# intrusion-detection-system-with-ml-dl

October 22, 2024

## 1 Problem Statement

With the dramatic growth of computer networks usage and the huge increase in the number of applications running on top of it, network security is becoming increasingly while the all the systems suffers from security vulnerabilities, which could increase the attacks that could negatively affects the economy. Therefore detecting vulnerabilities in the system in the network has been more important and need to be done as accurate as possible in real time. in this notebook a model will be created and trained using SVM classifier to distengush if there is an attack or not in the network packet.

### 1.1 Intrusion detection systems

An Intrusion Detection System (IDS) is a system that monitors network traffic for suspicious activity and issues alerts when such activity is discovered. It is a software application that scans a network or a system for the harmful activity or policy breaching. Any malicious venture or violation is normally reported either to an administrator or collected centrally using a security information and event management (SIEM) system. A SIEM system integrates outputs from multiple sources and uses alarm filtering techniques to differentiate malicious activity from false alarms.

- Host-Based IDS (HIDS): A host-based IDS is deployed on a particular endpoint and designed to protect it against internal and external threats. Such an IDS may have the ability to monitor network traffic to and from the machine, observe running processes, and inspect the system's logs. A host-based IDS's visibility is limited to its host machine, decreasing the available context for decision-making, but has deep visibility into the host computer's internals.
- Network-Based IDS (NIDS): A network-based IDS solution is designed to monitor an entire protected network. It has visibility into all traffic flowing through the network and makes determinations based upon packet metadata and contents. This wider viewpoint provides more context and the ability to detect widespread threats; however, these systems lack visibility into the internals of the endpoints that they protect.

Detection Method of IDS:

- Signature-based Method: Signature-based IDS detects the attacks on the basis of the specific patterns such as number of bytes or number of 1's or number of 0's in the network traffic. It also detects on the basis of the already known malicious instruction sequence that is used by the malware. The detected patterns in the IDS are known as signatures. Signature-based IDS can easily detect the attacks whose pattern (signature) already exists in system but it is quite difficult to detect the new malware attacks as their pattern (signature) is not known.

- Anomaly-based Method: Anomaly-based IDS was introduced to detect unknown malware attacks as new malware are developed rapidly. In anomaly-based IDS there is use of machine learning to create a trustful activity model and anything coming is compared with that model and it is declared suspicious if it is not found in model. Machine learning-based method has a better-generalized property in comparison to signature-based IDS as these models can be trained according to the applications and hardware configurations.

### 1.2 Importing necessary libraries

[1]:
```
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
```

### 1.3 Exploring the dataset

[2]:
```
# Read Train and Test dataset
data_train = pd.read_csv("../input/nslkdd/KDDTrain+.txt")
```

[3]:
```
# Check data
data_train.head()
```

[3]:
```
   0  tcp ftp_data   SF  491   0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  \
0  0  udp    other   SF  146     0    0    0    0    0    0    0    0    0
1  0  tcp  private   S0    0     0    0    0    0    0    0    0    0    0
2  0  tcp     http   SF  232  8153    0    0    0    0    0    1    0    0
3  0  tcp     http   SF  199   420    0    0    0    0    0    1    0    0
4  0  tcp  private  REJ    0     0    0    0    0    0    0    0    0    0
```

```
   0.10  0.11  0.12  0.13  0.14  0.15  0.16  0.17    2  2.1  0.00  0.00.1  \
0     0     0     0     0     0     0     0     0   13    1   0.0     0.0
1     0     0     0     0     0     0     0     0  123    6   1.0     1.0
2     0     0     0     0     0     0     0     0    5    5   0.2     0.2
3     0     0     0     0     0     0     0     0   30   32   0.0     0.0
4     0     0     0     0     0     0     0     0  121   19   0.0     0.0

   0.00.2  0.00.3  1.00  0.00.4  0.00.5  150   25  0.17.1  0.03  0.17.2  \
0     0.0     0.0  0.08    0.15    0.00  255    1    0.00  0.60    0.88
1     0.0     0.0  0.05    0.07    0.00  255   26    0.10  0.05    0.00
2     0.0     0.0  1.00    0.00    0.00   30  255    1.00  0.00    0.03
3     0.0     0.0  1.00    0.00    0.09  255  255    1.00  0.00    0.00
4     1.0     1.0  0.16    0.06    0.00  255   19    0.07  0.07    0.00

   0.00.6  0.00.7  0.00.8  0.05  0.00.9   normal  20
0    0.00    0.00    0.00   0.0    0.00   normal  15
1    0.00    1.00    1.00   0.0    0.00  neptune  19
2    0.04    0.03    0.01   0.0    0.01   normal  21
3    0.00    0.00    0.00   0.0    0.00   normal  21
4    0.00    0.00    0.00   1.0    1.00  neptune  21
```

[4]:
```
columns =␣
 ↪(['duration','protocol_type','service','flag','src_bytes','dst_bytes','land','wrong_fragmen
,'num_failed_logins','logged_in','num_compromised','root_shell','su_attempted','num_root','num
,'num_shells','num_access_files','num_outbound_cmds','is_host_login','is_guest_login','count',
,'srv_serror_rate','rerror_rate','srv_rerror_rate','same_srv_rate','diff_srv_rate','srv_diff_h
,'dst_host_same_srv_rate','dst_host_diff_srv_rate','dst_host_same_src_port_rate','dst_host_srv
,'dst_host_srv_serror_rate','dst_host_rerror_rate','dst_host_srv_rerror_rate','outcome','level
```

[5]:
```
# Assign name for columns
data_train.columns = columns
```

[6]:
```
data_train.head()
```

[6]:
```
   duration protocol_type  service flag  src_bytes  dst_bytes  land  \
0         0           udp    other   SF        146          0     0
1         0           tcp  private   S0          0          0     0
2         0           tcp     http   SF        232       8153     0
3         0           tcp     http   SF        199        420     0
4         0           tcp  private  REJ          0          0     0

   wrong_fragment  urgent  hot  num_failed_logins  logged_in  num_compromised  \
0               0       0    0                  0          0                0
1               0       0    0                  0          0                0
2               0       0    0                  0          1                0
3               0       0    0                  0          1                0
4               0       0    0                  0          0                0
```

```
   root_shell  su_attempted  num_root  num_file_creations  num_shells  \
0           0             0         0                   0           0
1           0             0         0                   0           0
2           0             0         0                   0           0
3           0             0         0                   0           0
4           0             0         0                   0           0

   num_access_files  num_outbound_cmds  is_host_login  is_guest_login  count  \
0                 0                  0              0               0     13
1                 0                  0              0               0    123
2                 0                  0              0               0      5
3                 0                  0              0               0     30
4                 0                  0              0               0    121

   srv_count  serror_rate  srv_serror_rate  rerror_rate  srv_rerror_rate  \
0          1          0.0              0.0          0.0              0.0
1          6          1.0              1.0          0.0              0.0
2          5          0.2              0.2          0.0              0.0
3         32          0.0              0.0          0.0              0.0
4         19          0.0              0.0          1.0              1.0

   same_srv_rate  diff_srv_rate  srv_diff_host_rate  dst_host_count  \
0           0.08           0.15                0.00             255
1           0.05           0.07                0.00             255
2           1.00           0.00                0.00              30
3           1.00           0.00                0.09             255
4           0.16           0.06                0.00             255

   dst_host_srv_count  dst_host_same_srv_rate  dst_host_diff_srv_rate  \
0                   1                    0.00                    0.60
1                  26                    0.10                    0.05
2                 255                    1.00                    0.00
3                 255                    1.00                    0.00
4                  19                    0.07                    0.07

   dst_host_same_src_port_rate  dst_host_srv_diff_host_rate  \
0                         0.88                         0.00
1                         0.00                         0.00
2                         0.03                         0.04
3                         0.00                         0.00
4                         0.00                         0.00

   dst_host_serror_rate  dst_host_srv_serror_rate  dst_host_rerror_rate  \
0                  0.00                      0.00                   0.0
1                  1.00                      1.00                   0.0
2                  0.03                      0.01                   0.0
```

```
3                0.00                     0.00                 0.0
4                0.00                     0.00                 1.0

   dst_host_srv_rerror_rate  outcome  level
0                      0.00   normal     15
1                      0.00  neptune     19
2                      0.01   normal     21
3                      0.00   normal     21
4                      1.00  neptune     21
```

[7]:
```
data_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 125972 entries, 0 to 125971
Data columns (total 43 columns):
 #   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
 42  level                        125972 non-null  int64
dtypes: float64(15), int64(24), object(4)
memory usage: 41.3+ MB
```

