INTRODUCTION

The configuration manual plays an important role on information about all the hardware, software and procedures used in the implementation of this project.

Specifications of the system

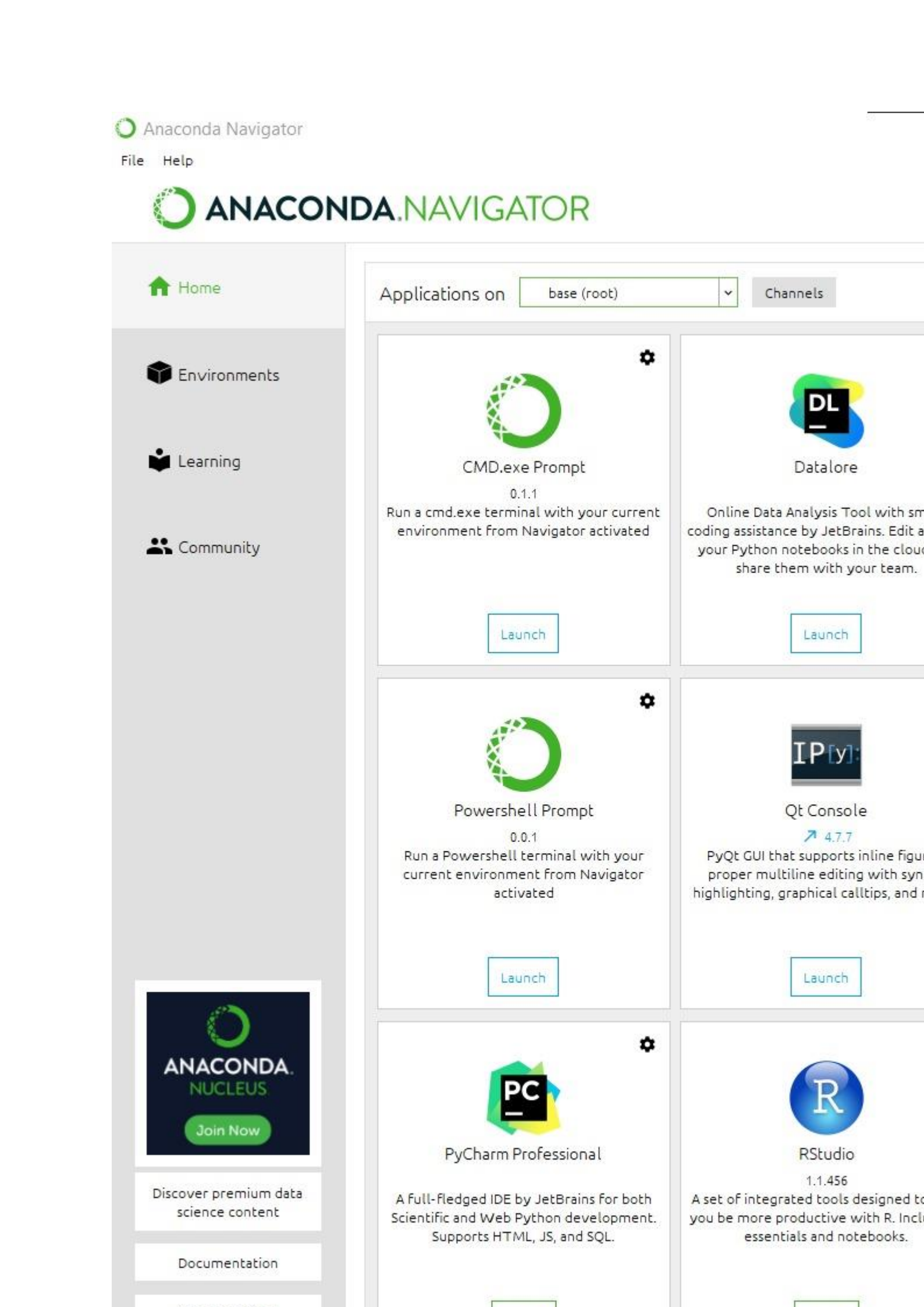
The system specifications are as follow ;

- Processor ; Intel Core i7, 7th generation @ 2.24 GHz
- GPU; Nvidia Geforce 6GB
- RAM ; 16 GB
- SSD ; 1 TB
- Operating System ; Windows 10 professional

Software Tools

The software and tools are used in the implantation of this work are as follows ;

