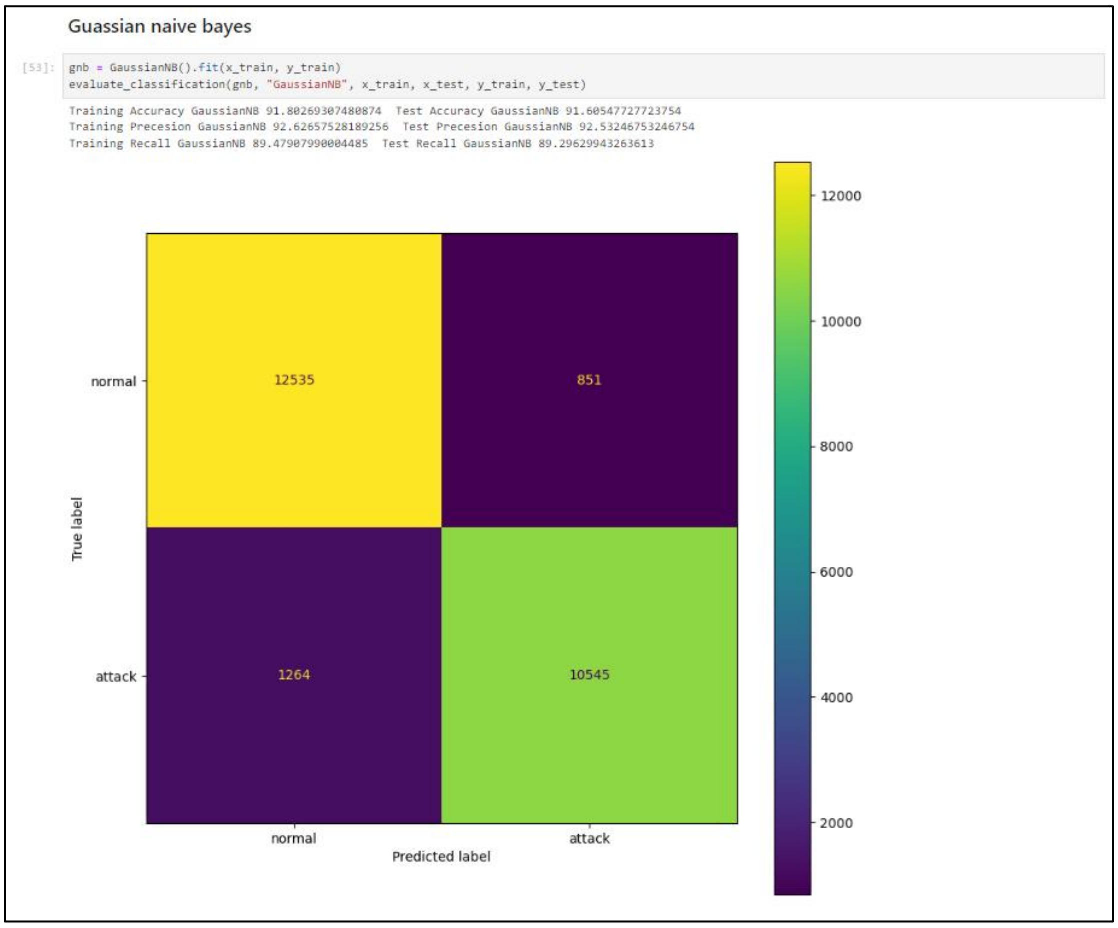
where A and B are events and $P(B) \neq 0$.

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.

Confusion Matrix for Naive bayes :



Guassian naive bayes

```
[53]: gnb = GaussianNB().fit(x_train, y_train)
      evaluate_classification(gnb, "GaussianNB", x_train, x_test, y_train, y_test)
```

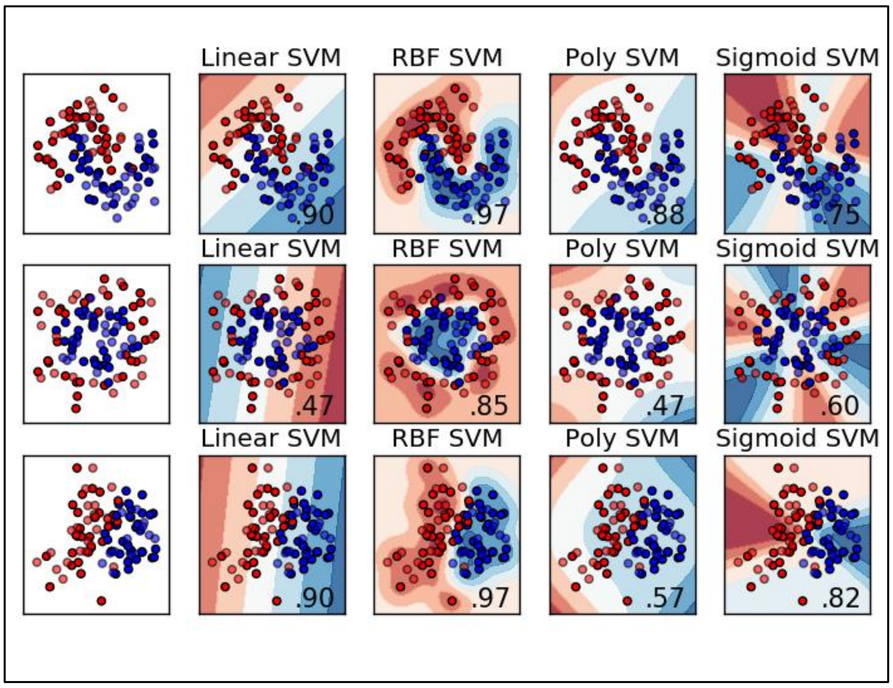
```
Training Accuracy GaussianNB 91.80269307480874  Test Accuracy GaussianNB 91.60547727723754
Training Precesion GaussianNB 92.62657528189256  Test Precesion GaussianNB 92.53246753246754
Training Recall GaussianNB 89.47907990004485  Test Recall GaussianNB 89.29629943263613
```

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.