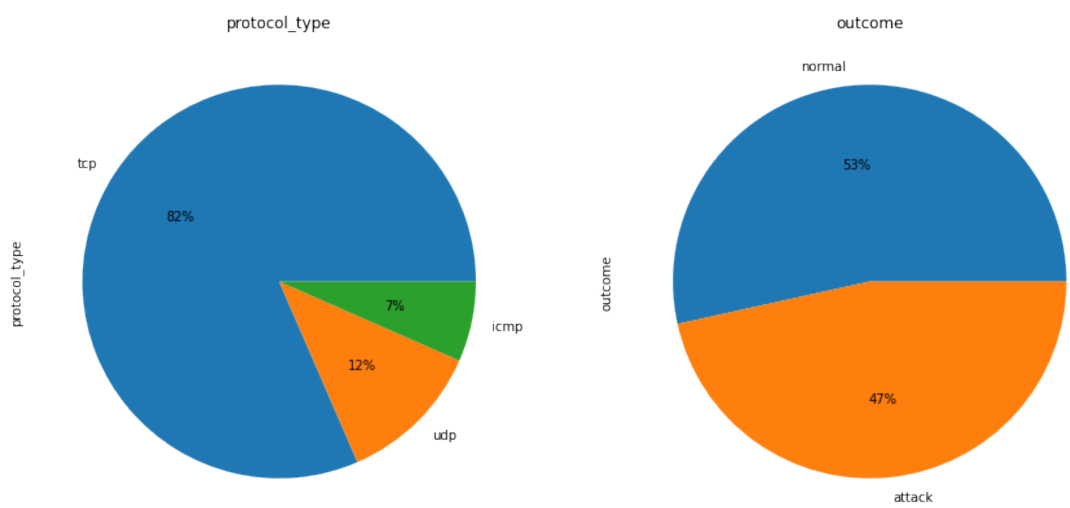
[8]:
```
data_train.describe().style.background_gradient(cmap='Blues').
 ↪set_properties(**{'font-family':'Segoe UI'})
```

[8]:
```
<pandas.io.formats.style.Styler at 0x7f2a83473450>
```

[9]:
```
data_train.loc[data_train['outcome'] == "normal", "outcome"] = 'normal'
data_train.loc[data_train['outcome'] != 'normal', "outcome"] = 'attack'
```

[10]:
```
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()
```

[11]:
```
pie_plot(data_train, ['protocol_type', 'outcome'], 1, 2)
```



### 1.4 Preprocessing the data

[12]:
```
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
```

[13]:
```
cat_cols = ['is_host_login','protocol_type','service','flag','land',␣
↪'logged_in','is_guest_login', 'level', 'outcome']
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
```

[14]:
```
scaled_train = preprocess(data_train)
```

#### 1.4.1 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.

[15]:
```
x = scaled_train.drop(['outcome', 'level'] , axis = 1).values
y = scaled_train['outcome'].values
```

```
y_reg = scaled_train['level'].values

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]))

y = y.astype('int')
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=0.2, random_state=42)
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
```

[16]:
```
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, test_recall]
    print("Training Accuracy " + str(name) + " {}  Test Accuracy ".
 ↪format(train_accuracy*100) + str(name) + " {}".format(test_accuracy*100))
    print("Training Precesion " + str(name) + " {}  Test Precesion ".
 ↪format(train_precision*100) + str(name) + " {}".format(test_precision*100))
    print("Training Recall " + str(name) + " {}  Test Recall ".
 ↪format(train_recall*100) + str(name) + " {}".format(test_recall*100))

    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', 'attack'])

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

## 1.5 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.

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

## 1.6 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:

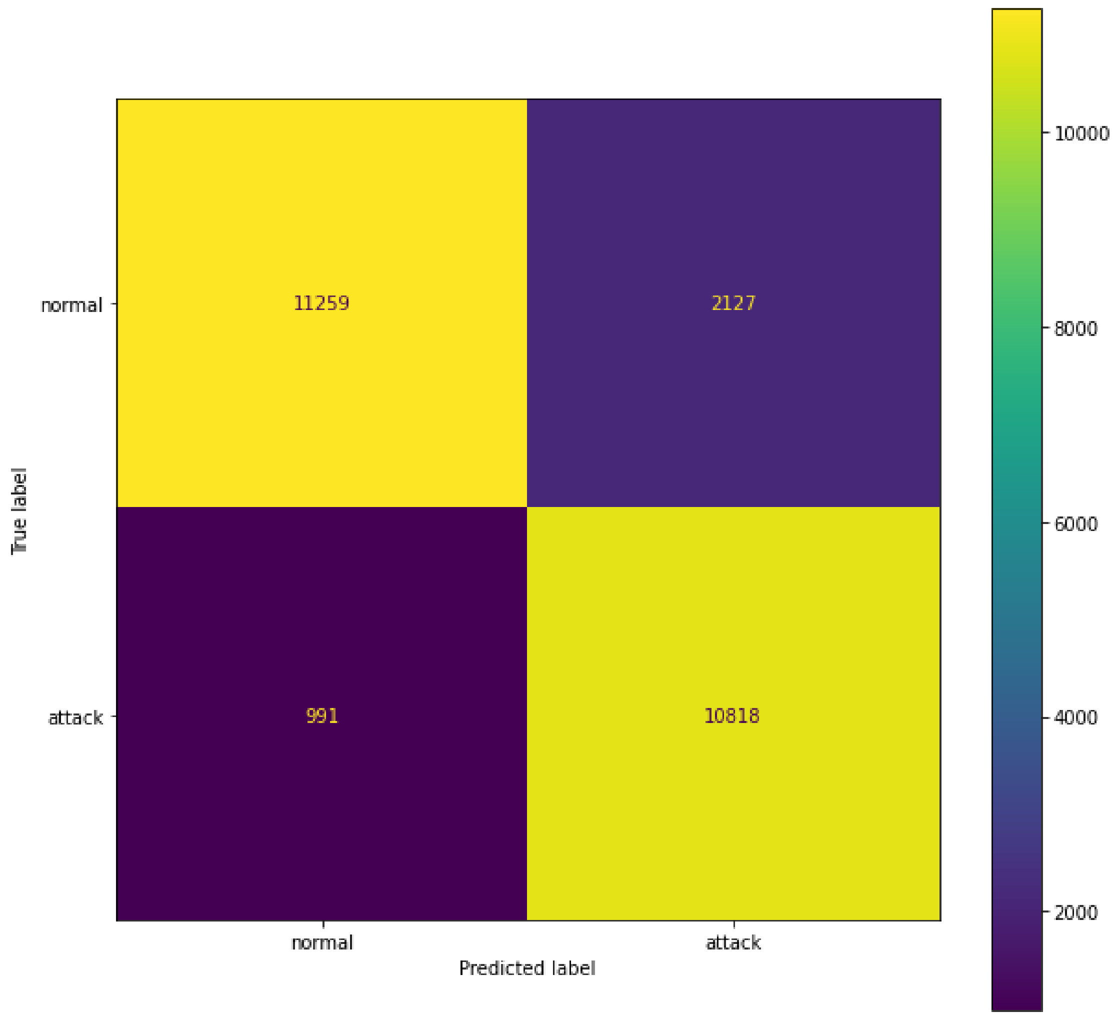
In this logistic regression equation, h is the dependent or response variable and x is the independent variable. The beta parameter, or coefficient, in this model is commonly estimated via maximum likelihood estimation (MLE). This method tests different values of beta through multiple iterations to optimize 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.

[17]:
```
lr = LogisticRegression().fit(x_train, y_train)
evaluate_classification(lr, "Logistic Regression", x_train, x_test, y_train,␣
 ↪y_test)
```

