**Appendix** :

**Importing packages :**

Code cell <G16lEwlNjpTW>

# %% [code]

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

**Information about the dataset**

Code cell <cdqou1mZkAfu>

# %% [code]

data.info()

Execution output from 29 Aug 2024 17:55

6KB

Stream

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 15367 entries, 0 to 15366

Data columns (total 80 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Querylength 15367 non-null int64

1 domain\_token\_count 15367 non-null int64

2 path\_token\_count 15367 non-null int64

3 avgdomaintokenlen 15367 non-null float64

4 longdomaintokenlen 15367 non-null int64

5 avgpathtokenlen 15096 non-null float64

6 tld 15367 non-null int64

7 charcompvowels 15367 non-null int64

8 charcompace 15367 non-null int64

9 ldl\_url 15367 non-null int64

10 ldl\_domain 15367 non-null int64

11 ldl\_path 15367 non-null int64

12 ldl\_filename 15367 non-null int64

13 ldl\_getArg 15367 non-null int64

14 dld\_url 15367 non-null int64

15 dld\_domain 15367 non-null int64

16 dld\_path 15367 non-null int64

17 dld\_filename 15367 non-null int64

18 dld\_getArg 15367 non-null int64

19 urlLen 15367 non-null int64

20 domainlength 15367 non-null int64

21 pathLength 15367 non-null int64

22 subDirLen 15367 non-null int64

23 fileNameLen 15367 non-null int64

24 this.fileExtLen 15367 non-null int64

25 ArgLen 15367 non-null int64

26 pathurlRatio 15367 non-null float64

27 ArgUrlRatio 15367 non-null float64

28 argDomanRatio 15367 non-null float64

29 domainUrlRatio 15367 non-null float64

30 pathDomainRatio 15367 non-null float64

31 argPathRatio 15367 non-null float64

32 executable 15367 non-null int64

33 isPortEighty 15367 non-null int64

34 NumberofDotsinURL 15367 non-null int64

35 ISIpAddressInDomainName 15367 non-null int64

36 CharacterContinuityRate 15367 non-null float64

37 LongestVariableValue 15367 non-null int64

38 URL\_DigitCount 15367 non-null int64

39 host\_DigitCount 15367 non-null int64

40 Directory\_DigitCount 15367 non-null int64

41 File\_name\_DigitCount 15367 non-null int64

42 Extension\_DigitCount 15367 non-null int64

43 Query\_DigitCount 15367 non-null int64

44 URL\_Letter\_Count 15367 non-null int64

45 host\_letter\_count 15367 non-null int64

46 Directory\_LetterCount 15367 non-null int64

47 Filename\_LetterCount 15367 non-null int64

48 Extension\_LetterCount 15367 non-null int64

49 Query\_LetterCount 15367 non-null int64

50 LongestPathTokenLength 15367 non-null int64

51 Domain\_LongestWordLength 15367 non-null int64

52 Path\_LongestWordLength 15367 non-null int64

53 sub-Directory\_LongestWordLength 15367 non-null int64

54 Arguments\_LongestWordLength 15367 non-null int64

55 URL\_sensitiveWord 15367 non-null int64

56 URLQueries\_variable 15367 non-null int64

57 spcharUrl 15367 non-null int64

58 delimeter\_Domain 15367 non-null int64

59 delimeter\_path 15367 non-null int64

60 delimeter\_Count 15367 non-null int64

61 NumberRate\_URL 15367 non-null float64

62 NumberRate\_Domain 15367 non-null float64

63 NumberRate\_DirectoryName 15358 non-null float64

64 NumberRate\_FileName 15358 non-null float64

65 NumberRate\_Extension 8012 non-null float64

66 NumberRate\_AfterPath 15364 non-null float64

67 SymbolCount\_URL 15367 non-null int64

68 SymbolCount\_Domain 15367 non-null int64

69 SymbolCount\_Directoryname 15367 non-null int64

70 SymbolCount\_FileName 15367 non-null int64

71 SymbolCount\_Extension 15367 non-null int64

72 SymbolCount\_Afterpath 15367 non-null int64

73 Entropy\_URL 15367 non-null float64

74 Entropy\_Domain 15367 non-null float64

75 Entropy\_DirectoryName 13541 non-null float64

76 Entropy\_Filename 15177 non-null float64

77 Entropy\_Extension 15364 non-null float64

78 Entropy\_Afterpath 15364 non-null float64

79 URL\_Type\_obf\_Type 15367 non-null object

dtypes: float64(21), int64(58), object(1)

memory usage: 9.4+ MB

**Instantiate the model : Decision tree with max depth 3,5,7,10 and with and without feature importance :**

Code cell <QNrQuMJMlxQ0>

# %% [code]

# Creating holders to store the model performance results

ML\_Model = []

acc\_train = []

acc\_test = []