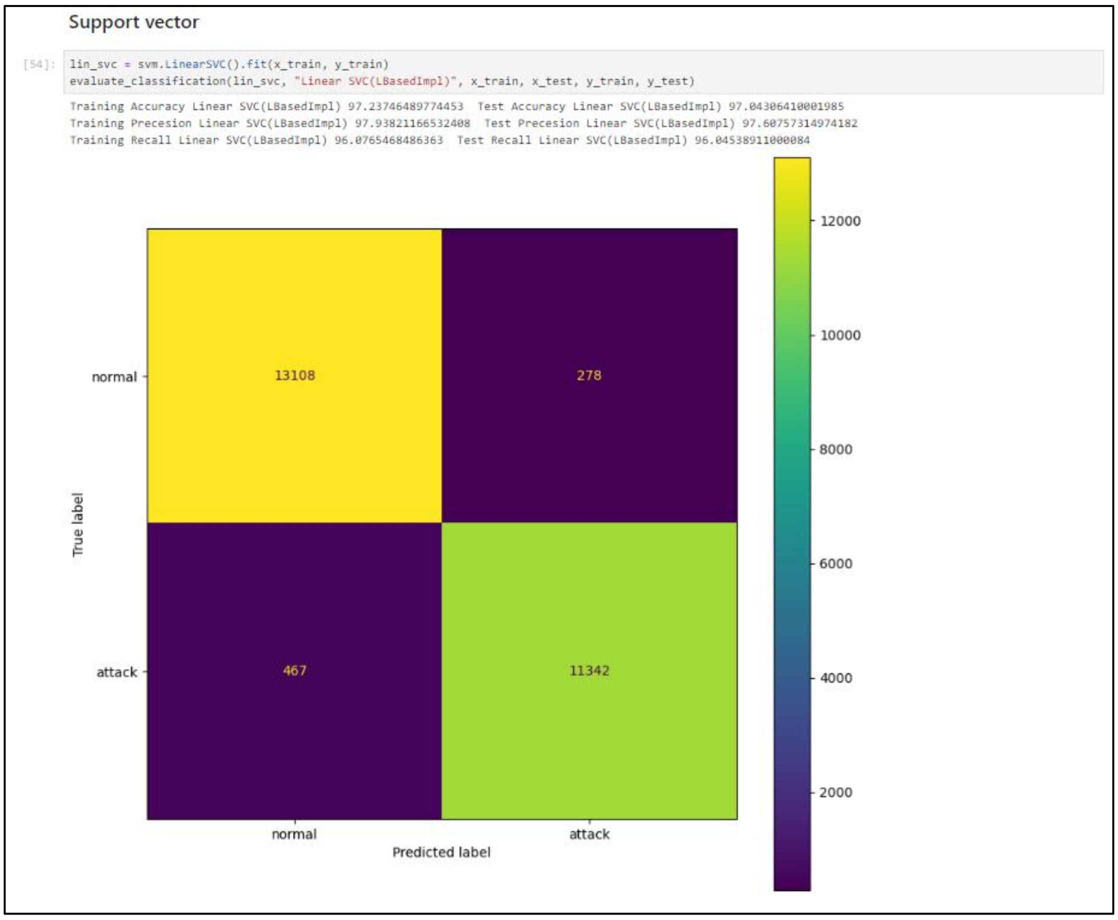
For example, in a class of fruits, to perform multi-class classification, we can create a binary classifier for each fruit. For say, the ‘mango’ class, there will be a binary classifier to predict if it IS a mango OR it is NOT a mango. The classifier with the highest score is chosen as the output of 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

Confusion matrix for SVM :