```
Training Accuracy Logistic Regression 87.97443861198488  Test Accuracy Logistic
Regression 87.62452867632466
Training Precesion Logistic Regression 83.81338426160502  Test Precesion
Logistic Regression 83.56894553881807
Training Recall Logistic Regression 91.85621836355482  Test Recall Logistic
Regression 91.60809552036582
```




### 1.7 k-nearest neighbors

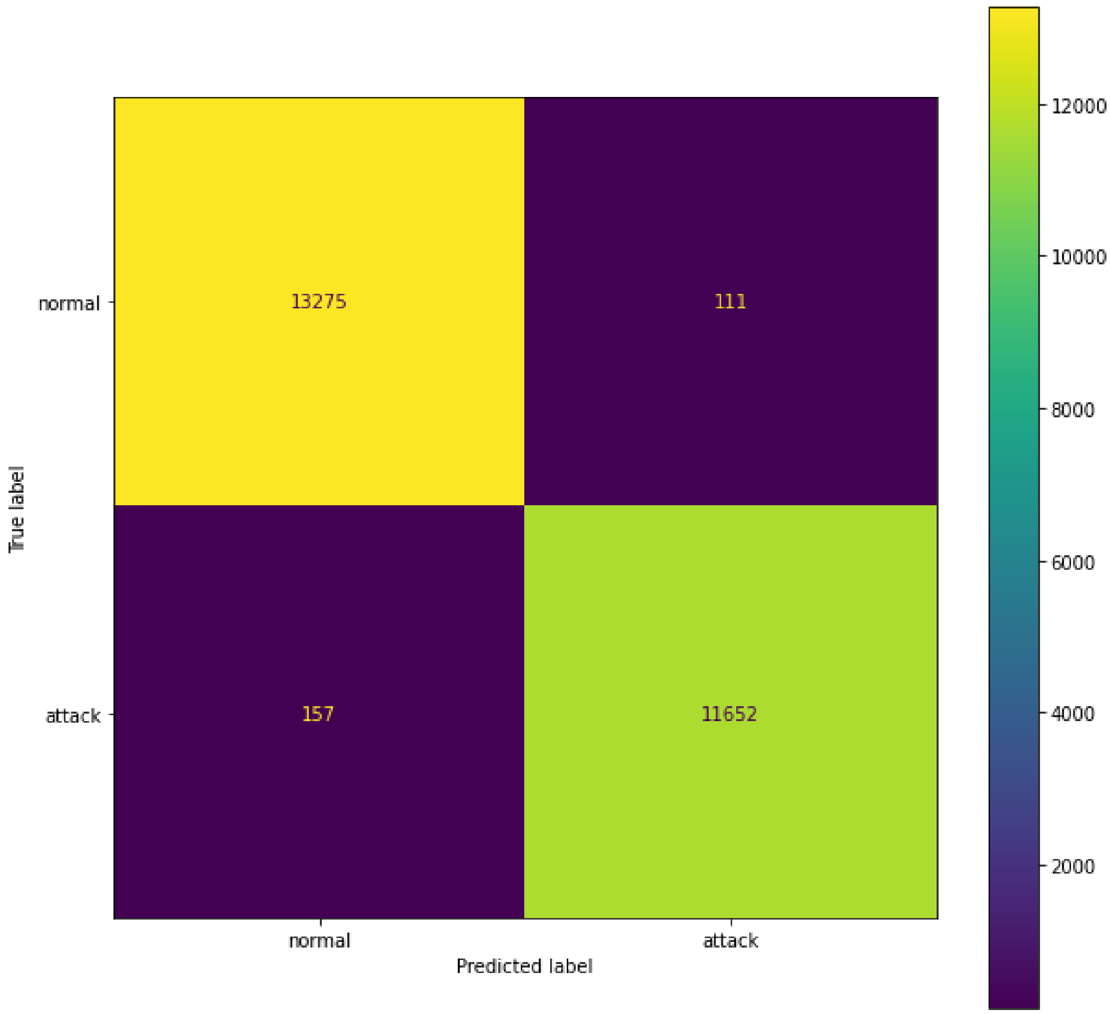
The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. While it can be used for either regression or classification problems, it is typically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

### 1.7.1 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.

[18]:
```
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
```




## 1.8 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.

### 1.8.1 Bayes' Theorem

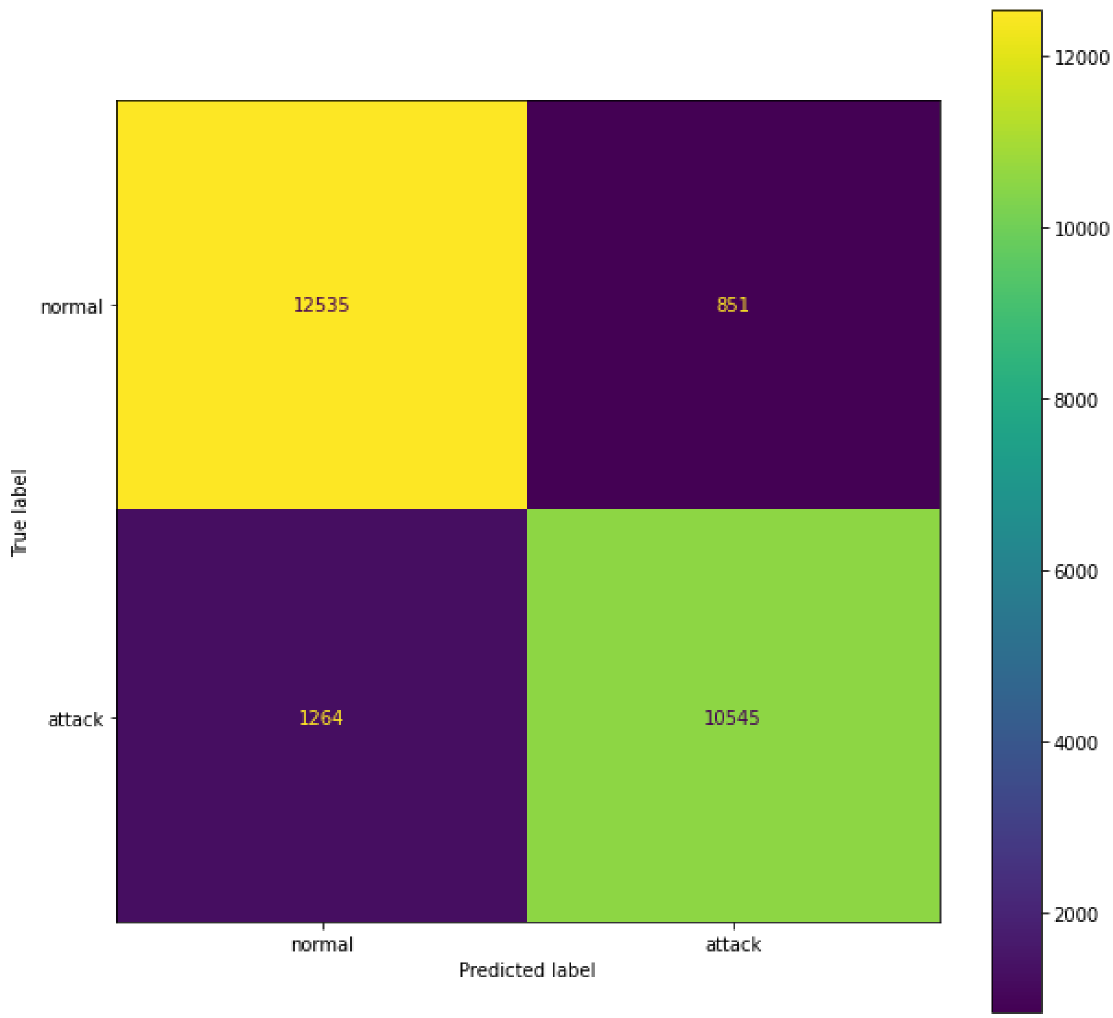
Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is stated mathematically as the following equation:

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.

[19]:
```
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
```



### 1.9 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 :

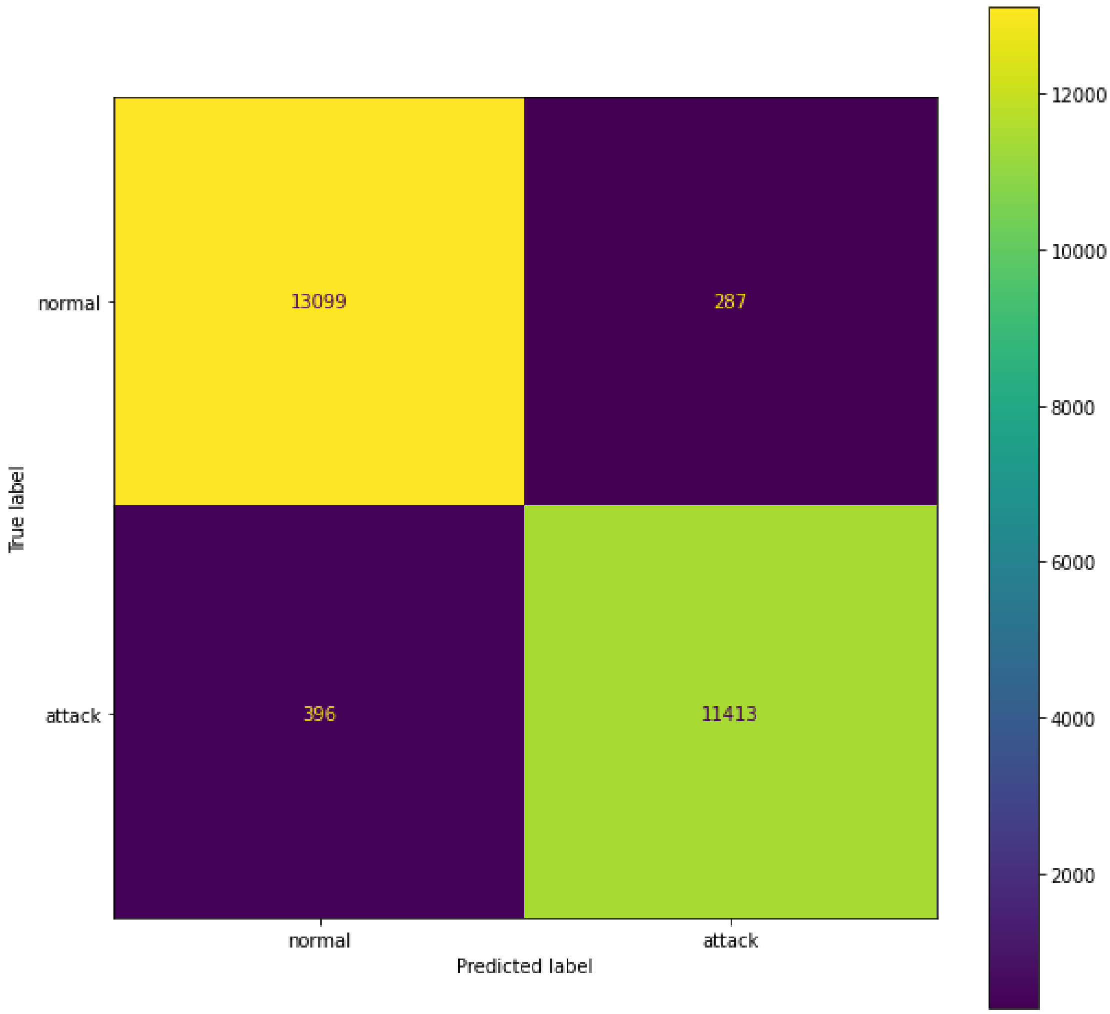
- Radial Basis Function Kernel (RBF): The similarity between two points in the transformed feature space is an exponentially decaying function of the distance between the vectors and the original input space as shown below. RBF is the default kernel used in SVM.

- Polynomial Kernel: The Polynomial kernel takes an additional parameter, 'degree' that controls the model's complexity and computational cost of the transformation

[20]:
```
lin_svc = svm.LinearSVC().fit(x_train, y_train)
```

[21]:
```
evaluate_classification(lin_svc, "Linear SVC(LBasedImpl)", x_train, x_test,␣
↪y_train, y_test)
```