- Anaconda Navigator
- Python
- Spyder
- Origin


Anaconda Navigator
File Help
ANACONDA.NAVIGATOR
Home
Environments
Learning
Community
Applications on
base (root)
Channels
CMD.exe Prompt
0.1.1
Run a cmd.exe terminal with your current environment from Navigator activated
Launch
Datalore
Online Data Analysis Tool with sm
coding assistance by JetBrains. Edit a
your Python notebooks in the cloud
share them with your team.
Launch
Powershell Prompt
0.0.1
Run a Powershell terminal with your current environment from Navigator activated
Launch
Qt Console
4.7.7
PyQt GUI that supports inline figu
proper multiline editing with syn
highlighting, graphical calltips, and
Launch
PyCharm Professional
A full-fledged IDE by JetBrains for both Scientific and Web Python development. Supports HTML, JS, and SQL.
RStudio
1.1.456
A set of integrated tools designed to
you be more productive with R. Inclu
essentials and notebooks.
ANACONDA.
NUCLEUS
Join Now
Discover premium data science content
Documentation

Import Libraries

In [1]:

```
#Data loading and exploration libraries
import shutil
import numpy as np
from brothon import bro_log_reader as blr
import pandas as pd
import os
from tqdm import tqdm
import datetime

#Data Visualisation libraries
import seaborn as sns
from pylab import plot, show
import matplotlib.pyplot as plt
plt.style.use('ggplot')
from mlxtend.plotting import plot_confusion_matrix

#Data normalisation librabry
from sklearn.preprocessing import MinMaxScaler, StandardScaler

#Machine learning libraries
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import VotingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

# Deep Learning Libraries
from keras.layers import LSTM, Activation, Dense, Dropout, Input, Embedding, BatchNormalization
from keras.optimizers import RMSprop
from keras.preprocessing import sequence
```

arch

12°C

```
# Deep Learning Libraries
from keras.layers import LSTM, Activation, Dense, Dropout, Input, Embedding, BatchNormalization
from keras.optimizers import RMSprop
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.models import Model
from keras.layers import Input
from keras.layers import MaxPooling2D
from keras.layers import SeparableConv2D, Dense
import tensorflow as tf

#Data processing libraries
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import RFE
from sklearn.feature_selection import RFECV
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.metrics import precision_recall_fscore_support as score
from sklearn.metrics import accuracy_score
```

Figure 1 import of Library

Load Data

In [2]:
```
df = pd.read_csv('dataset.csv', low_memory = False)
```

In [3]:
```
df.shape
```

Out[3]: (20000000, 21)

In [4]:
```
df.head()
```

Out[4]:

	ts	uid	id.orig_h	id.orig_p	id.resp_h	id.resp_p	proto	service	duration	orig_bytes	...	conn_sta
0	2018-09-07 06:48:35.086226	CjKfOn3HBY0Q9XSbu4	192.168.100.111	18088	212.38.19.44	80	tcp	-	0 days 00:00:00.000002000	0	...	
1	2018-09-06 19:55:08.347506	CTwFrA37B3cqldNIpe	192.168.100.111	18088	212.245.120.75	80	tcp	-	0 days 00:00:00.000002000	0	...	
2	2018-09-07 01:01:16.193973	CJrhp749zn0Bo92QFe	192.168.100.111	17576	4.184.224.179	8081	tcp	-	0 days 00:00:00.000002000	0	...	
3	2018-09-07 01:01:38.448529	CZkOCF4t1R8RT6GMh9	192.168.100.111	17576	212.191.14.110	8081	tcp	-	0 days 00:00:00.000002000	0	...	
4	2018-09-07 01:15:03.956982	COad3U3ND2EjUA5nGg	192.168.100.111	18088	35.248.6.170	80	tcp	-	0 days 00:00:00.000002000	0	...	

5 rows × 21 columns

In [5]:
```
# timestamp (ts) and unqiue id (uid) as these columns are not needed
df=df.drop(columns=['ts', 'uid'])
```

In [6]:
```
# Replace '-' with NaN. In this case, '-' differs from 0 as the latter represents an actual value
```

Figure 2 ; Loading of Data

In [6]:
```
df=df.drop(columns=['ts', 'uid'])

# Replace '-' with NaN. In this case, '-' differs from 0 as the latter represents an actual value
df['service'] = df['service'].replace('-', np.NaN)
df['local_orig'] = df['local_orig'].replace('-', np.NaN)
df['local_resp'] = df['local_resp'].replace('-', np.NaN)
```