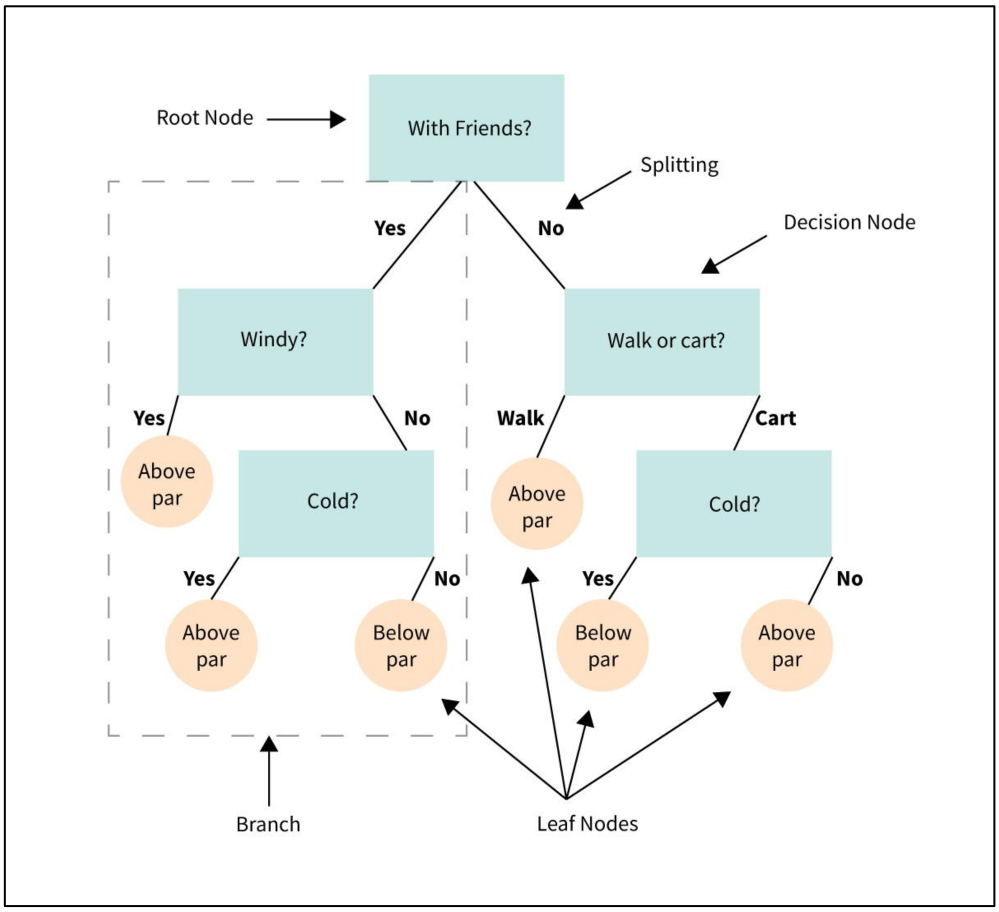

Support vector

```
[54]: 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) 97.23746489774453  Test Accuracy Linear SVC(LBasedImpl) 97.04306410001985
Training Precesion Linear SVC(LBasedImpl) 97.93821166532408  Test Precesion Linear SVC(LBasedImpl) 97.60757314974182
Training Recall Linear SVC(LBasedImpl) 96.0765468486363  Test Recall Linear SVC(LBasedImpl) 96.04538911000084
```

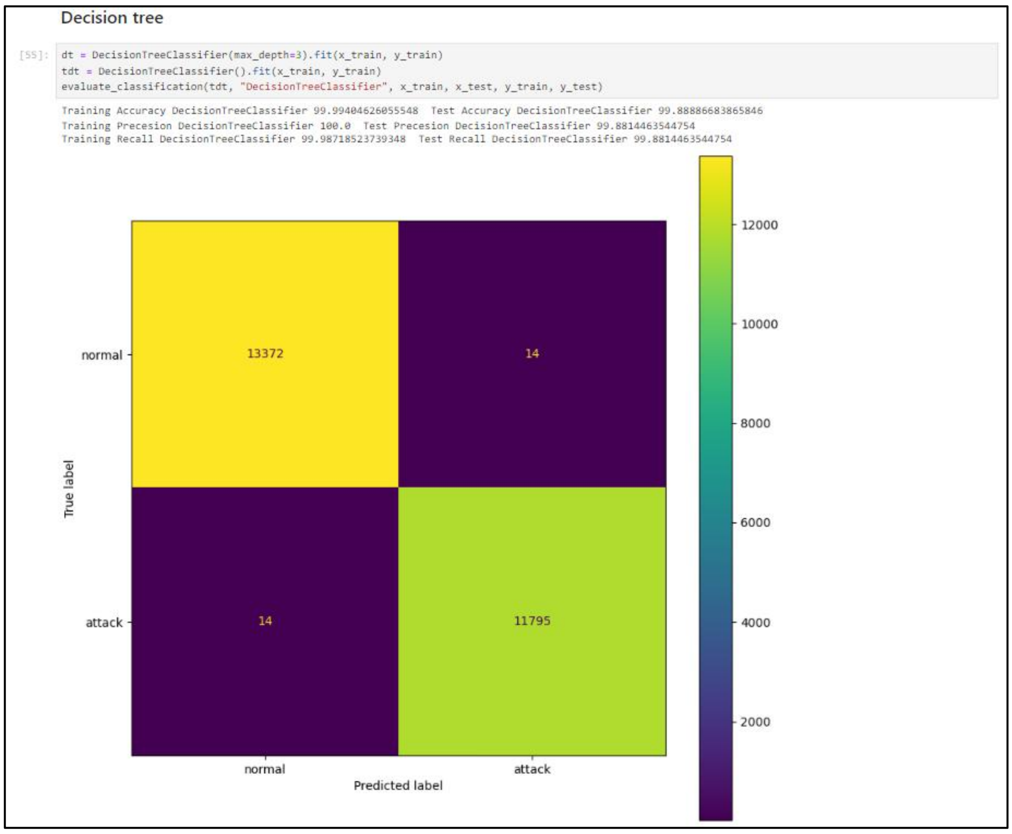
Decision Tree¶

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).