```
Training Accuracy Linear SVC(LBasedImpl) 97.48454508469194  Test Accuracy Linear
SVC(LBasedImpl) 97.28914467156183
Training Precesion Linear SVC(LBasedImpl) 97.85399377593362  Test Precesion
Linear SVC(LBasedImpl) 97.54700854700855
Training Recall Linear SVC(LBasedImpl) 96.70660601012366  Test Recall Linear
SVC(LBasedImpl) 96.64662545516131
```



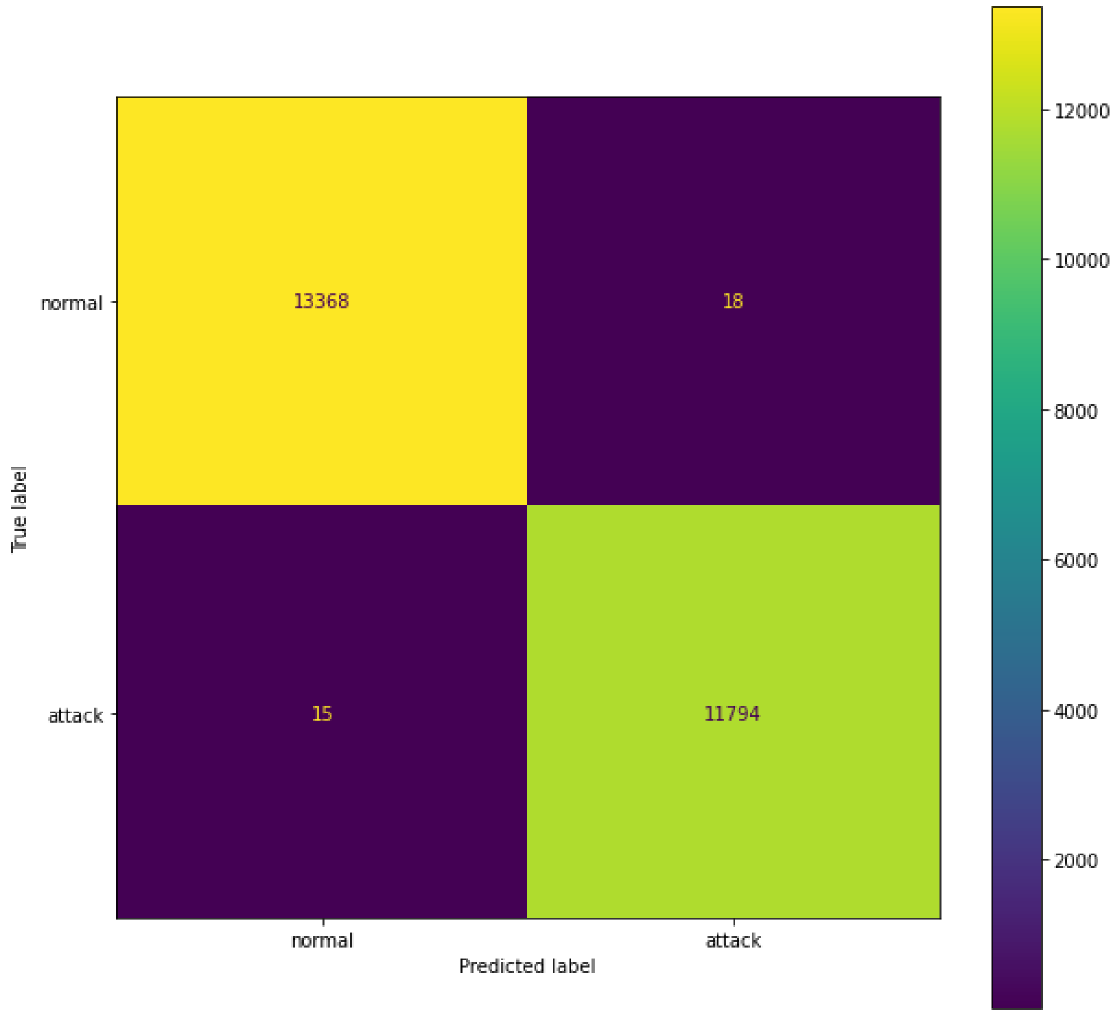

## 1.10 Decision Tree

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).

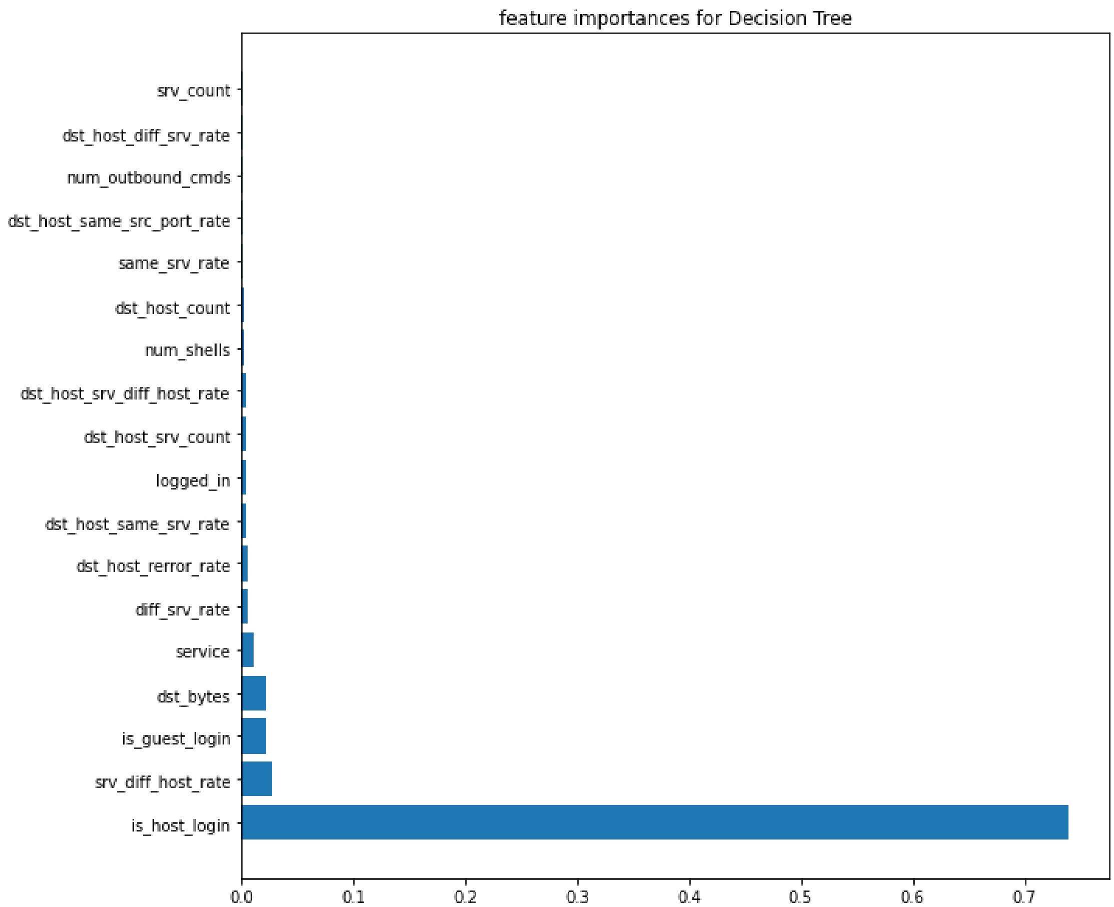
[22]:
```
dt = DecisionTreeClassifier(max_depth=3).fit(x_train, y_train)
tdt = DecisionTreeClassifier().fit(x_train, y_train)
evaluate_classification(tdt, "DecisionTreeClassifier", x_train, x_test,␣
↪y_train, y_test)
```