Checking for Null values

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
id.orig_h              0
id.orig_p              0
id.resp_h              0
id.resp_p              0
proto                  0
service          19991921
duration               0
orig_bytes             0
resp_bytes             0
conn_state             0
local_orig      20000000
local_resp      20000000
missed_bytes           0
history                0
orig_pkts              0
orig_ip_bytes          0
resp_pkts              0
resp_ip_bytes          0
label                  0
dtype: int64
```

Figure 3 Checking for Null Values

```
In [8]: # Drop columns with null values
        df=df.drop(columns=['service', 'local_orig', 'local_resp'])
```

Variable Encoding

```
In [9]: df["id.orig_h"] = df["id.orig_h"].astype('category').cat.codes
        df["id.resp_h"] = df["id.resp_h"].astype('category').cat.codes
        df["proto"] = df["proto"].astype('category').cat.codes
        df["duration"] = df["duration"].astype('category').cat.codes
        df["orig_bytes"] = df["orig_bytes"].astype('category').cat.codes
        df["resp_bytes"] = df["resp_bytes"].astype('category').cat.codes
        df["conn_state"] = df["conn_state"].astype('category').cat.codes
        df["history"] = df["history"].astype('category').cat.codes
        df["label"] = df["label"].astype('category').cat.codes
```

```
In [10]: labels = ['Port Scan', 'DDoS', 'Benign', 'Okiru']
```

```
In [11]: df.shape
Out[11]: (20000000, 16)
```

Univariate Selection

```
In [12]: X = df.iloc[:,0:16]
         y = df.iloc[:,-1]
```

```
In [13]: #apply SelectKBest class to extract top 10 best features
         bestfeatures = SelectKBest(score_func=chi2, k=10)
         fit = bestfeatures.fit(X,y)
```

Figure 4; Variable Encoding

Univariate Selection

```
In [12]: X = df.iloc[:,0:16]
         y = df.iloc[:,-1]
```

```
In [13]: #apply SelectKBest class to extract top 10 best features
         bestfeatures = SelectKBest(score_func=chi2, k=10)
         fit = bestfeatures.fit(X,y)
         dfscores = pd.DataFrame(fit.scores_)
         dfcolumns = pd.DataFrame(X.columns)
         #concat two dataframes for better visualization
         featureScores = pd.concat([dfcolumns,dfscores],axis=1)
         featureScores.columns = ['Features','Score']  #naming the dataframe columns
         print(featureScores.nlargest(10,'Score'))  #print 10 best features
         top10 = featureScores.nlargest(10,'Score')
```

```
          Features         Score
2       id.resp_h  1.135168e+12
5        duration  1.041933e+12
3       id.resp_p  3.615433e+11
1       id.orig_p  1.161968e+11
6      orig_bytes  4.343725e+08
12  orig_ip_bytes  8.281666e+07
15          label  1.666667e+07
11      orig_pkts  2.448942e+06
14  resp_ip_bytes  2.378789e+06
7      resp_bytes  7.474695e+05
```

```
In [14]: col = top10.Features.to_numpy()
```

```
In [15]: df[col]
         df_final = df[col]
```

Figure 5 ; Unvariable Selection

```
14  resp_ip_bytes  2.378789e+06
7       resp_bytes  7.474695e+05
```

In [14]:
```
col = top10.Features.to_numpy()
```

In [15]:
```
df[col]
df_final = df[col]
```

Spliting Dataset into 70% Training and 30% Testing

In [37]:
```
X = df_final.iloc[:,0:10]
y = df.iloc[:,-1]
```

In [38]:
```
standardscaler = MinMaxScaler()
X= standardscaler.fit_transform(X)
```

In [18]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state = 123)
```

Machine Learning Algorithm

Support Vector Machine Classifier

In [19]:
```
svm=LinearSVC(random_state = 123, tol=1e-5)
svm= svm.fit(X_train , y_train)
svm
```

```
/Users/remi/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:976: ConvergenceWarning: Liblinear failed to conv
```


Machine Learning Algorithm

Support Vector Machine Classifier

In [19]:
```
svm=LinearSVC(random_state = 123, tol=1e-5)
svm= svm.fit(X_train , y_train)
svm
```


```
/Users/remi/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:976: ConvergenceWarning: Liblinear failed to conv
erge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

Out[19]:
```
LinearSVC(random_state=123)
```

In [20]:
```
predSVM = svm.predict(X_test)
svmScore=svm.score(X_test, y_test)
print('Accuracy score= {:.4f}'.format(svmScore))
```