Decision tree confusion matrix :

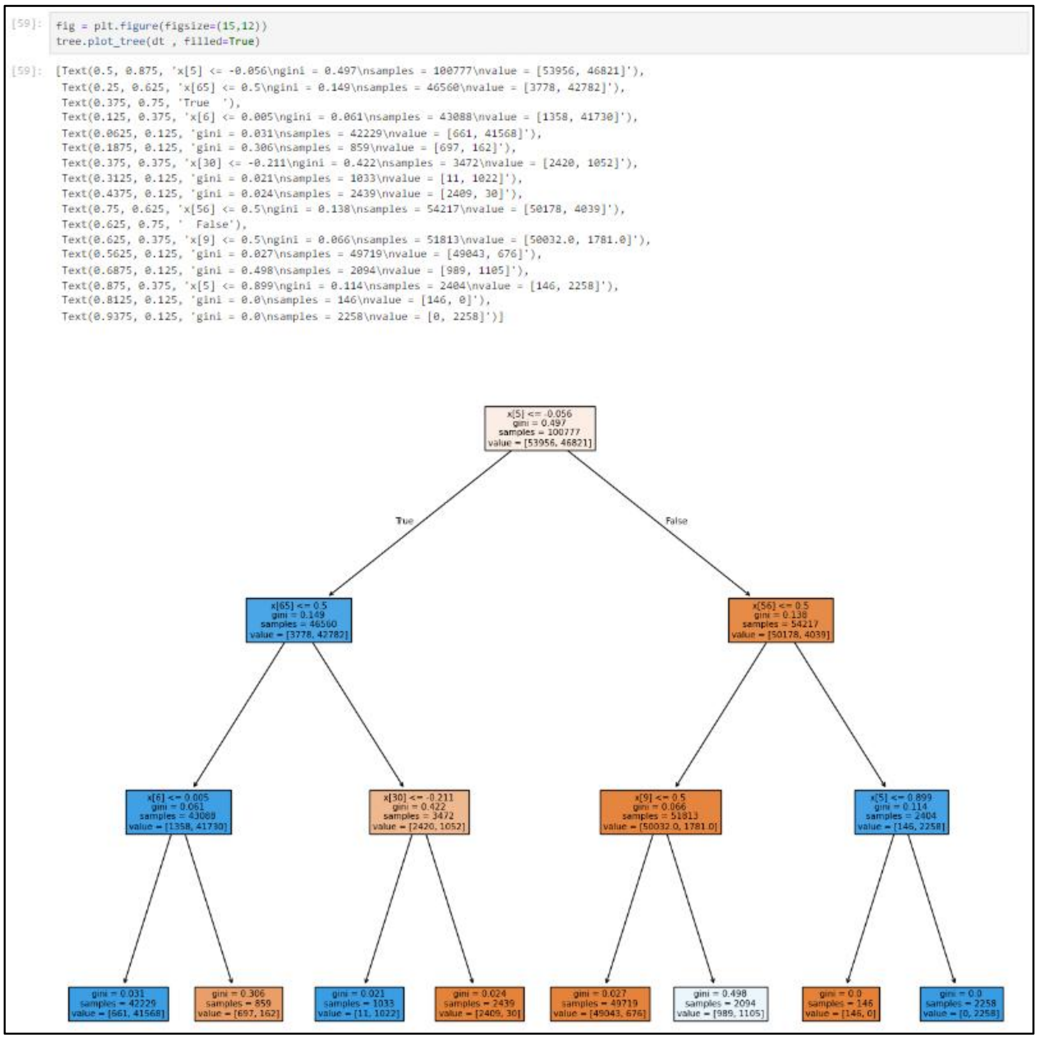


Decision tree

```
[55]: 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 99.99404626055548  Test Accuracy DecisionTreeClassifier 99.88886683865846
      Training Precesion DecisionTreeClassifier 100.0  Test Precesion DecisionTreeClassifier 99.8814463544754
      Training Recall DecisionTreeClassifier 99.98718523739348  Test Recall DecisionTreeClassifier 99.8814463544754
```

Tree plot for decision tree:



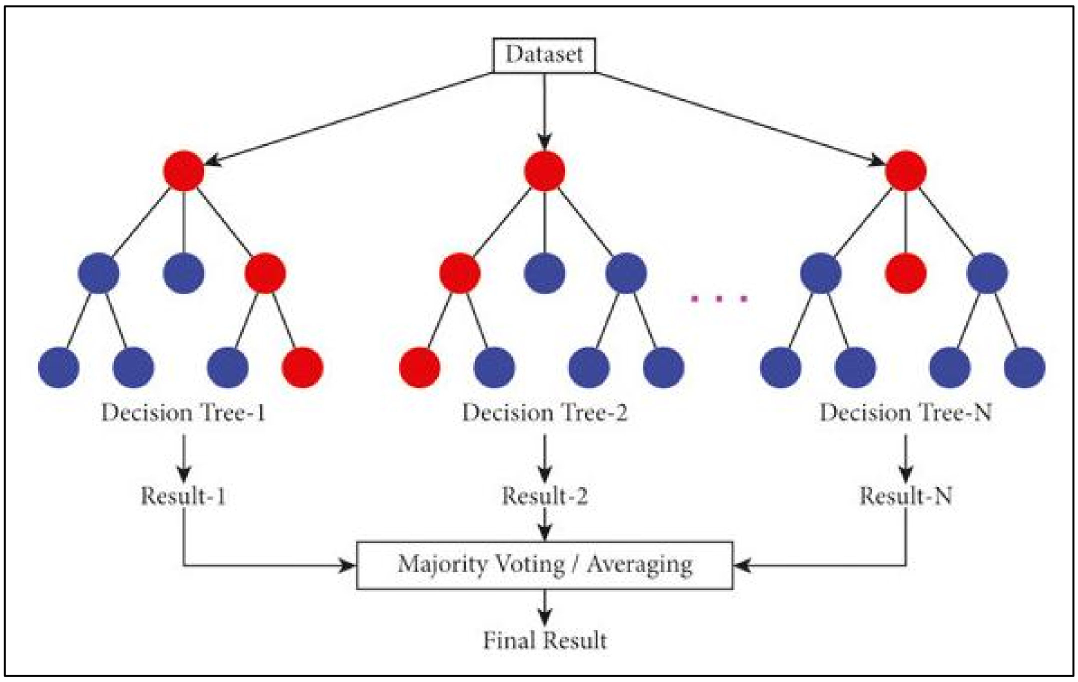
```
[59]: fig = plt.figure(figsize=(15,12))
      tree.plot_tree(dt , filled=True)

[59]: [Text(0.5, 0.875, 'x[5] <= -0.056\ngini = 0.497\nsamples = 100777\nvalue = [53956, 46821]'),
       Text(0.25, 0.625, 'x[65] <= 0.5\ngini = 0.149\nsamples = 46560\nvalue = [3778, 42782]'),
       Text(0.375, 0.75, 'True  '),
       Text(0.125, 0.375, 'x[6] <= 0.005\ngini = 0.061\nsamples = 43088\nvalue = [1358, 41730]'),
       Text(0.0625, 0.125, 'gini = 0.031\nsamples = 42229\nvalue = [661, 41568]'),
       Text(0.1875, 0.125, 'gini = 0.306\nsamples = 859\nvalue = [697, 162]'),
       Text(0.375, 0.375, 'x[30] <= -0.211\ngini = 0.422\nsamples = 3472\nvalue = [2420, 1052]'),
       Text(0.3125, 0.125, 'gini = 0.021\nsamples = 1033\nvalue = [11, 1022]'),
       Text(0.4375, 0.125, 'gini = 0.024\nsamples = 2439\nvalue = [2409, 30]'),
       Text(0.75, 0.625, 'x[56] <= 0.5\ngini = 0.138\nsamples = 54217\nvalue = [50178, 4039]'),
       Text(0.625, 0.75, '  False'),
       Text(0.625, 0.375, 'x[9] <= 0.5\ngini = 0.066\nsamples = 51813\nvalue = [50032.0, 1781.0]'),
       Text(0.5625, 0.125, 'gini = 0.027\nsamples = 49719\nvalue = [49043, 676]'),
       Text(0.6875, 0.125, 'gini = 0.498\nsamples = 2094\nvalue = [989, 1105]'),
       Text(0.875, 0.375, 'x[5] <= 0.899\ngini = 0.114\nsamples = 2404\nvalue = [146, 2258]'),
       Text(0.8125, 0.125, 'gini = 0.0\nsamples = 146\nvalue = [146, 0]'),
       Text(0.9375, 0.125, 'gini = 0.0\nsamples = 2258\nvalue = [0, 2258]')]
```

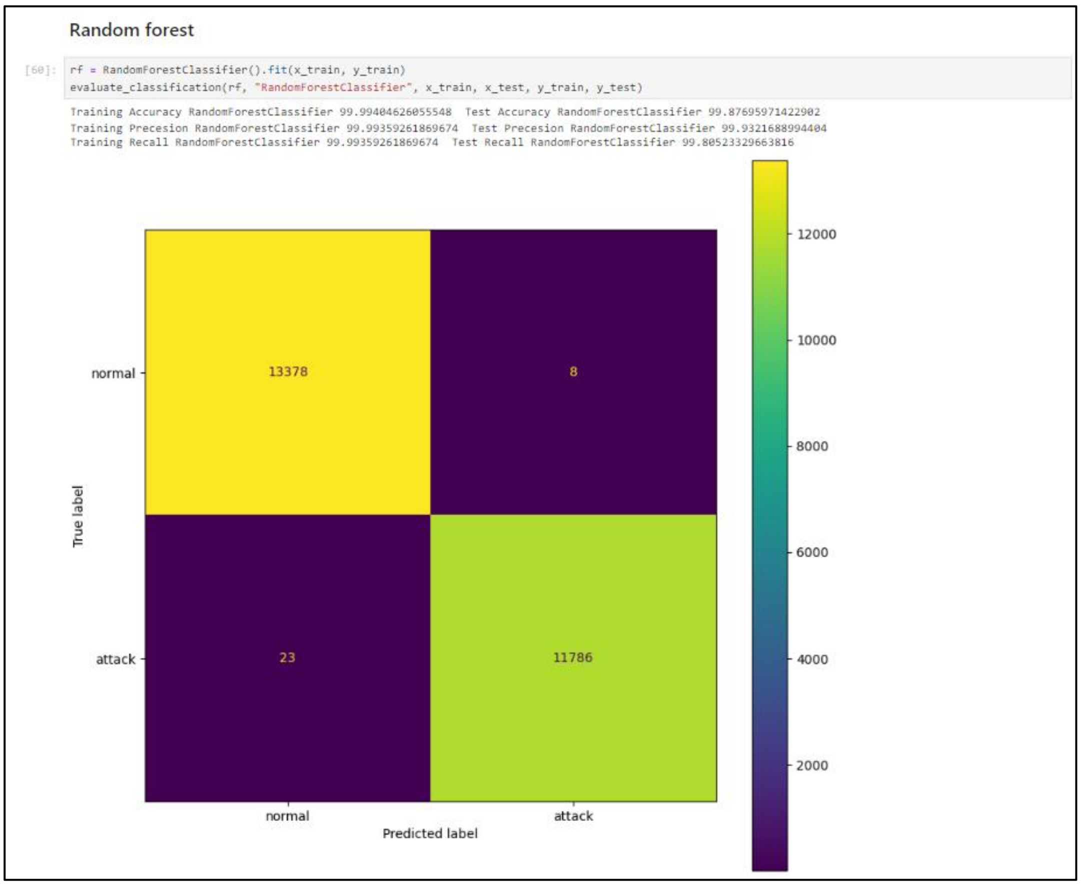
x[5] <= -0.056 gini = 0.497 samples = 100777 value = [53956, 46821]
True
False
x[65] <= 0.5 gini = 0.149 samples = 46560 value = [3778, 42782]
x[56] <= 0.5 gini = 0.138 samples = 54217 value = [50178, 4039]
x[6] <= 0.005 gini = 0.061 samples = 43088 value = [1358, 41730]
x[30] <= -0.211 gini = 0.422 samples = 3472 value = [2420, 1052]
x[9] <= 0.5 gini = 0.066 samples = 51813 value = [50032.0, 1781.0]
x[5] <= 0.899 gini = 0.114 samples = 2404 value = [146, 2258]
gini = 0.031 samples = 42229 value = [661, 41568]
gini = 0.306 samples = 859 value = [697, 162]
gini = 0.021 samples = 1033 value = [11, 1022]
gini = 0.024 samples = 2439 value = [2409, 30]
gini = 0.027 samples = 49719 value = [49043, 676]
gini = 0.498 samples = 2094 value = [989, 1105]
gini = 0.0 samples = 146 value = [146, 0]
gini = 0.0 samples = 2258 value = [0, 2258]