```
Training Accuracy DecisionTreeClassifier 99.99404626055548  Test Accuracy
DecisionTreeClassifier 99.86902163127604
Training Precesion DecisionTreeClassifier 100.0  Test Precesion
DecisionTreeClassifier 99.84761259735862
Training Recall DecisionTreeClassifier 99.98718523739348  Test Recall
DecisionTreeClassifier 99.87297823693793
```



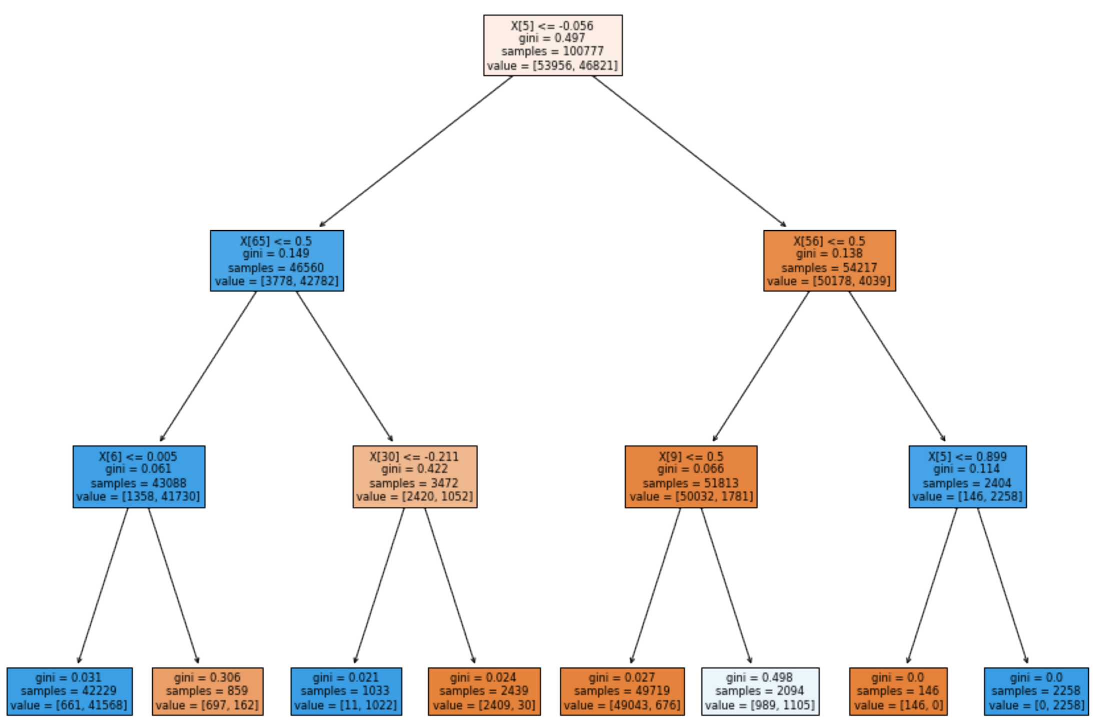

[23]:
```
def f_importances(coef, names, top=-1):
    imp = coef
```

```
    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)
```




[24]:
```
fig = plt.figure(figsize=(15,12))
tree.plot_tree(dt , filled=True)
```

```
[24]: [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(0.125, 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, 0.375, 'X[9] <= 0.5\ngini = 0.066\nsamples = 51813\nvalue = [50032,
1781]'),
 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]')]
```

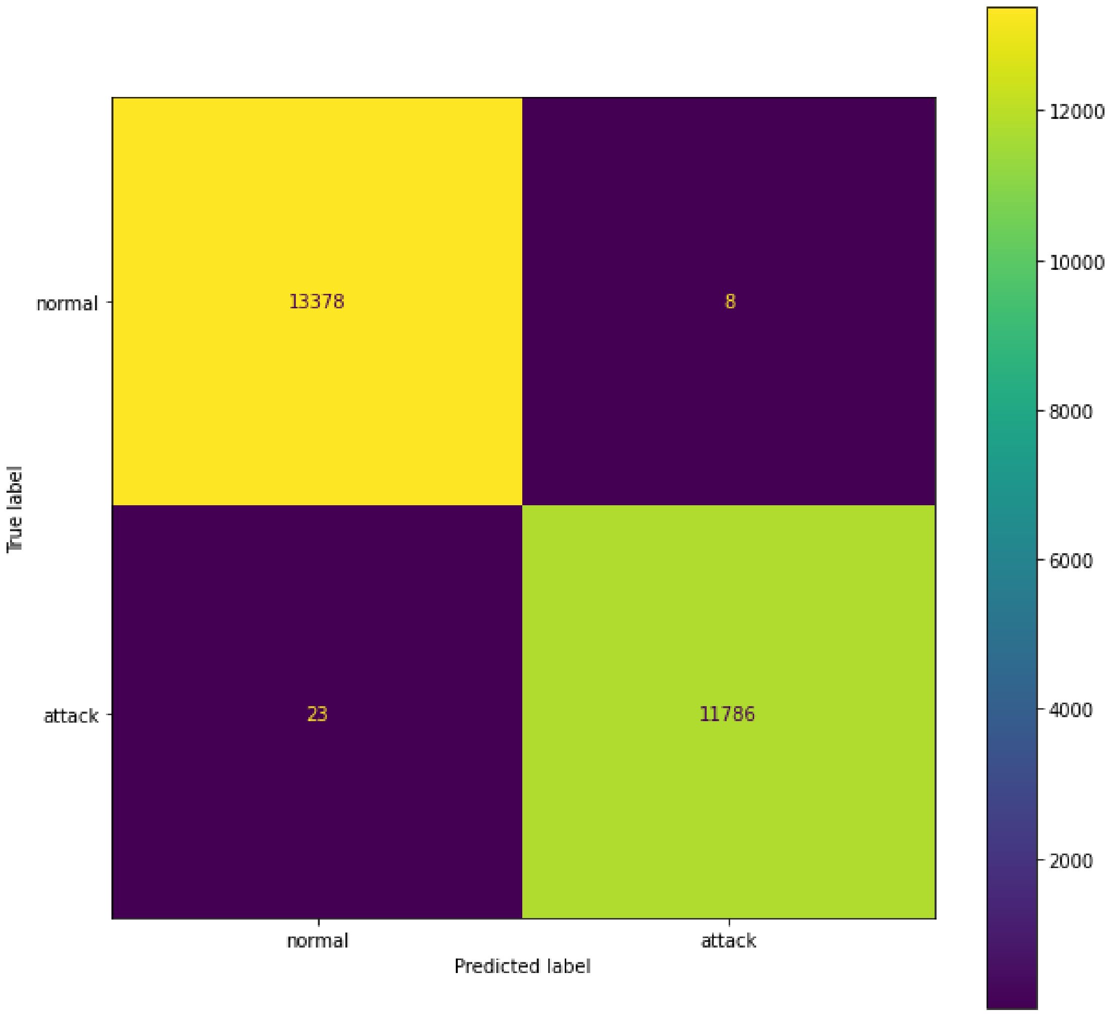


## 1.11 Random forest

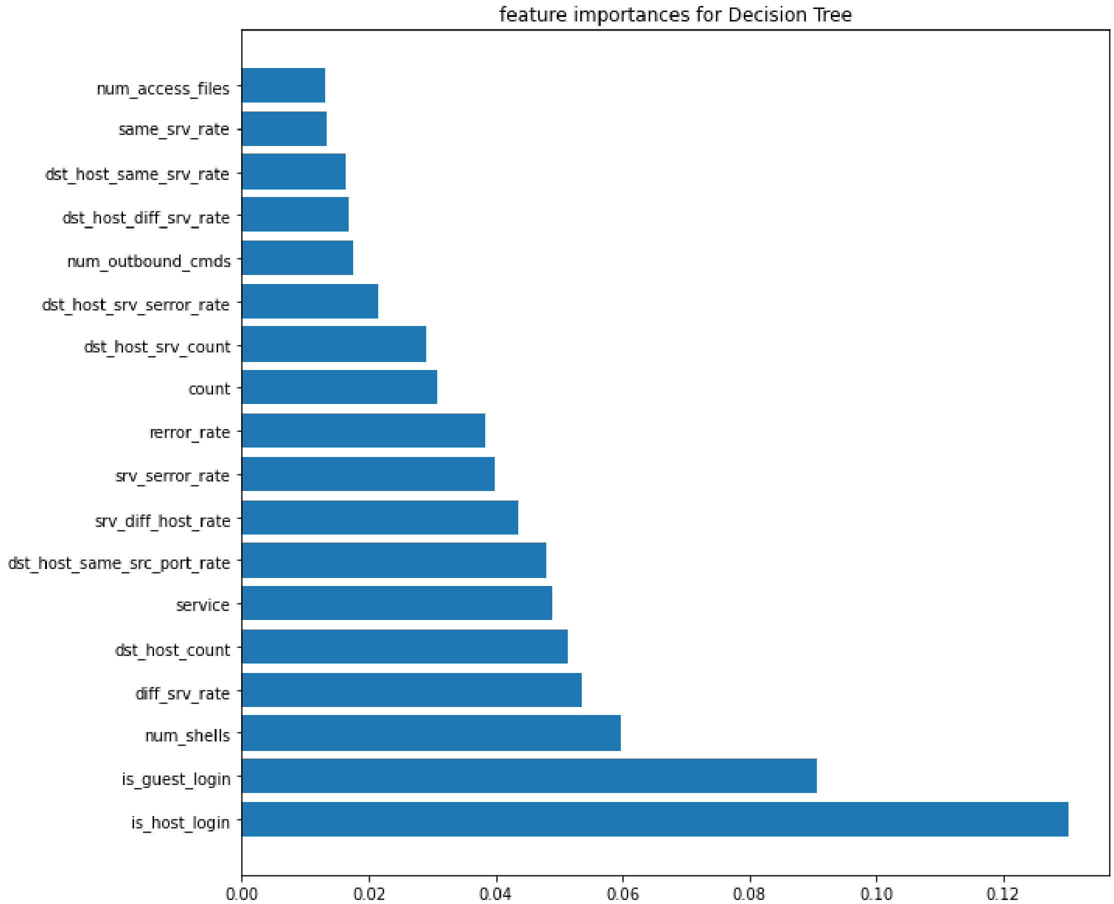
Random forest is a supervised learning algorithm. The "forest" it builds is an ensemble of decision trees, usually trained with the "bagging" method. The general idea of the bagging method is that a combination of learning models increases the overall result.One big advantage of random forest is that 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.