```
Accuracy score= 0.9892
```

Confusion Matrix for SVM

In [21]:
```
print("confusion matrix")
cmSVM=confusion_matrix(y_test, predSVM)
fig, ax = plot_confusion_matrix(conf_mat=cmSVM,figsize=(10, 10),
                                show_absolute=True,
                                show_normed=True,
                                colorbar=True)
ax.set_xticklabels(['''] + labels)
ax.set_yticklabels(['''] + labels)
plt.show()
```

Figure 6 ;Inputing Machine Learning Algorithm

Figure 7 ; Confusion Matrix

Classification Report for SVM

In [22]:
```
crSVM=classification_report(y_test, predSVM, digits =4, target_names= labels)
print(crSVM)
```

```
              precision    recall  f1-score   support

   Port Scan     0.9611    0.9998    0.9801   1499821
        DDoS     0.9998    0.9569    0.9779   1500097
      Benign     1.0000    1.0000    1.0000   1500072
       Okiru     0.9973    1.0000    0.9987   1500010

    accuracy                         0.9892   6000000
   macro avg     0.9896    0.9892    0.9892   6000000
weighted avg     0.9896    0.9892    0.9892   6000000
```

Random Forest Classifier

In [49]:
```
rf= RandomForestClassifier(n_estimators = 20, max_depth=10, max_leaf_nodes = 2, random_state =123)
rf= rf.fit(X_train, y_train)
rf
```

Out[49]:
```
RandomForestClassifier(max_depth=10, max_leaf_nodes=2, n_estimators=20,
                       random_state=123)
```

```
rf= RandomForestClassifier(n_estimators=50, max_depth=10, random_state =123, max_leaf_nodes=5)
rf= rf.fit(X_train, y_train)
rf
```

In [50]:
```
predRF = rf.predict(X_test)
rfScore=rf.score(X test, y test)
```

Figure 8; Classification Report for SVM

Confusion Matrix for Random Forest Classifier

In [52]:
```
print("confusion matrix")
cmRF=confusion_matrix(y_test, predRF)
fig, ax = plot_confusion_matrix(conf_mat=cmRF,figsize=(10, 10),
                                show_absolute=True,
                                show_normed=True,
                                colorbar=True)
ax.set_xticklabels(['''] + labels)
ax.set_yticklabels(['''] + labels)
plt.show()
```

confusion matrix

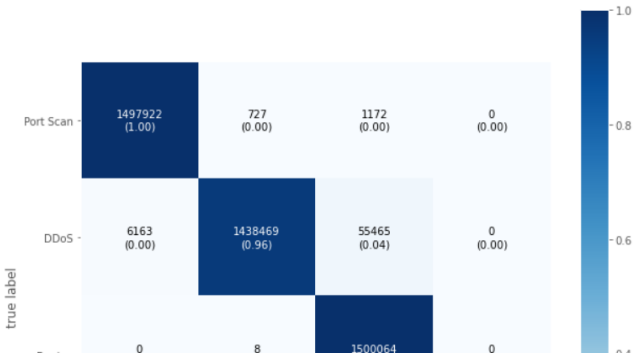


Figure 9 ; Confusion Matrix for Random Forest Classifier

Classification Report for Random Forest Classifier

In [53]:
```
crRF=classification_report(y_test, predRF, digits =4, target_names= labels )
print(crRF)
```

```
              precision    recall  f1-score   support

   Port Scan     0.9959    0.9987    0.9973   1499821
        DDoS     0.9995    0.9589    0.9788   1500097
      Benign     0.9636    1.0000    0.9815   1500072
       Okiru     1.0000    1.0000    1.0000   1500010

    accuracy                         0.9894   6000000
   macro avg     0.9898    0.9894    0.9894   6000000
weighted avg     0.9898    0.9894    0.9894   6000000
```

Deep Learning Algorithms

In [54]:
```
model = Sequential()
model.add(Dense(128, input_dim = 10, name = 'input', activation = 'relu'))
model.add(Dense(64, activation = 'sigmoid', name= 'dense1'))
#model.add(Dense(64, activation="relu", name="dense3"))
#model.add(Dropout(0.5))
model.add(Dense(64, activation="sigmoid", name="dense2"))
#model.add(Dropout(0.5))
#model.add(BatchNormalization())
```

Figure 10 ; Classification Report for Random Forest classifier

Deep Learning Algorithms

In [54]:
```
model = Sequential()
model.add(Dense(128, input_dim = 10, name = 'input', activation = 'relu'))
model.add(Dense(64, activation = 'sigmoid', name= 'dense1'))
#model.add(Dense(64, activation="relu", name="dense3"))
#model.add(Dropout(0.5))
model.add(Dense(64, activation="sigmoid", name="dense2"))
#model.add(Dropout(0.5))
#model.add(BatchNormalization())
model.add(Dense(4, activation="softmax", name="out"))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input (Dense)                (None, 128)               1408
_________________________________________________________________
dense1 (Dense)               (None, 64)                8256
_________________________________________________________________
dense2 (Dense)               (None, 64)                4160
_________________________________________________________________
out (Dense)                  (None, 4)                 260
=================================================================
Total params: 14,084
Trainable params: 14,084
Non-trainable params: 0
_________________________________________________________________
```

In [55]:
```
early_stopping = tf.keras.callbacks.EarlyStopping(patience=2)
```