#function to call for storing the results

def storeResults(model, a,b):

ML\_Model.append(model)

acc\_train.append(round(a, 3))

acc\_test.append(round(b, 3))

Text cell <q7Dswr8El2sO>

# %% [markdown]

# Decision tree with max depth 3, 5, 7, 10

Code cell <KHkCZWnIl2QV>

# %% [code]

import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

# Lists to store accuracies

train\_accuracies = []

test\_accuracies = []

depths = [3, 5, 7, 10]

# Loop through each depth

for depth in depths:

# Instantiate and train the model

tree = DecisionTreeClassifier(max\_depth=depth)

tree.fit(X\_train, y\_train)

# Predict for both train and test sets

y\_train\_pred = tree.predict(X\_train)

y\_test\_pred = tree.predict(X\_test)

# Calculate accuracies

acc\_train\_tree = accuracy\_score(y\_train, y\_train\_pred)

acc\_test\_tree = accuracy\_score(y\_test, y\_test\_pred)

# Append the accuracies to the lists

train\_accuracies.append(acc\_train\_tree)

test\_accuracies.append(acc\_test\_tree)

# Plotting the results

plt.figure(figsize=(10, 6))

plt.plot(depths, train\_accuracies, label='Training Accuracy', marker='o', color='purple')

plt.plot(depths, test\_accuracies, label='Test Accuracy', marker='o', color='green')

plt.title('Decision Tree Accuracy vs. Max Depth')

plt.xlabel('Max Depth')

plt.ylabel('Accuracy')

plt.xticks(depths)

plt.legend()

plt.grid(True)

plt.show()

Execution output from 29 Aug 2024 17:56

59KB

text/plain

<Figure size 1000x600 with 1 Axes>

Text cell <BCjKOAeJmDLU>

# %% [markdown]

In the above plot we can see that the accuracy is increasing as we are going more deep into the decision tree.

Code cell <tE0E7bkOmG2t>

# %% [code]

#checking the feature improtance in the model

plt.figure(figsize=(10,30))

n\_features = X\_train.shape[1]

plt.barh(range(n\_features), tree.feature\_importances\_, align='center')

plt.yticks(np.arange(n\_features), X\_train.columns)

plt.xlabel("Feature importance")

plt.ylabel("Feature")

plt.show()

Execution output from 29 Aug 2024 17:56

238KB

text/plain

<Figure size 1000x3000 with 1 Axes>

Code cell <DCPb19D3mKd->

# %% [code]

#storing the results. The below mentioned order of parameter passing is important.

#Caution: Execute only once to avoid duplications.

storeResults('Decision Tree', acc\_train\_tree, acc\_test\_tree)

Text cell <KcYsdPjmmNMG>

# %% [markdown]

# Applying Decision tree removing features which are not useful.

Code cell <tIMNiaUvmSmN>

# %% [code]

ydt = df['URL\_Type\_obf\_Type']

Xdt = df.drop(['URL\_Type\_obf\_Type','Querylength', 'domain\_token\_count', 'path\_token\_count',

'avgdomaintokenlen', 'longdomaintokenlen', 'avgpathtokenlen',

'charcompvowels', 'charcompace', 'ldl\_path', 'dld\_url', 'dld\_domain', 'dld\_path',

'dld\_filename', 'dld\_getArg', 'pathLength',

'subDirLen', 'fileNameLen', 'this.fileExtLen', 'ArgLen', 'pathurlRatio',

'ArgUrlRatio', 'argDomanRatio', 'domainUrlRatio', 'pathDomainRatio',

'argPathRatio', 'isPortEighty', 'NumberofDotsinURL',

'CharacterContinuityRate', 'LongestVariableValue', 'URL\_DigitCount',

'host\_DigitCount', 'Directory\_DigitCount', 'File\_name\_DigitCount',

'Extension\_DigitCount', 'Query\_DigitCount', 'URL\_Letter\_Count', 'Directory\_LetterCount', 'Query\_LetterCount', 'LongestPathTokenLength',

'Domain\_LongestWordLength', 'Path\_LongestWordLength',

'sub-Directory\_LongestWordLength', 'Arguments\_LongestWordLength',

'URL\_sensitiveWord',

'delimeter\_Domain', 'delimeter\_path', 'delimeter\_Count',

'NumberRate\_URL', 'NumberRate\_Domain',

'NumberRate\_FileName', 'NumberRate\_Extension', 'NumberRate\_AfterPath',

'SymbolCount\_URL', 'SymbolCount\_Domain',

'SymbolCount\_FileName', 'SymbolCount\_Extension',

'SymbolCount\_Afterpath', 'Entropy\_URL',

'Entropy\_DirectoryName', 'Entropy\_Filename', 'Entropy\_Extension',

'Entropy\_Afterpath',],axis=1)