Random forest¶

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 overall result.One big advantage of random forest is that it can be used for both classification and regression problems, which form the majority of current machine learning systems. and It also resists overfitting found in decision trees.



Confusion matrix for Random forest:

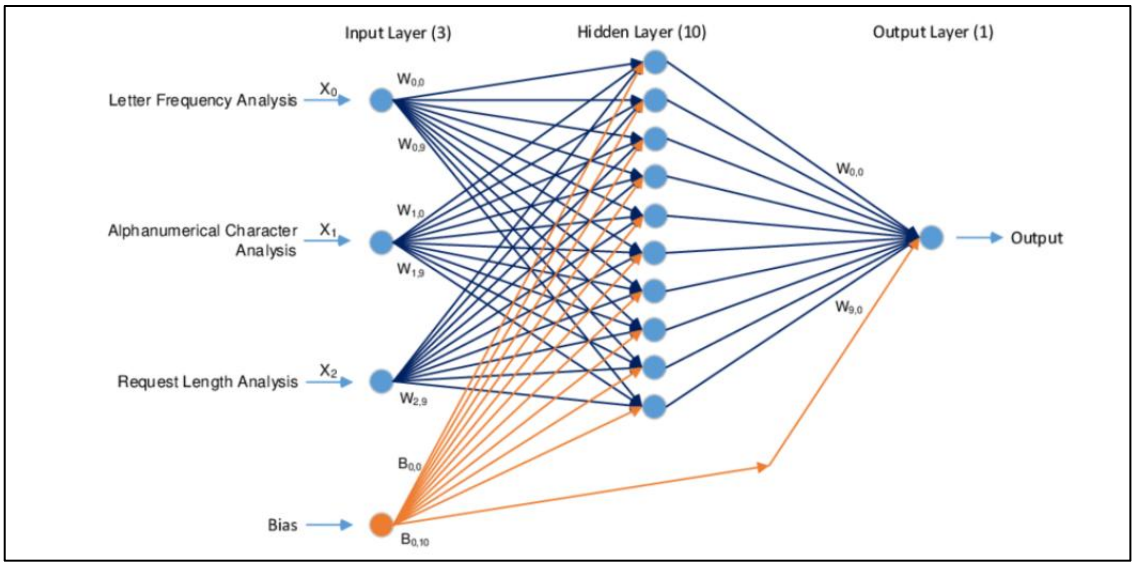

Random forest

```
[60]: rf = RandomForestClassifier().fit(x_train, y_train)
      evaluate_classification(rf, "RandomForestClassifier", x_train, x_test, y_train, y_test)
```

```
Training Accuracy RandomForestClassifier 99.99404626055548  Test Accuracy RandomForestClassifier 99.87695971422902
Training Precesion RandomForestClassifier 99.99359261869674  Test Precesion RandomForestClassifier 99.9321688994404
Training Recall RandomForestClassifier 99.99359261869674  Test Recall RandomForestClassifier 99.80523329663816
```