```
_________________________________________________________________
```

In [55]:
```
early_stopping = tf.keras.callbacks.EarlyStopping(patience=2)
```

In [56]:
```
from keras.optimizers import SGD
sgd = SGD(lr=0.001, decay=1e-6, momentum=0.9, nesterov=True)
```

In [57]:
```
model.compile(optimizer = sgd, loss='sparse_categorical_crossentropy', metrics = ['accuracy'])
```

In [58]:
```
history = model.fit(X_train, y_train, batch_size = 10000, epochs = 5,
                    validation_split = 0.3, callbacks=[early_stopping])
```

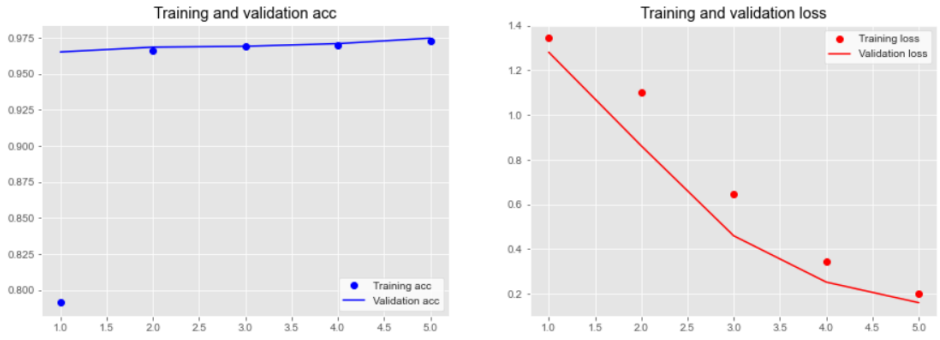
```
Epoch 1/5
980/980 [==============================] - 12s 12ms/step - loss: 1.3441 - accuracy: 0.7913 - val_loss: 1.2809 - val_accurac
y: 0.9652
Epoch 2/5
980/980 [==============================] - 12s 12ms/step - loss: 1.1015 - accuracy: 0.9662 - val_loss: 0.8619 - val_accurac
y: 0.9685
Epoch 3/5
980/980 [==============================] - 11s 12ms/step - loss: 0.6434 - accuracy: 0.9689 - val_loss: 0.4587 - val_accurac
y: 0.9691
Epoch 4/5
980/980 [==============================] - 11s 12ms/step - loss: 0.3423 - accuracy: 0.9697 - val_loss: 0.2515 - val_accurac
y: 0.9710
Epoch 5/5
980/980 [==============================] - 12s 12ms/step - loss: 0.1993 - accuracy: 0.9727 - val_loss: 0.1593 - val_accurac
y: 0.9748
```

Figure 11 ; Deep learning Algorithms

Loss and Accuracy Plots

In [60]:
```
# Visualize the result
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))
sns.set_style("white")
plt.suptitle('Train history', size = 15)

ax1.plot(epochs, acc, "bo", label = "Training acc")
ax1.plot(epochs, val_acc, "b", label = "Validation acc")
ax1.set_title("Training and validation acc")
ax1.legend()

ax2.plot(epochs, loss, "bo", label = "Training loss", color = 'red')
ax2.plot(epochs, val_loss, "b", label = "Validation loss", color = 'red')
ax2.set_title("Training and validation loss")
ax2.legend()

plt.show()
```

Figure 12; Loss and Accuracy plots




```
In [61]: accr = model.evaluate(X_test,y_test)
```

```
187500/187500 [==============================] - 142s 759us/step - loss: 0.1596 - accuracy: 0.9747
```

```
In [62]: predNN = model.predict(X_test)
         print('Test set\n   Accuracy: {:0.4f}'.format(accr[1]))
```