```
[25]: 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 99.9914571898426  Test Precesion
RandomForestClassifier 99.9321688994404
Training Recall RandomForestClassifier 99.99572841246449  Test Recall
RandomForestClassifier 99.80523329663816
```




[26]:
```
f_importances(abs(rf.feature_importances_), features_names, top=18)
```




### 1.12 Building an XGBOOST REgressor regressor in order to predict threat level

[27]:
```
xg_r = xgb.XGBRegressor(objective ='reg:linear',n_estimators = 20).
↪fit(x_train_reg, y_train_reg)
```

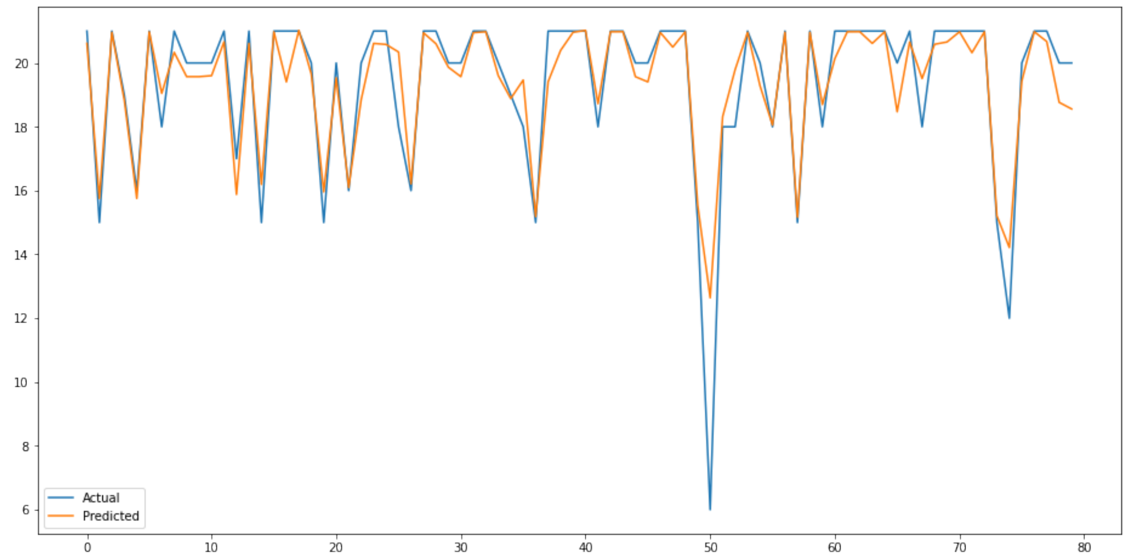
```
[15:03:28] WARNING: ../src/objective/regression_obj.cu:203: reg:linear is now
deprecated in favor of reg:squarederror.
```

[28]:
```
name = "XGBOOST"
train_error = metrics.mean_squared_error(y_train_reg, xg_r.
↪predict(x_train_reg), squared=False)
test_error = metrics.mean_squared_error(y_test_reg, xg_r.predict(x_test_reg),␣
↪squared=False)
print("Training Error " + str(name) + " {}  Test error ".format(train_error) +␣
↪str(name) + " {}".format(test_error))
```

```
Training Error XGBOOST 0.9467538200777741  Test error XGBOOST 1.006973984055062
```

[29]:
```
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'])
```

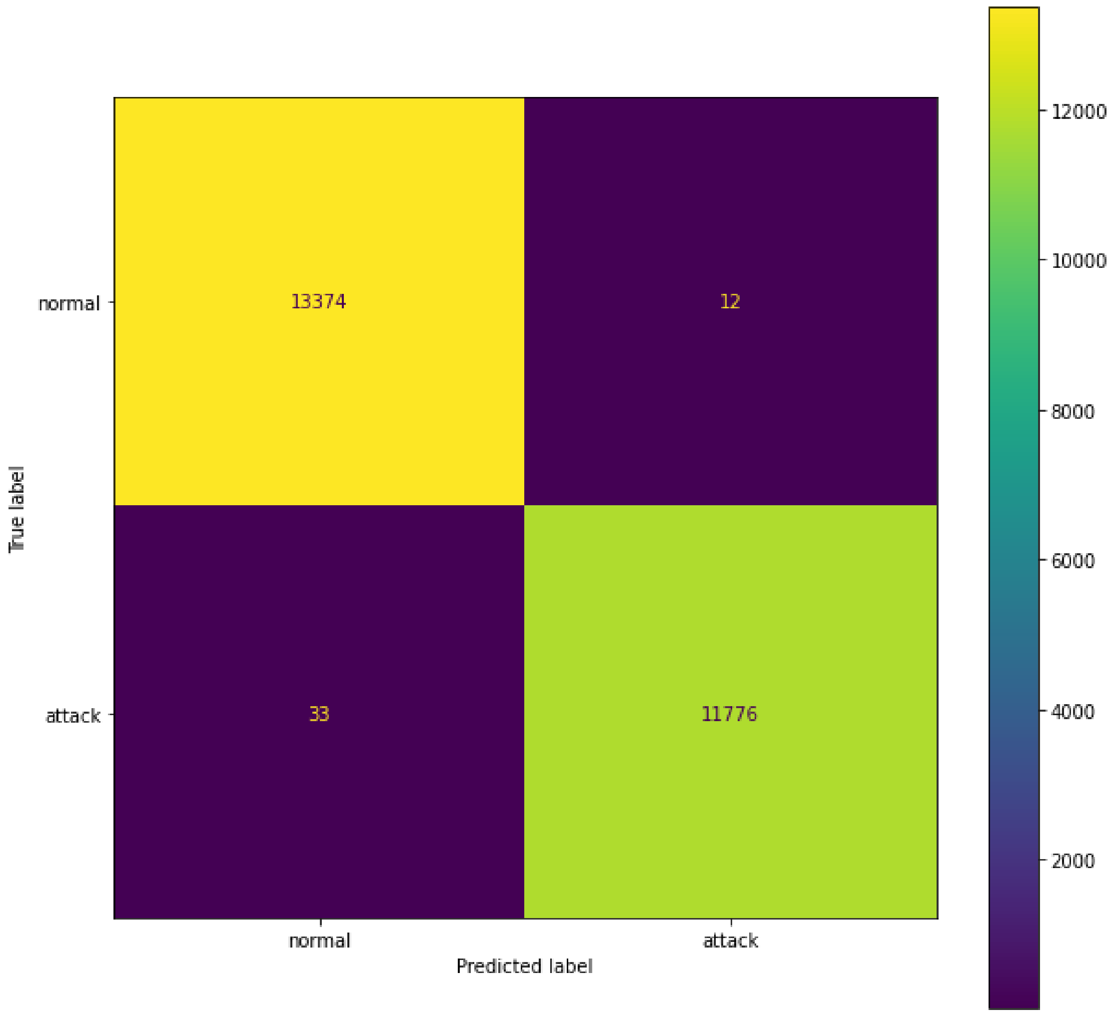
[29]: <matplotlib.legend.Legend at 0x7f2a821ba4d0>




### 1.13 Measuring effect of PCA

[30]:
```
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_reduced)
```

```
Training Accuracy PCA RandomForest 99.99404626055548  Test Accuracy PCA
RandomForest 99.82139313355825
Training Precesion PCA RandomForest 99.9914571898426  Test Precesion PCA
RandomForest 99.8982015609094
Training Recall PCA RandomForest 99.99572841246449  Test Recall PCA RandomForest
99.72055212126344
```




### 1.14 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.

[31]:
```
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'),
])
```