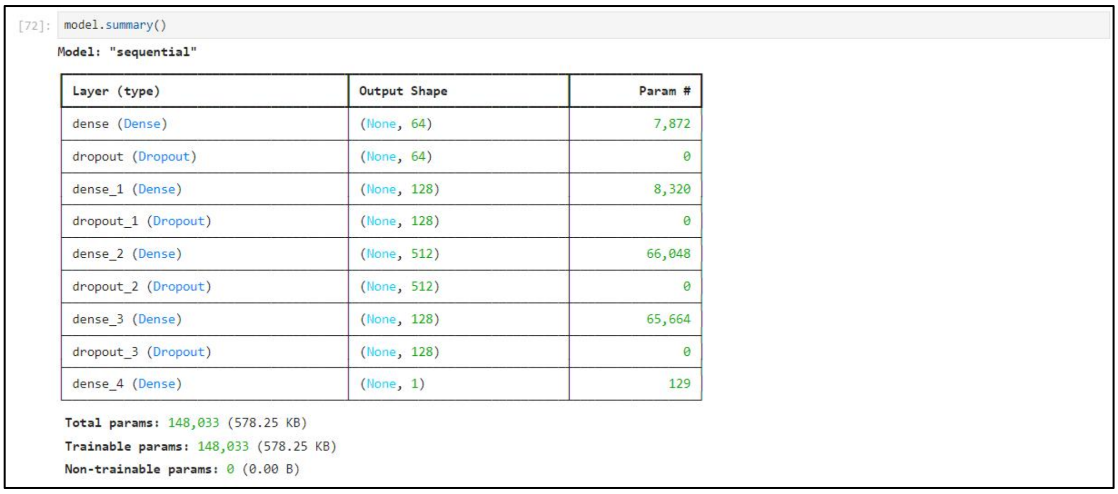
Neural networks¶

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 summary :

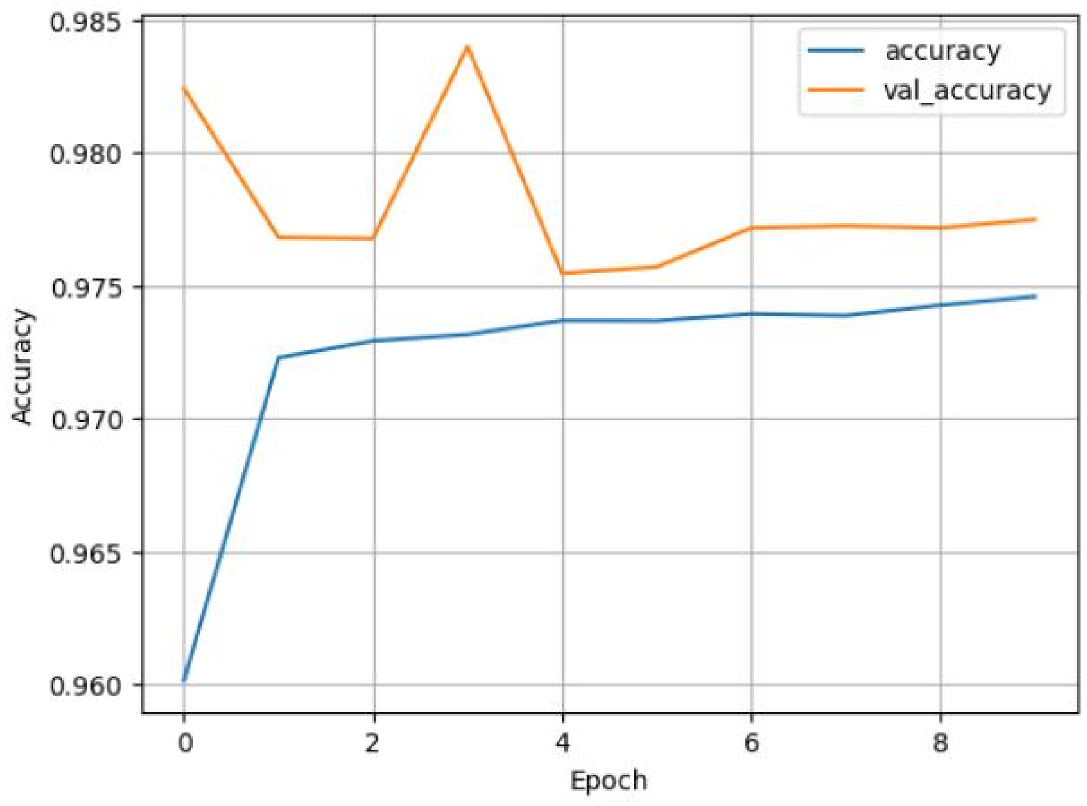


```
[72]: model.summary()
Model: "sequential"
```

```
┃ 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)
```

Accuracy by epochs :

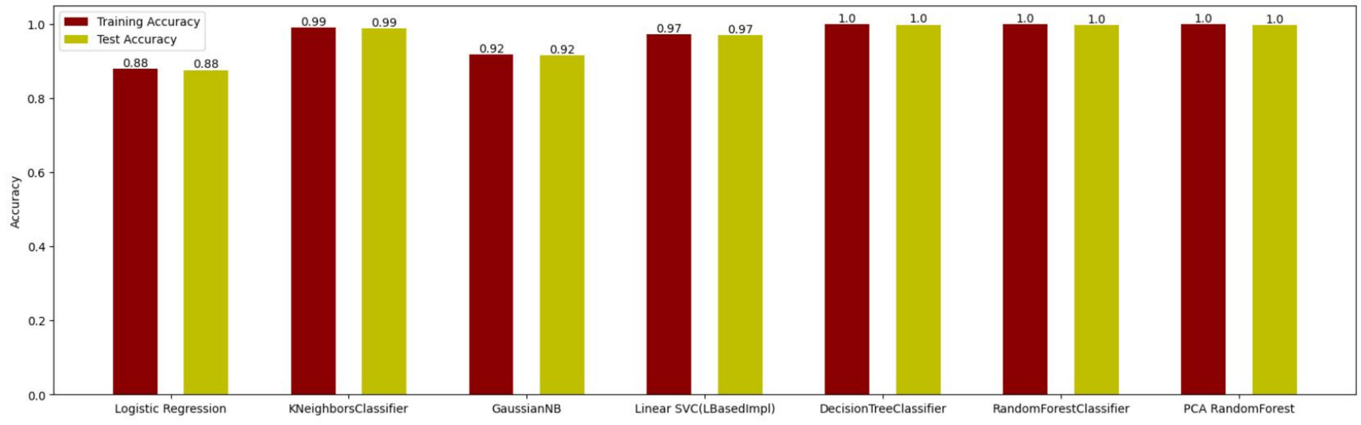


Conclusion :

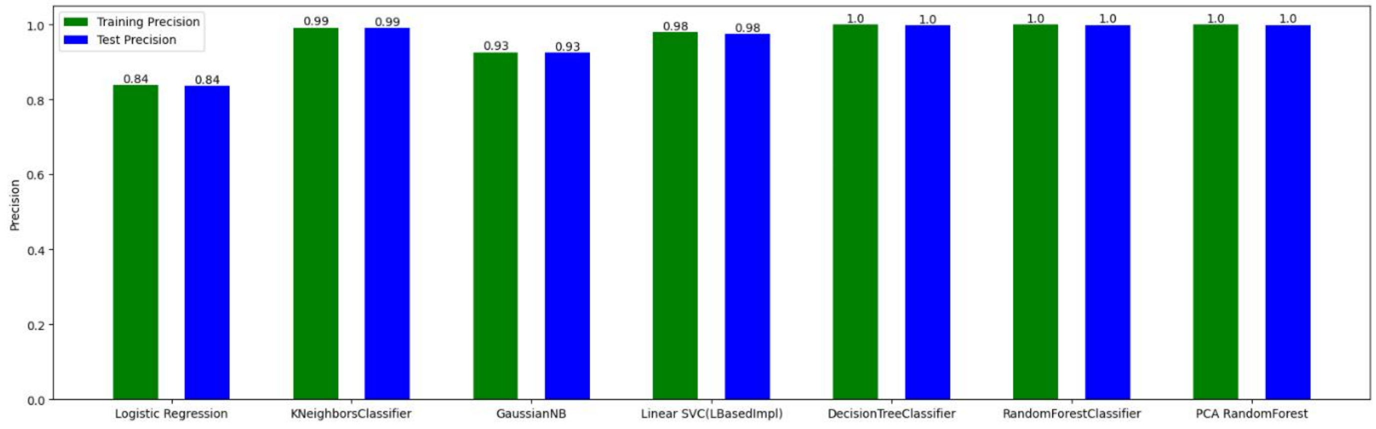
- Comparative analysis done between logistic regression, KNN, naive bayes, svm, decision tree, random forest, XGboost and using neural network classification for dataset, to analyze their accuracy.
- Decision tree, Random forest and K-nearest neighbours had shown good accuracy.

Results :

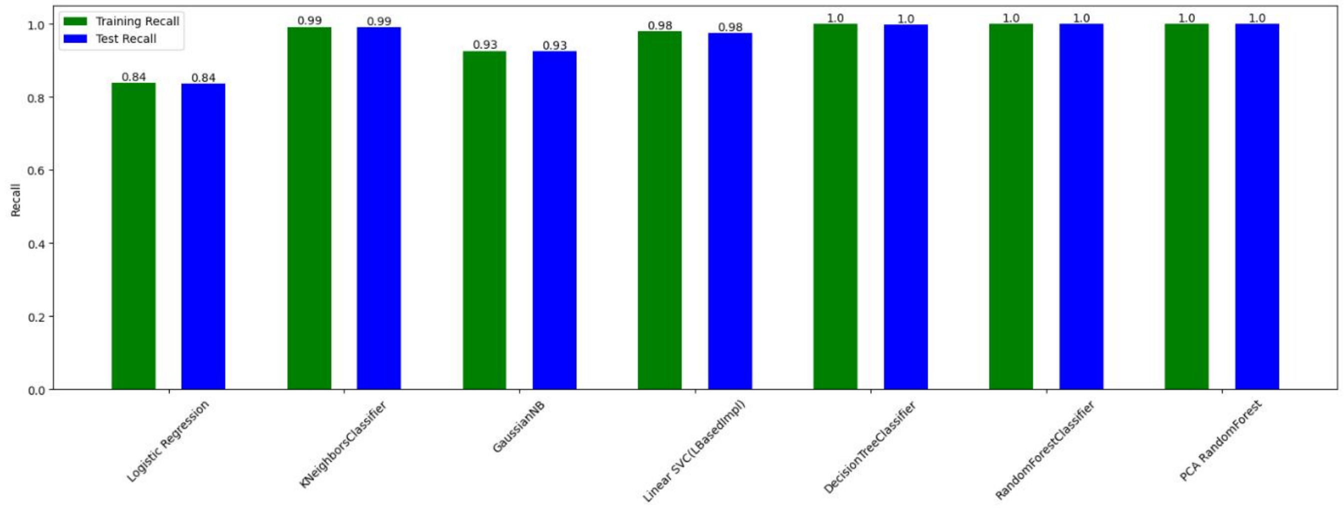
➢ Train and test accuracy score



❖ Train and test precision



❖ Train and test recall


Training Recall
Test Recall
Recall
Logistic Regression
KNeighborsClassifier
GaussianNB
Linear SVC(LBasedImpl)
DecisionTreeClassifier
RandomForestClassifier
PCA RandomForest