```
Test set
   Accuracy: 0.9747
```

```
In [63]: predNN = np.argmax(predNN, 1)
```

Confusion Matrix for Neural Network

```
In [64]: print("confusion matrix")
         CM=confusion_matrix(y_test, predNN)
         fig, ax = plot_confusion_matrix(conf_mat=CM,figsize=(10, 10),
                                         show_absolute=True,
                                         show_normed=True,
                                         colorbar=True)
         ax.set_xticklabels([''] + labels)
         ax.set_yticklabels([''] + labels)
         plt.show()
```

```
confusion matrix
```

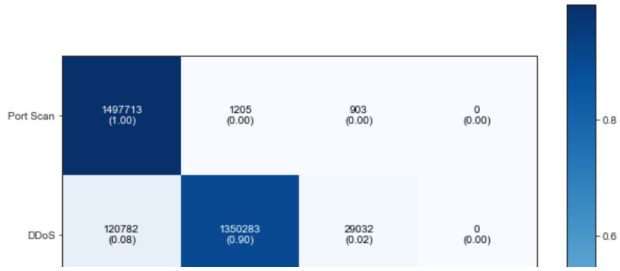


Figure 13

Classification Report for Neural Network

```
In [65]: CR=classification_report(y_test, predNN, digits = 4, target_names = labels)
         print(CR)
```

```
              precision    recall  f1-score   support

   Port Scan     0.9253    0.9986    0.9606   1499821
        DDoS     0.9991    0.9001    0.9470   1500097
      Benign     0.9804    1.0000    0.9901   1500072
       Okiru     1.0000    0.9999    1.0000   1500010

    accuracy                         0.9747   6000000
   macro avg     0.9762    0.9747    0.9744   6000000
weighted avg     0.9762    0.9747    0.9744   6000000
```

Neural Network Model 2

```
In [73]: model2 = Sequential()
         model2.add(Dense(256, input_dim=X.shape[1], kernel_initializer='he_uniform', activation='relu'))
         model2.add(Dense(128, activation='sigmoid'))
         model2.add(Dense(128, activation='sigmoid'))
         model2.add(Dense(128, activation='sigmoid'))
```

fig 13; Confusion Manual on Neutral network

Neural Network Model 2

In [73]:
```
model2 = Sequential()
model2.add(Dense(256, input_dim=X.shape[1], kernel_initializer='he_uniform', activation='relu'))
model2.add(Dense(128, activation='sigmoid'))
model2.add(Dense(128, activation='sigmoid'))
model2.add(Dense(128, activation='sigmoid'))
model2.add(Dense(4, activation='softmax'))
model2.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 256)               2816
_________________________________________________________________
dense_11 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_12 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_13 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_14 (Dense)             (None, 4)                 516
=================================================================
Total params: 69,252
Trainable params: 69,252
Non-trainable params: 0
_________________________________________________________________
```

In [74]:
```
model2.compile(optimizer = sgd, loss='sparse_categorical_crossentropy', metrics = ['accuracy'])
```

In [75]:
```
history2 = model2.fit(X_train, y_train, batch_size = 10000, epochs = 10,
                      validation_split = 0.3, callbacks=[early_stopping])
```