Xdt.shape, ydt.shape

Execution output from 29 Aug 2024 17:56

0KB

text/plain

((15367, 14), (15367,))

Code cell <CfRtOWBzmchD>

# %% [code]

# Splitting the dataset into train and test sets: 80-20 split

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

Xdt\_train, Xdt\_test, ydt\_train, ydt\_test = train\_test\_split(Xdt, ydt,

test\_size = 0.2, random\_state = 12)

Xdt\_train.shape, Xdt\_test.shape

Execution output from 29 Aug 2024 17:56

0KB

text/plain

((12293, 14), (3074, 14))

Code cell <QHCg\_QHnmdde>

# %% [code]

# Decision Tree model

from sklearn.tree import DecisionTreeClassifier

# instantiate the model

treedt = DecisionTreeClassifier(max\_depth = 5)

# fit the model

tree.fit(Xdt\_train, ydt\_train)

Execution output from 29 Aug 2024 17:56

5KB

text/plain

DecisionTreeClassifier(max\_depth=10)

Code cell <DTA16wvCmnWe>

# %% [code]

#predicting the target value from the model for the samples

ydt\_test\_tree = tree.predict(Xdt\_test)

ydt\_train\_tree = tree.predict(Xdt\_train)

Code cell <A9Q5YUTUmrfz>

# %% [code]

#computing the accuracy of the model performance

dtacc\_train\_tree = accuracy\_score(ydt\_train,ydt\_train\_tree)

dtacc\_test\_tree = accuracy\_score(ydt\_test,ydt\_test\_tree)

print("Decision Tree: Accuracy on training Data: {:.3f}".format(dtacc\_train\_tree))

print("Decision Tree: Accuracy on test Data: {:.3f}".format(dtacc\_test\_tree))

Execution output from 29 Aug 2024 17:56

0KB

Stream

Decision Tree: Accuracy on training Data: 0.978

Decision Tree: Accuracy on test Data: 0.966

**Instantiate the model : XGBoost model**

xgb = XGBClassifier(learning\_rate=0.4,max\_depth=7)

#fit the model

xgb.fit(X\_train, y\_train)

Execution output from 29 Aug 2024 17:56

7KB

text/plain

XGBClassifier(base\_score=None, booster=None, callbacks=None,

colsample\_bylevel=None, colsample\_bynode=None,

colsample\_bytree=None, device=None, early\_stopping\_rounds=None,

enable\_categorical=False, eval\_metric=None, feature\_types=None,

gamma=None, grow\_policy=None, importance\_type=None,

interaction\_constraints=None, learning\_rate=0.4, max\_bin=None,

max\_cat\_threshold=None, max\_cat\_to\_onehot=None,

max\_delta\_step=None, max\_depth=7, max\_leaves=None,

min\_child\_weight=None, missing=nan, monotone\_constraints=None,

multi\_strategy=None, n\_estimators=None, n\_jobs=None,

num\_parallel\_tree=None, random\_state=None, ...)

Code cell <b31y4o\_FvG81>

# %% [code]

#predicting the target value from the model for the samples

y\_test\_xgb = xgb.predict(X\_test)

y\_train\_xgb = xgb.predict(X\_train)

Code cell <Z3gzDuf3vLy7>

# %% [code]

#computing the accuracy of the model performance

acc\_train\_xgb = accuracy\_score(y\_train,y\_train\_xgb)

acc\_test\_xgb = accuracy\_score(y\_test,y\_test\_xgb)

print("XGBoost: Accuracy on training Data: {:.3f}".format(acc\_train\_xgb))

print("XGBoost : Accuracy on test Data: {:.3f}".format(acc\_test\_xgb))

Execution output from 29 Aug 2024 17:56

0KB

Stream

XGBoost: Accuracy on training Data: 1.000

XGBoost : Accuracy on test Data: 0.987

Code cell <\_XCUsEzVvMqT>

# %% [code]

#storing the results. The below mentioned order of parameter passing is important.