```
2022-09-16 15:04:39.351148: I
tensorflow/core/common_runtime/process_util.cc:146] Creating new thread pool
with default inter op setting: 2. Tune using inter_op_parallelism_threads for
best performance.
```

[32]:
```
model.compile(optimizer='adam', loss=tf.keras.losses.
 ↪BinaryCrossentropy(from_logits=True), metrics=['accuracy'])
```

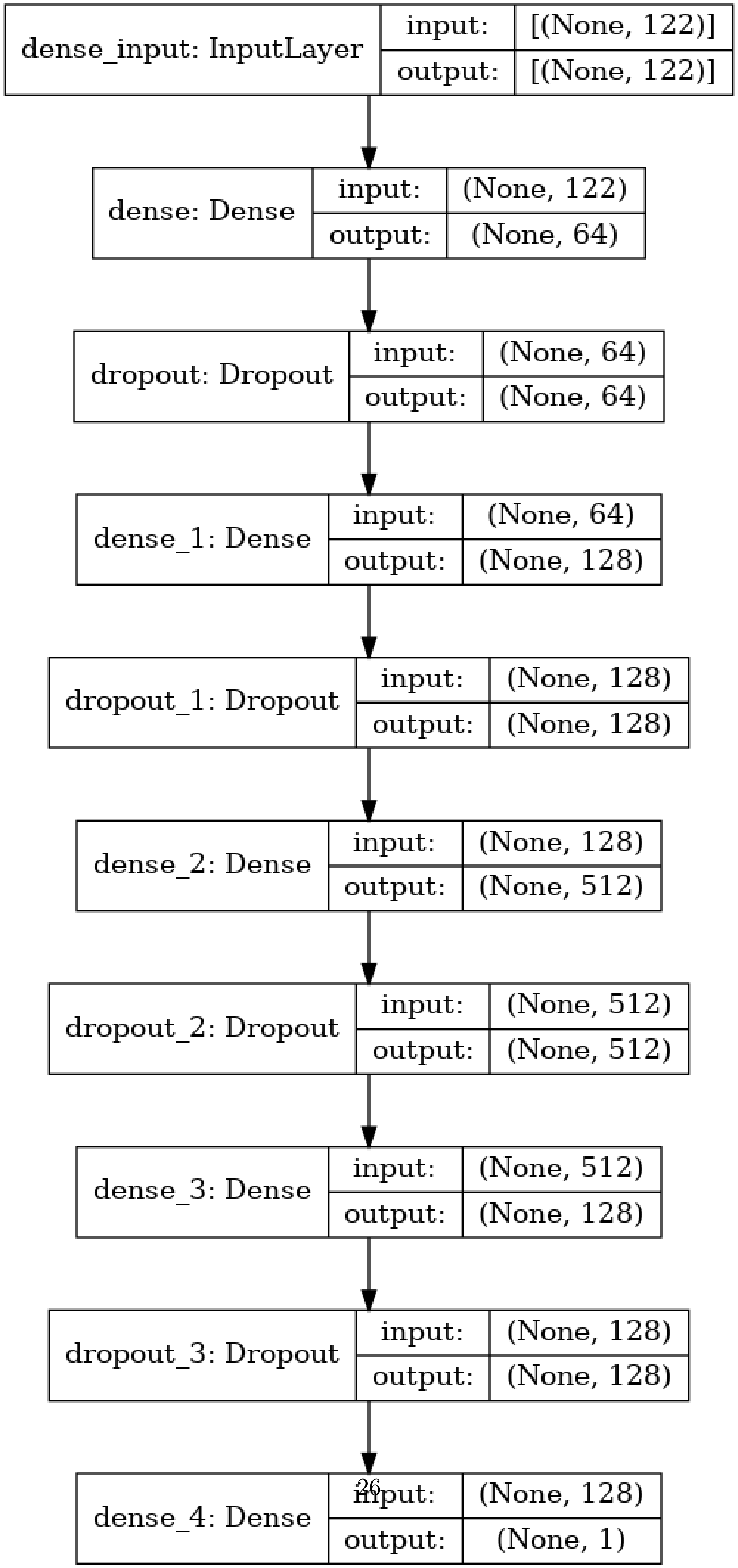
[33]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                7872

_________________________________________________________________
dropout (Dropout)            (None, 64)                0
```

```
_________________________________________________________________
dense_1 (Dense)              (None, 128)               8320
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               66048
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               65664
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 129
=================================================================
Total params: 148,033
Trainable params: 148,033
Non-trainable params: 0
_________________________________________________________________
```

[34]:
```
from keras.utils.vis_utils import plot_model
plot_model(model, to_file='model_plot.png', show_shapes=True,␣
 ↪show_layer_names=True)
```

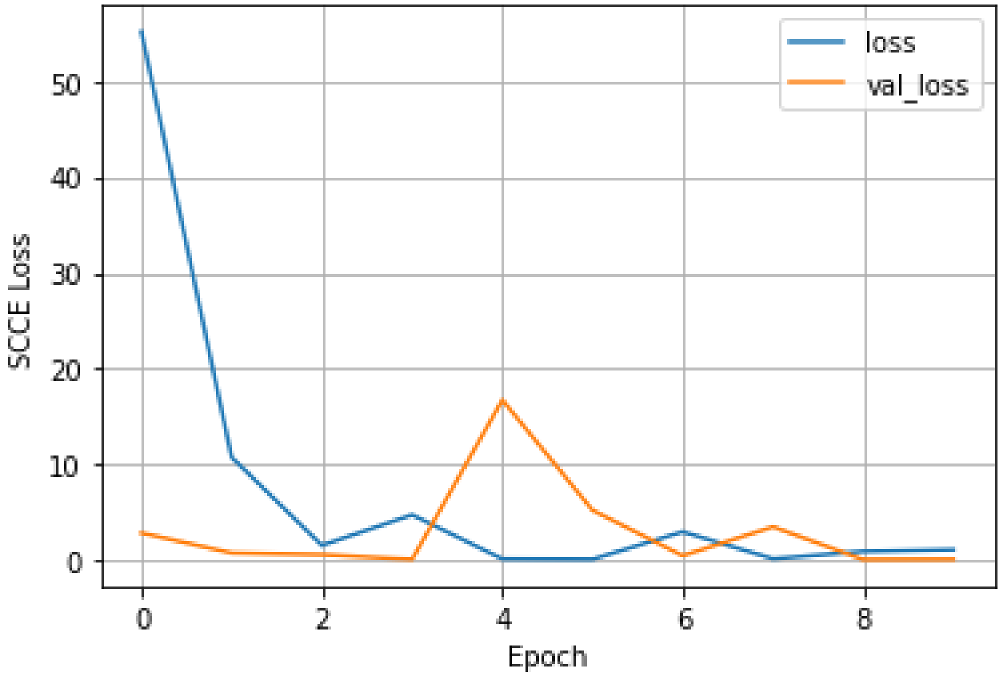
[34]:


dense_input: InputLayer
input: [(None, 122)]
output: [(None, 122)]
dense: Dense
input: (None, 122)
output: (None, 64)
dropout: Dropout
input: (None, 64)
output: (None, 64)
dense_1: Dense
input: (None, 64)
output: (None, 128)
dropout_1: Dropout
input: (None, 128)
output: (None, 128)
dense_2: Dense
input: (None, 128)
output: (None, 512)
dropout_2: Dropout
input: (None, 512)
output: (None, 512)
dense_3: Dense
input: (None, 512)
output: (None, 128)
dropout_3: Dropout
input: (None, 128)
output: (None, 128)
dense_4: Dense
input: (None, 128)
output: (None, 1)

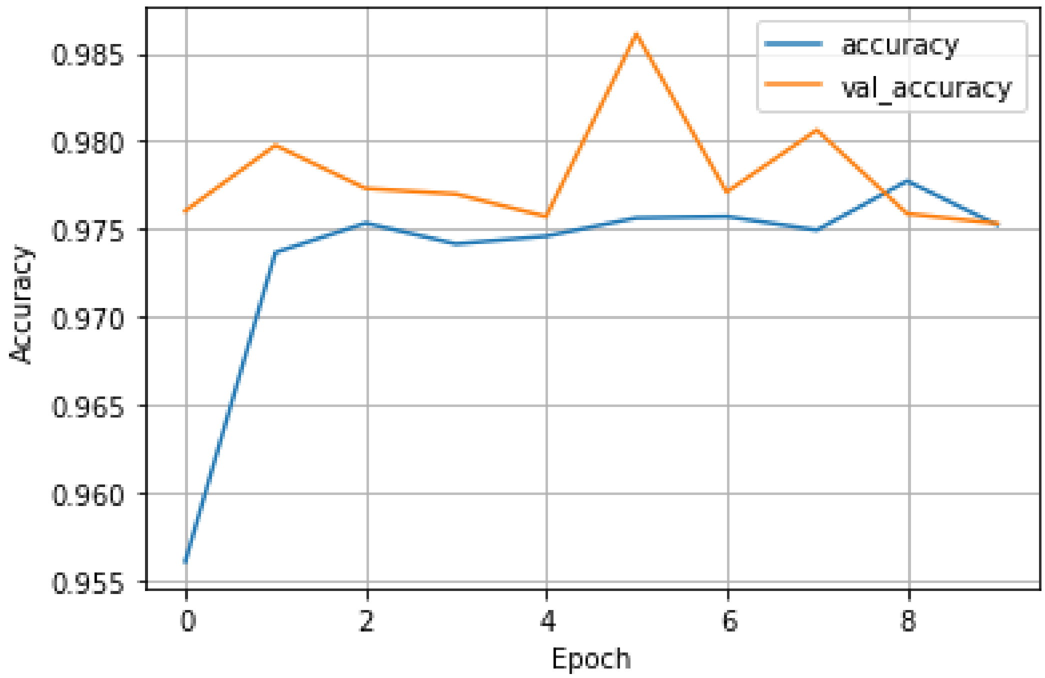
[35]:
```
history = model.fit(x_train, y_train, validation_data=(x_test, y_test),␣
    ↪epochs=10, verbose=1)
```

```
2022-09-16 15:04:41.515464: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR
Optimization Passes are enabled (registered 2)

Epoch 1/10
3150/3150 [==============================] - 25s 7ms/step - loss: 55.1458 -
accuracy: 0.9560 - val_loss: 2.8677 - val_accuracy: 0.9760
Epoch 2/10
3150/3150 [==============================] - 21s 7ms/step - loss: 10.7582 -
accuracy: 0.9737 - val_loss: 0.7768 - val_accuracy: 0.9798
Epoch 3/10
3150/3150 [==============================] - 21s 7ms/step - loss: 1.5692 -
accuracy: 0.9753 - val_loss: 0.6050 - val_accuracy: 0.9773
Epoch 4/10
3150/3150 [==============================] - 21s 7ms/step - loss: 4.7620 -
accuracy: 0.9741 - val_loss: 0.1078 - val_accuracy: 0.9770
Epoch 5/10
3150/3150 [==============================] - 21s 7ms/step - loss: 0.1535 -
accuracy: 0.9746 - val_loss: 16.7390 - val_accuracy: 0.9757
Epoch 6/10
3150/3150 [==============================] - 21s 7ms/step - loss: 0.1148 -
accuracy: 0.9756 - val_loss: 5.2444 - val_accuracy: 0.9861
Epoch 7/10
3150/3150 [==============================] - 20s 6ms/step - loss: 2.9888 -
accuracy: 0.9757 - val_loss: 0.4911 - val_accuracy: 0.9771
Epoch 8/10
3150/3150 [==============================] - 20s 6ms/step - loss: 0.1688 -
accuracy: 0.9749 - val_loss: 3.5074 - val_accuracy: 0.9806
Epoch 9/10
3150/3150 [==============================] - 20s 6ms/step - loss: 0.9335 -
accuracy: 0.9777 - val_loss: 0.0815 - val_accuracy: 0.9759
Epoch 10/10
3150/3150 [==============================] - 20s 6ms/step - loss: 1.1241 -
accuracy: 0.9752 - val_loss: 0.0975 - val_accuracy: 0.9753
```

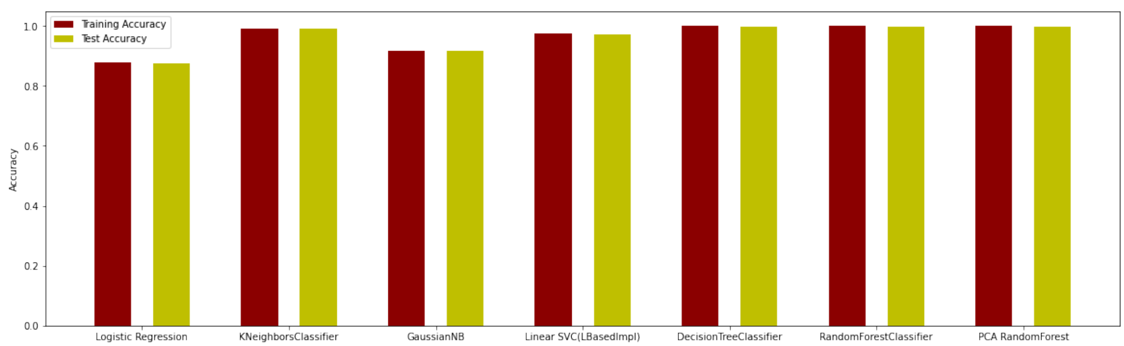
[36]:
```
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('SCCE Loss')
plt.legend()
plt.grid(True)
```



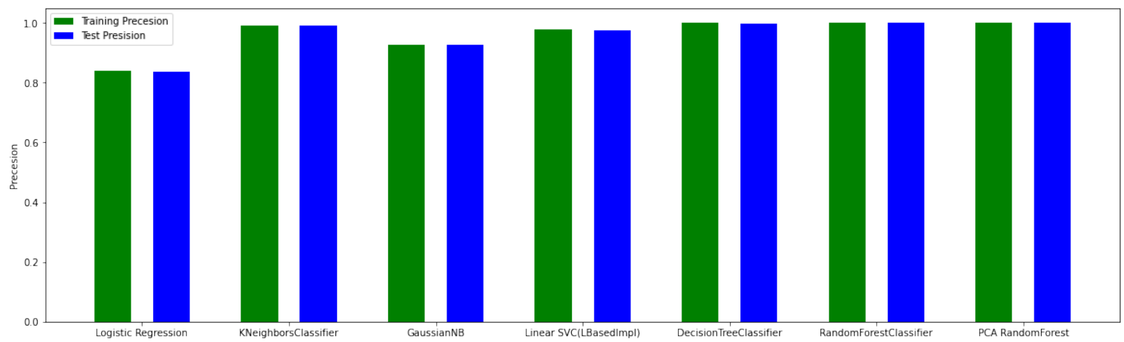

[37]:
```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
```



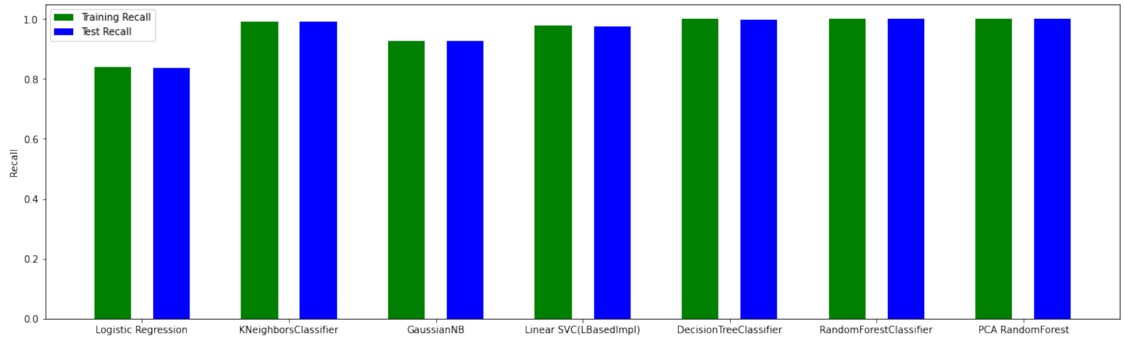

[38]:
```python
keys = [key for key in kernal_evals.keys()]
values = [value for value in kernal_evals.values()]
fig, ax = plt.subplots(figsize=(20, 6))
ax.bar(np.arange(len(keys)) - 0.2, [value[0] for value in values],␣
 ↪color='darkred', width=0.25, align='center')
ax.bar(np.arange(len(keys)) + 0.2, [value[1] for value in values], color='y',␣
 ↪width=0.25, align='center')
ax.legend(["Training Accuracy", "Test Accuracy"])
ax.set_xticklabels(keys)
ax.set_xticks(np.arange(len(keys)))
plt.ylabel("Accuracy")
plt.show()
```



[39]:
```
keys = [key for key in kernal_evals.keys()]
values = [value for value in kernal_evals.values()]
fig, ax = plt.subplots(figsize=(20, 6))
ax.bar(np.arange(len(keys)) - 0.2, [value[2] for value in values], color='g',␣
↪width=0.25, align='center')
ax.bar(np.arange(len(keys)) + 0.2, [value[3] for value in values], color='b',␣
↪width=0.25, align='center')
ax.legend(["Training Precesion", "Test Presision"])
ax.set_xticklabels(keys)
ax.set_xticks(np.arange(len(keys)))
plt.ylabel("Precesion")
plt.show()
```




[40]:
```
keys = [key for key in kernal_evals.keys()]
values = [value for value in kernal_evals.values()]
fig, ax = plt.subplots(figsize=(20, 6))
ax.bar(np.arange(len(keys)) - 0.2, [value[2] for value in values], color='g',␣
↪width=0.25, align='center')
ax.bar(np.arange(len(keys)) + 0.2, [value[3] for value in values], color='b',␣
↪width=0.25, align='center')
ax.legend(["Training Recall", "Test Recall"])
ax.set_xticklabels(keys)
ax.set_xticks(np.arange(len(keys)))
plt.ylabel("Recall")
plt.show()
```


Training Recall
Test Recall
Recall
1.0
0.8
0.6
0.4
0.2
0.0
Logistic Regression
KNeighborsClassifier
GaussianNB
Linear SVC(LBasedImpl)
DecisionTreeClassifier
RandomForestClassifier
PCA RandomForest