```
Epoch 1/10
980/980 [==============================] - 34s 34ms/step - loss: 1.3355 - accuracy: 0.7806 - val_loss: 1.2552 - val_accurac
y: 0.9652
Epoch 2/10
980/980 [==============================] - 34s 34ms/step - loss: 1.0362 - accuracy: 0.9639 - val_loss: 0.7582 - val_accurac
y: 0.9652
Epoch 3/10
980/980 [==============================] - 34s 35ms/step - loss: 0.5336 - accuracy: 0.9661 - val_loss: 0.3610 - val_accurac
y: 0.9697
Epoch 4/10
980/980 [==============================] - 34s 35ms/step - loss: 0.2559 - accuracy: 0.9744 - val_loss: 0.1781 - val_accurac
y: 0.9780
Epoch 5/10
980/980 [==============================] - 34s 35ms/step - loss: 0.1425 - accuracy: 0.9796 - val_loss: 0.1175 - val_accurac
y: 0.9809
Epoch 6/10
980/980 [==============================] - 34s 35ms/step - loss: 0.1035 - accuracy: 0.9818 - val_loss: 0.0921 - val_accurac
y: 0.9827
Epoch 7/10
980/980 [==============================] - 34s 35ms/step - loss: 0.0843 - accuracy: 0.9834 - val_loss: 0.0772 - val_accurac
y: 0.9841
Epoch 8/10
```

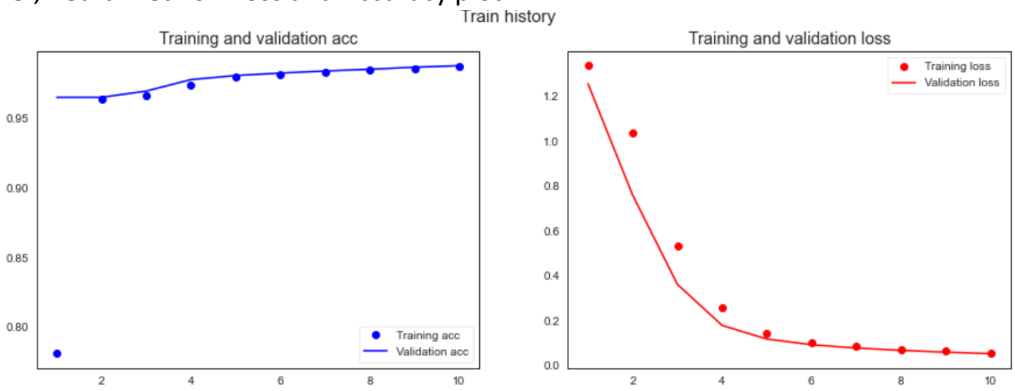
Figure 14 ; neural network 2

Loss and Accuracy Plot

In [76]:
```
# Visualize the result
acc = history2.history['accuracy']
val_acc = history2.history['val_accuracy']
loss = history2.history['loss']
val_loss = history2.history['val_loss']

epochs = range(1, len(acc) + 1)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))
sns.set_style("white")
plt.suptitle('Train history', size = 15)

ax1.plot(epochs, acc, "bo", label = "Training acc")
ax1.plot(epochs, val_acc, "b", label = "Validation acc")
ax1.set_title("Training and validation acc")
ax1.legend()

ax2.plot(epochs, loss, "bo", label = "Training loss", color = 'red')
ax2.plot(epochs, val_loss, "b", label = "Validation loss", color = 'red')
ax2.set_title("Training and validation loss")
ax2.legend()

plt.show()
```

Train history

Figure 15 ; neural network loss and Accuracy plot




```
80]:  predNN2 = model2.predict(X_test)
```

```
Test set
   Accuracy: 0.9747
```

Classification Report

```
In [83]:  CR=classification_report(y_test, predNN2, digits = 4, target_names = labels)
          print(CR)
```

```
                precision    recall  f1-score   support

     Port Scan     0.9569    0.9977    0.9769   1499821
          DDoS     0.9975    0.9540    0.9753   1500097
        Benign     0.9989    1.0000    0.9994   1500072
         Okiru     1.0000    0.9999    1.0000   1500010

      accuracy                         0.9879   6000000
     macro avg     0.9883    0.9879    0.9879   6000000
  weighted avg     0.9883    0.9879    0.9879   6000000
```

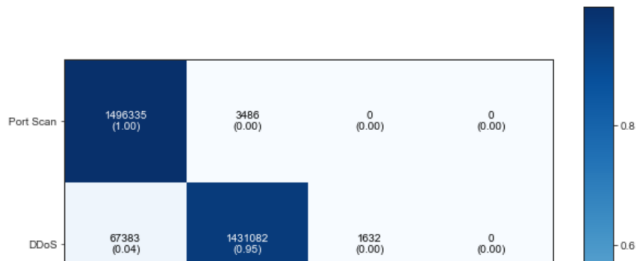
```
In [84]:  precSVM, recSVM, fscoreSVM, _ = score(y_test, predSVM,average='macro')
          accSVM = accuracy_score(y_test, predSVM)

          precRF, recRF, fscoreRF, _ = score(y_test, predRF,average='macro')
          accRF = accuracy_score(y_test, predRF)

          precNN, recNN, fscoreNN, _ = score(y_test, predNN,average='macro')
          accNN = accuracy_score(y_test, predNN)

          precNN2, recNN2, fscoreNN2, _ = score(y_test, predNN2,average='macro')
          accNN2 = accuracy_score(y_test, predNN2)
```

```
In [85]:  gr1 = [precSVM, recSVM, fscoreSVM, accSVM]
          gr2 = [precRF, recRF, fscoreRF, accRF]
          gr3 = [precNN, recNN, fscoreNN, accNN]
```

Confusion Matrix