#Caution: Execute only once to avoid duplications.

storeResults('XGBoost', acc\_train\_xgb, acc\_test\_xgb)

Text cell <v5iMP5TNvQ9Y>

# %% [markdown]

# Support Vector Machine

Code cell <4kKnTTC3vXNB>

# %% [code]

from sklearn.svm import SVC # import the SVC class from sklearn.svm

**Instantiate the model : SVM Model**

svm = SVC(kernel='linear', max\_iter=1000) # limiting the number of iterations

# fit the model

svm.fit(X\_train, y\_train)

Execution output from 29 Aug 2024 17:56

5KB

Stream

/usr/local/lib/python3.10/dist-packages/sklearn/svm/\_base.py:299: ConvergenceWarning: Solver terminated early (max\_iter=1000). Consider pre-processing your data with StandardScaler or MinMaxScaler.

warnings.warn(

text/plain

SVC(kernel='linear', max\_iter=1000)

Code cell <2YMyzUwIve-N>

# %% [code]

#predicting the target value from the model for the samples

y\_test\_svm = svm.predict(X\_test)

y\_train\_svm = svm.predict(X\_train)

Code cell <FemfPyrBvjgY>

# %% [code]

#computing the accuracy of the model performance

acc\_train\_svm = accuracy\_score(y\_train,y\_train\_svm)

acc\_test\_svm = accuracy\_score(y\_test,y\_test\_svm)

print("SVM: Accuracy on training Data: {:.3f}".format(acc\_train\_svm))

print("SVM : Accuracy on test Data: {:.3f}".format(acc\_test\_svm))

Execution output from 29 Aug 2024 17:56

0KB

Stream

SVM: Accuracy on training Data: 0.575

SVM : Accuracy on test Data: 0.569

Code cell <Q1Aift60vmzh>

# %% [code]

#storing the results. The below mentioned order of parameter passing is important.

storeResults('SVM', acc\_train\_svm, acc\_test\_svm)

Text cell <FgI\_r0slvqKW>

# %% [markdown]

**Instantiate the model : Ensembel Model**

Text cell <FSVovLERvxVU>

# %% [markdown]

We have used Decision tree, Random forest, XGBoost and SVM for ensembelled model.

Code cell <QlxH\_kmKvru4>

# %% [code]

**Instantiate the model : LSTM Model**

model = Sequential()

model.add(LSTM(50, activation='relu', input\_shape=(time\_steps, X\_train\_scaled.shape[2])))

model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

Code cell <anZqewS\_weLB>

# %% [code]

# Train the model

history = model.fit(X\_train\_scaled, y\_train, epochs=10, batch\_size=32, validation\_split=0.2)

Execution output from 29 Aug 2024 17:57

2KB

Stream

Epoch 1/10

308/308 [==============================] - 9s 10ms/step - loss: 0.2077 - accuracy: 0.9300 - val\_loss: 0.1238 - val\_accuracy: 0.9516

Epoch 2/10

308/308 [==============================] - 2s 8ms/step - loss: 0.0985 - accuracy: 0.9642 - val\_loss: 0.1073 - val\_accuracy: 0.9569

Epoch 3/10

308/308 [==============================] - 3s 8ms/step - loss: 0.0832 - accuracy: 0.9712 - val\_loss: 0.0950 - val\_accuracy: 0.9634

Epoch 4/10

308/308 [==============================] - 3s 10ms/step - loss: 0.0744 - accuracy: 0.9743 - val\_loss: 0.0971 - val\_accuracy: 0.9642

Epoch 5/10

308/308 [==============================] - 3s 10ms/step - loss: 0.0668 - accuracy: 0.9778 - val\_loss: 0.0855 - val\_accuracy: 0.9658

Epoch 6/10

308/308 [==============================] - 2s 7ms/step - loss: 0.0604 - accuracy: 0.9779 - val\_loss: 0.0847 - val\_accuracy: 0.9691

Epoch 7/10

308/308 [==============================] - 2s 7ms/step - loss: 0.0569 - accuracy: 0.9809 - val\_loss: 0.0771 - val\_accuracy: 0.9719

Epoch 8/10

308/308 [==============================] - 2s 6ms/step - loss: 0.0528 - accuracy: 0.9820 - val\_loss: 0.0760 - val\_accuracy: 0.9715

Epoch 9/10

308/308 [==============================] - 1s 4ms/step - loss: 0.0490 - accuracy: 0.9832 - val\_loss: 0.0765 - val\_accuracy: 0.9736

Epoch 10/10

308/308 [==============================] - 1s 3ms/step - loss: 0.0459 - accuracy: 0.9843 - val\_loss: 0.0720 - val\_accuracy: 0.9732

Code cell <xXj6B8rBwkHg>

# %% [code]

# Predict and evaluate

y\_pred\_lstm = model.predict(X\_test\_scaled)

y\_pred\_lstm\_classes = (y\_pred\_lstm > 0.5).astype(int)

accuracy\_lstm = accuracy\_score(y\_test, y\_pred\_lstm\_classes)

print(f'LSTM Model Accuracy: {accuracy\_lstm:.2f}')

Execution output from 29 Aug 2024 17:57

0KB

Stream

97/97 [==============================] - 0s 2ms/step

LSTM Model Accuracy: 0.98