```
In [82]:  print("confusion matrix")
          CM=confusion_matrix(y_test, predNN2)
          fig, ax = plot_confusion_matrix(conf_mat=CM,figsize=(10, 10),
                                          show_absolute=True,
                                          show_normed=True,
                                          colorbar=True)
          ax.set_xticklabels([''] + labels)
          ax.set_yticklabels([''] + labels)
          plt.show()
```

```
confusion matrix
```



Classification Report

In [83]:
```
CR=classification_report(y_test, predNN2, digits = 4, target_names = labels)
print(CR)
```

```
              precision    recall  f1-score   support

   Port Scan     0.9569    0.9977    0.9769   1499821
        DDoS     0.9975    0.9540    0.9753   1500097
      Benign     0.9989    1.0000    0.9994   1500072
       Okiru     1.0000    0.9999    1.0000   1500010

    accuracy                         0.9879   6000000
   macro avg     0.9883    0.9879    0.9879   6000000
weighted avg     0.9883    0.9879    0.9879   6000000
```

In [84]:
```
precSVM, recSVM, fscoreSVM, _ = score(y_test, predSVM,average='macro')
accSVM = accuracy_score(y_test, predSVM)

precRF, recRF, fscoreRF, _ = score(y_test, predRF,average='macro')
accRF = accuracy_score(y_test, predRF)

precNN, recNN, fscoreNN, _ = score(y_test, predNN,average='macro')
accNN = accuracy_score(y_test, predNN)

precNN2, recNN2, fscoreNN2, _ = score(y_test, predNN2,average='macro')
accNN2 = accuracy_score(y_test, predNN2)
```

In [85]:
```
gr1 = [precSVM, recSVM, fscoreSVM, accSVM]
gr2 = [precRF, recRF, fscoreRF, accRF]
gr3 = [precNN, recNN, fscoreNN, accNN]
```

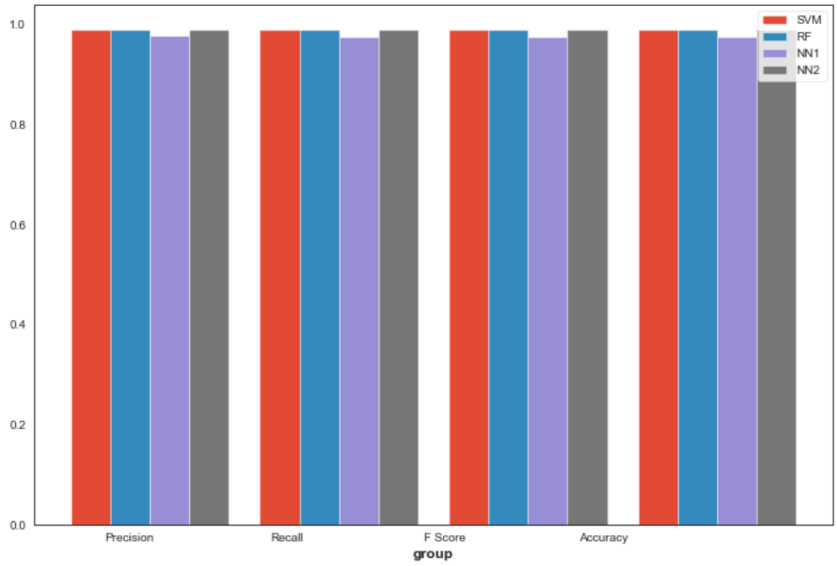
```
gr2 = [precRF, recRF, fscoreRF, accRF]
gr3 = [precNN, recNN, fscoreNN, accNN]
gr4 = [precNN2, recNN2, fscoreNN2, accNN2]
```

In [91]:

```
# set width of bar
barWidth = 0.25

# Set position of bar on X axis
r1 = np.arange(len(gr1))*1.2
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]
r4 = [x + barWidth for x in r3]

# Make the plot

plt.figure(figsize=[12,8])
plt.bar(r1, gr1, width=barWidth, edgecolor='white', label='SVM')
plt.bar(r2, gr2, width=barWidth, edgecolor='white', label='RF')
plt.bar(r3, gr3, width=barWidth, edgecolor='white', label='NN1')
plt.bar(r4, gr4, width=barWidth, edgecolor='white', label='NN2')

# Add xticks on the middle of the group bars
plt.xlabel('group', fontweight='bold')
plt.xticks([r + barWidth for r in range(len(gr1))], ['Precision', 'Recall', 'F Score', 'Accuracy'])

# Create legend & Show graphic
plt.legend()

plt.show()
```

Figure 17; Cl

```
plt.show()
```



In [ ]: