**Phishing Website Detection by Machine Learning Techniques**

**1. Objective:**

A phishing website is a common social engineering method that mimics trustful uniform resource locators (URLs) and webpages. The objective of this project is to train machine learning models and deep neural nets on the dataset to predict phishing websites. Both phishing and benign URLs of websites are gathered to form a dataset and from them required URL and website content-based features are extracted. The performance level of each model is measures and compared.

**2. Loading Data:**

The dataset that is given is loaded.

In [1]:

*#importing basic packages*

**import** pandas **as** pd

**import** numpy **as** np

**import** seaborn **as** sns

**import** matplotlib.pyplot **as** plt

In [2]:

**import** os**,** types

**import** pandas **as** pd

**from** botocore.client **import** Config

**import** ibm\_boto3

**def** \_\_iter\_\_(self): **return** 0

*# @hidden\_cell*

*# The following code accesses a file in your IBM Cloud Object Storage. It includes your credentials.*

*# You might want to remove those credentials before you share the notebook.*

cos\_client **=** ibm\_boto3**.**client(service\_name**=**'s3',

ibm\_api\_key\_id**=**'trsqH-dXZ870ShVaUKIhFKXjYPq5sEjpPEwSeHiHSvoQ',

ibm\_auth\_endpoint**=**"https://iam.cloud.ibm.com/oidc/token",

config**=**Config(signature\_version**=**'oauth'),

endpoint\_url**=**'https://s3.private.us.cloud-object-storage.appdomain.cloud')

bucket **=** 'webphishingdetection-donotdelete-pr-icmjtvktnzli2s'

object\_key **=** 'dataset\_website.csv'

body **=** cos\_client**.**get\_object(Bucket**=**bucket,Key**=**object\_key)['Body']

*# add missing \_\_iter\_\_ method, so pandas accepts body as file-like object*

**if** **not** hasattr(body, "\_\_iter\_\_"): body**.**\_\_iter\_\_ **=** types**.**MethodType( \_\_iter\_\_, body )

data0 **=** pd**.**read\_csv(body)

data0**.**head()

Out[2]:

|  | **index** | **having\_IPhaving\_IP\_Address** | **URLURL\_Length** | **Shortining\_Service** | **having\_At\_Symbol** | **double\_slash\_redirecting** | **Prefix\_Suffix** | **having\_Sub\_Domain** | **SSLfinal\_State** | **Domain\_registeration\_length** | **...** | **popUpWidnow** | **Iframe** | **age\_of\_domain** | **DNSRecord** | **web\_traffic** | **Page\_Rank** | **Google\_Index** | **Links\_pointing\_to\_page** | **Statistical\_report** | **Result** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | -1 | 1 | 1 | 1 | -1 | -1 | -1 | -1 | -1 | ... | 1 | 1 | -1 | -1 | -1 | -1 | 1 | 1 | -1 | -1 |
| **1** | 2 | 1 | 1 | 1 | 1 | 1 | -1 | 0 | 1 | -1 | ... | 1 | 1 | -1 | -1 | 0 | -1 | 1 | 1 | 1 | -1 |
| **2** | 3 | 1 | 0 | 1 | 1 | 1 | -1 | -1 | -1 | -1 | ... | 1 | 1 | 1 | -1 | 1 | -1 | 1 | 0 | -1 | -1 |
| **3** | 4 | 1 | 0 | 1 | 1 | 1 | -1 | -1 | -1 | 1 | ... | 1 | 1 | -1 | -1 | 1 | -1 | 1 | -1 | 1 | -1 |
| **4** | 5 | 1 | 0 | -1 | 1 | 1 | -1 | 1 | 1 | -1 | ... | -1 | 1 | -1 | -1 | 0 | -1 | 1 | 1 | 1 | 1 |

5 rows × 32 columns

In [3]:

*#Loading the data*

*#data0 = pd.read\_csv('dataset\_website.csv')*

*#data0.head()*

**3. Familiarizing with Data**

In this step, few dataframe methods are used to look into the data and its features.

In [4]:

*#Checking the shape of the dataset*

data0**.**shape

Out[4]:

(11055, 32)

In [5]:

*#Listing the features of the dataset*

data0**.**columns

Out[5]:

Index(['index', 'having\_IPhaving\_IP\_Address', 'URLURL\_Length',

'Shortining\_Service', 'having\_At\_Symbol', 'double\_slash\_redirecting',

'Prefix\_Suffix', 'having\_Sub\_Domain', 'SSLfinal\_State',

'Domain\_registeration\_length', 'Favicon', 'port', 'HTTPS\_token',

'Request\_URL', 'URL\_of\_Anchor', 'Links\_in\_tags', 'SFH',

'Submitting\_to\_email', 'Abnormal\_URL', 'Redirect', 'on\_mouseover',

'RightClick', 'popUpWidnow', 'Iframe', 'age\_of\_domain', 'DNSRecord',

'web\_traffic', 'Page\_Rank', 'Google\_Index', 'Links\_pointing\_to\_page',

'Statistical\_report', 'Result'],

dtype='object')

In [6]:

*#Information about the dataset*

data0**.**info()

RangeIndex: 11055 entries, 0 to 11054

Data columns (total 32 columns):

# Column Non-Null Count Dtype

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

0 index 11055 non-null int64

1 having\_IPhaving\_IP\_Address 11055 non-null int64

2 URLURL\_Length 11055 non-null int64

3 Shortining\_Service 11055 non-null int64

4 having\_At\_Symbol 11055 non-null int64

5 double\_slash\_redirecting 11055 non-null int64

6 Prefix\_Suffix 11055 non-null int64

7 having\_Sub\_Domain 11055 non-null int64

8 SSLfinal\_State 11055 non-null int64

9 Domain\_registeration\_length 11055 non-null int64

10 Favicon 11055 non-null int64

11 port 11055 non-null int64

12 HTTPS\_token 11055 non-null int64

13 Request\_URL 11055 non-null int64

14 URL\_of\_Anchor 11055 non-null int64

15 Links\_in\_tags 11055 non-null int64

16 SFH 11055 non-null int64

17 Submitting\_to\_email 11055 non-null int64

18 Abnormal\_URL 11055 non-null int64

19 Redirect 11055 non-null int64

20 on\_mouseover 11055 non-null int64

21 RightClick 11055 non-null int64

22 popUpWidnow 11055 non-null int64

23 Iframe 11055 non-null int64

24 age\_of\_domain 11055 non-null int64

25 DNSRecord 11055 non-null int64

26 web\_traffic 11055 non-null int64

27 Page\_Rank 11055 non-null int64

28 Google\_Index 11055 non-null int64

29 Links\_pointing\_to\_page 11055 non-null int64

30 Statistical\_report 11055 non-null int64

31 Result 11055 non-null int64

dtypes: int64(32)

memory usage: 2.7 MB

**4. Visualizing the data**

Few plots and graphs are displayed to find how the data is distributed and the how features are related to each other.

In [7]:

*#Plotting the data distribution*

data0**.**hist(bins **=** 50,figsize **=** (15,15))

plt**.**show()

In [8]:

*#Correlation heatmap*

plt**.**figure(figsize**=**(15,13))

sns**.**heatmap(data0**.**corr())

plt**.**show()

**5. Data Preprocessing & EDA**

Here, we clean the data by applying data preprocesssing techniques and transform the data to use it in the models.

In [9]:

data0**.**describe()

Out[9]:

|  | **index** | **having\_IPhaving\_IP\_Address** | **URLURL\_Length** | **Shortining\_Service** | **having\_At\_Symbol** | **double\_slash\_redirecting** | **Prefix\_Suffix** | **having\_Sub\_Domain** | **SSLfinal\_State** | **Domain\_registeration\_length** | **...** | **popUpWidnow** | **Iframe** | **age\_of\_domain** | **DNSRecord** | **web\_traffic** | **Page\_Rank** | **Google\_Index** | **Links\_pointing\_to\_page** | **Statistical\_report** | **Result** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 11055.000000 | 11055.000000 | 11055.000000 | 11055.000000 | 11055.000000 | 11055.000000 | 11055.000000 | 11055.000000 | 11055.000000 | 11055.000000 | ... | 11055.000000 | 11055.000000 | 11055.000000 | 11055.000000 | 11055.000000 | 11055.000000 | 11055.000000 | 11055.000000 | 11055.000000 | 11055.000000 |
| **mean** | 5528.000000 | 0.313795 | -0.633198 | 0.738761 | 0.700588 | 0.741474 | -0.734962 | 0.063953 | 0.250927 | -0.336771 | ... | 0.613388 | 0.816915 | 0.061239 | 0.377114 | 0.287291 | -0.483673 | 0.721574 | 0.344007 | 0.719584 | 0.113885 |
| **std** | 3191.447947 | 0.949534 | 0.766095 | 0.673998 | 0.713598 | 0.671011 | 0.678139 | 0.817518 | 0.911892 | 0.941629 | ... | 0.789818 | 0.576784 | 0.998168 | 0.926209 | 0.827733 | 0.875289 | 0.692369 | 0.569944 | 0.694437 | 0.993539 |
| **min** | 1.000000 | -1.000000 | -1.000000 | -1.000000 | -1.000000 | -1.000000 | -1.000000 | -1.000000 | -1.000000 | -1.000000 | ... | -1.000000 | -1.000000 | -1.000000 | -1.000000 | -1.000000 | -1.000000 | -1.000000 | -1.000000 | -1.000000 | -1.000000 |
| **25%** | 2764.500000 | -1.000000 | -1.000000 | 1.000000 | 1.000000 | 1.000000 | -1.000000 | -1.000000 | -1.000000 | -1.000000 | ... | 1.000000 | 1.000000 | -1.000000 | -1.000000 | 0.000000 | -1.000000 | 1.000000 | 0.000000 | 1.000000 | -1.000000 |
| **50%** | 5528.000000 | 1.000000 | -1.000000 | 1.000000 | 1.000000 | 1.000000 | -1.000000 | 0.000000 | 1.000000 | -1.000000 | ... | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | -1.000000 | 1.000000 | 0.000000 | 1.000000 | 1.000000 |
| **75%** | 8291.500000 | 1.000000 | -1.000000 | 1.000000 | 1.000000 | 1.000000 | -1.000000 | 1.000000 | 1.000000 | 1.000000 | ... | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 |
| **max** | 11055.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | ... | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 |

8 rows × 32 columns

From data distribution graph and correlation matrix, we can conclude that the following features do not have much impact on the result:

* having\_Sub\_Domain
* Domain\_registeration\_length
* Favicon
* Request\_URL
* URL\_of\_Anchor
* Links\_in\_tags
* Submitting\_to\_email
* Redirect
* on\_mouseover
* RightClick
* age\_of\_domain
* web\_traffic
* Page\_Rank
* Google\_Index
* Links\_pointing\_to\_page

In [10]:

*#Removing the features which do not have much impact on Result*

data**=**data0**.**iloc[:,[1,2,3,4,5,6,12,20,21,22,23,24,25,30,31]]

data**.**head()

Out[10]:

|  | **having\_IPhaving\_IP\_Address** | **URLURL\_Length** | **Shortining\_Service** | **having\_At\_Symbol** | **double\_slash\_redirecting** | **Prefix\_Suffix** | **HTTPS\_token** | **on\_mouseover** | **RightClick** | **popUpWidnow** | **Iframe** | **age\_of\_domain** | **DNSRecord** | **Statistical\_report** | **Result** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | -1 | 1 | 1 | 1 | -1 | -1 | -1 | 1 | 1 | 1 | 1 | -1 | -1 | -1 | -1 |
| **1** | 1 | 1 | 1 | 1 | 1 | -1 | -1 | 1 | 1 | 1 | 1 | -1 | -1 | 1 | -1 |
| **2** | 1 | 0 | 1 | 1 | 1 | -1 | -1 | 1 | 1 | 1 | 1 | 1 | -1 | -1 | -1 |
| **3** | 1 | 0 | 1 | 1 | 1 | -1 | -1 | 1 | 1 | 1 | 1 | -1 | -1 | 1 | -1 |
| **4** | 1 | 0 | -1 | 1 | 1 | -1 | 1 | -1 | 1 | -1 | 1 | -1 | -1 | 1 | 1 |

In [11]:

data**.**info()

RangeIndex: 11055 entries, 0 to 11054

Data columns (total 15 columns):

# Column Non-Null Count Dtype

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

0 having\_IPhaving\_IP\_Address 11055 non-null int64

1 URLURL\_Length 11055 non-null int64

2 Shortining\_Service 11055 non-null int64

3 having\_At\_Symbol 11055 non-null int64

4 double\_slash\_redirecting 11055 non-null int64

5 Prefix\_Suffix 11055 non-null int64

6 HTTPS\_token 11055 non-null int64

7 on\_mouseover 11055 non-null int64

8 RightClick 11055 non-null int64

9 popUpWidnow 11055 non-null int64

10 Iframe 11055 non-null int64

11 age\_of\_domain 11055 non-null int64

12 DNSRecord 11055 non-null int64

13 Statistical\_report 11055 non-null int64

14 Result 11055 non-null int64

dtypes: int64(15)

memory usage: 1.3 MB

This leaves us with 13 features & a target column.

In [12]:

*#checking the data for null or missing values*

data**.**isnull()**.**sum()

Out[12]:

having\_IPhaving\_IP\_Address 0

URLURL\_Length 0

Shortining\_Service 0

having\_At\_Symbol 0

double\_slash\_redirecting 0

Prefix\_Suffix 0

HTTPS\_token 0

on\_mouseover 0

RightClick 0

popUpWidnow 0

Iframe 0

age\_of\_domain 0

DNSRecord 0

Statistical\_report 0

Result 0

dtype: int64

From the above execution, it is clear that the data doesnot have any missing values.

By this, the data is throughly preprocessed & is ready for training.

**6. Splitting the Data**

In [13]:

*# Sepratating & assigning features and target columns to X & y*

X**=**data**.**iloc[:,:13]

y**=**data**.**iloc[:,13]

X**.**shape, y**.**shape

Out[13]:

((11055, 13), (11055,))

In [31]:

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

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

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X**.**values, y**.**values, test\_size **=** 0.2, random\_state **=** 12)

X\_train**.**shape, X\_test**.**shape

Out[31]:

((8844, 13), (2211, 13))

**7. Machine Learning Models & Training**

From the dataset above, it is clear that this is a supervised machine learning task. There are two major types of supervised machine learning problems, called classification and regression.

This data set comes under classification problem, as the input URL is classified as phishing (1) or legitimate (0). The supervised machine learning models (classification) considered to train the dataset in this notebook are:

* XGBoost
* Decision Tree
* Random Forest
* Support Vector Machines

In [15]:

*#importing packages*

**from** sklearn.metrics **import** accuracy\_score

In [16]:

*# 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))

**7.1. XGBoost Classifier**

XGBoost is one of the most popular machine learning algorithms these days. XGBoost stands for eXtreme Gradient Boosting. Regardless of the type of prediction task at hand; regression or classification. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance.

In [32]:

*#XGBoost Classification model*

**from** xgboost **import** XGBClassifier

**import** warnings

warnings**.**filterwarnings("ignore", category**=**UserWarning)

*# instantiate the model*

xgb **=** XGBClassifier(learning\_rate**=**0.4,max\_depth**=**7,verbosity **=** 0)

*#fit the model*

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

Out[32]:

XGBClassifier(base\_score=0.5, booster='gbtree', colsample\_bylevel=1,

colsample\_bynode=1, colsample\_bytree=1, enable\_categorical=False,

gamma=0, gpu\_id=-1, importance\_type=None,

interaction\_constraints='', learning\_rate=0.4, max\_delta\_step=0,

max\_depth=7, min\_child\_weight=1, missing=nan,

monotone\_constraints='()', n\_estimators=100, n\_jobs=56,

num\_parallel\_tree=1, predictor='auto', random\_state=0,

reg\_alpha=0, reg\_lambda=1, scale\_pos\_weight=1, subsample=1,

tree\_method='exact', validate\_parameters=1, verbosity=0)

In [33]:

*#predicting the target value from the model for the samples*

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

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

**Performance Evaluation:**

In [34]:

*#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))

XGBoost: Accuracy on training Data: 0.913

XGBoost : Accuracy on test Data: 0.905

**Storing the results:**

In [35]:

*#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)

**7.2. Decision Tree Classifier**

Decision trees are widely used models for classification and regression tasks. Essentially, they learn a hierarchy of if/else questions, leading to a decision. Learning a decision tree means learning the sequence of if/else questions that gets us to the true answer most quickly.

In the machine learning setting, these questions are called tests (not to be confused with the test set, which is the data we use to test to see how generalizable our model is). To build a tree, the algorithm searches over all possible tests and finds the one that is most informative about the target variable.

In [17]:

*# Decision Tree model*

**from** sklearn.tree **import** DecisionTreeClassifier

*# instantiate the model*

tree **=** DecisionTreeClassifier(max\_depth **=** 5)

*# fit the model*

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

Out[17]:

DecisionTreeClassifier(max\_depth=5)

In [18]:

*#predicting the target value from the model for the samples*

y\_test\_tree **=** tree**.**predict(X\_test)

y\_train\_tree **=** tree**.**predict(X\_train)

**Performance Evaluation:**

In [19]:

*#computing the accuracy of the model performance*

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

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

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

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

Decision Tree: Accuracy on training Data: 0.898

Decision Tree: Accuracy on test Data: 0.894

In [20]:

*#checking the feature improtance in the model*

plt**.**figure(figsize**=**(9,7))

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()

**Storing the results:**

In [21]:

*#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)

**7.3. Random Forest Classifier**

Random forests for regression and classification are currently among the most widely used machine learning methods.A random forest is essentially a collection of decision trees, where each tree is slightly different from the others. The idea behind random forests is that each tree might do a relatively good job of predicting, but will likely overfit on part of the data.

If we build many trees, all of which work well and overfit in different ways, we can reduce the amount of overfitting by averaging their results. To build a random forest model, you need to decide on the number of trees to build (the n\_estimators parameter of RandomForestRegressor or RandomForestClassifier). They are very powerful, often work well without heavy tuning of the parameters, and don’t require scaling of the data.

In [22]:

*# Random Forest model*

**from** sklearn.ensemble **import** RandomForestClassifier

*# instantiate the model*

forest **=** RandomForestClassifier(max\_depth**=**5)

*# fit the model*

forest**.**fit(X\_train, y\_train)

Out[22]:

RandomForestClassifier(max\_depth=5)

In [23]:

*#predicting the target value from the model for the samples*

y\_test\_forest **=** forest**.**predict(X\_test)

y\_train\_forest **=** forest**.**predict(X\_train)

**Performance Evaluation:**

In [24]:

*#computing the accuracy of the model performance*

acc\_train\_forest **=** accuracy\_score(y\_train,y\_train\_forest)

acc\_test\_forest **=** accuracy\_score(y\_test,y\_test\_forest)

print("Random forest: Accuracy on training Data: {:.3f}"**.**format(acc\_train\_forest))

print("Random forest: Accuracy on test Data: {:.3f}"**.**format(acc\_test\_forest))

Random forest: Accuracy on training Data: 0.893

Random forest: Accuracy on test Data: 0.886

In [25]:

*#checking the feature improtance in the model*

plt**.**figure(figsize**=**(9,7))

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

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

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

plt**.**xlabel("Feature importance")

plt**.**ylabel("Feature")

plt**.**show()

**Storing the results:**

In [26]:

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

*#Caution: Execute only once to avoid duplications.*

storeResults('Random Forest', acc\_train\_forest, acc\_test\_forest)

**7.4. Support Vector Machines**

In machine learning, support-vector machines (SVMs, also support-vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier.

In [27]:

*#Support vector machine model*

**from** sklearn.svm **import** SVC

*# instantiate the model*

svm **=** SVC(kernel**=**'linear', C**=**1.0, random\_state**=**12)

*#fit the model*

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

Out[27]:

SVC(kernel='linear', random\_state=12)

In [28]:

*#predicting the target value from the model for the samples*

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

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

**Performance Evaluation:**

In [29]:

*#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))

SVM: Accuracy on training Data: 0.886

SVM : Accuracy on test Data: 0.883

**Storing the results:**

In [30]:

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

*#Caution: Execute only once to avoid duplications.*

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

**8. Comparision of Models**

To compare the models performance, a dataframe is created. The columns of this dataframe are the lists created to store the results of the model.

In [36]:

*#creating dataframe*

results **=** pd**.**DataFrame({ 'ML Model': ML\_Model,

'Train Accuracy': acc\_train,

'Test Accuracy': acc\_test})

results

Out[36]:

|  | **ML Model** | **Train Accuracy** | **Test Accuracy** |
| --- | --- | --- | --- |
| **0** | Decision Tree | 0.898 | 0.894 |
| **1** | Random Forest | 0.893 | 0.886 |
| **2** | SVM | 0.886 | 0.883 |
| **3** | XGBoost | 0.913 | 0.905 |

In [37]:

*#Sorting the datafram on accuracy*

results**.**sort\_values(by**=**['Test Accuracy', 'Train Accuracy'], ascending**=False**)

Out[37]:

|  | **ML Model** | **Train Accuracy** | **Test Accuracy** |
| --- | --- | --- | --- |
| **3** | XGBoost | 0.913 | 0.905 |
| **0** | Decision Tree | 0.898 | 0.894 |
| **1** | Random Forest | 0.893 | 0.886 |
| **2** | SVM | 0.886 | 0.883 |

For the above comparision, it is clear that the XGBoost Classifier works well with this dataset.

So, saving the model for future use.

In [38]:

*# save XGBoost model to file*

**import** pickle

pickle**.**dump(xgb, open("XGBoostClassifier.pkl", "wb"))

**Testing the saved model:**

In [39]:

*# load model from file*

loaded\_model **=** pickle**.**load(open("XGBoostClassifier.pkl", "rb"))

loaded\_model

Out[39]:

XGBClassifier(base\_score=0.5, booster='gbtree', colsample\_bylevel=1,

colsample\_bynode=1, colsample\_bytree=1, enable\_categorical=False,

gamma=0, gpu\_id=-1, importance\_type=None,

interaction\_constraints='', learning\_rate=0.4, max\_delta\_step=0,

max\_depth=7, min\_child\_weight=1, missing=nan,

monotone\_constraints='()', n\_estimators=100, n\_jobs=56,

num\_parallel\_tree=1, predictor='auto', random\_state=0,

reg\_alpha=0, reg\_lambda=1, scale\_pos\_weight=1, subsample=1,

tree\_method='exact', validate\_parameters=1, verbosity=0)

**9. References**

* <https://machinelearningmastery.com/save-gradient-boosting-models-xgboost-python/>

**10. Deployment**

In [40]:

pwd

Out[40]:

'/home/wsuser/work'

In [41]:

**!**pip install -U ibm-watson-machine-learning

Requirement already satisfied: ibm-watson-machine-learning in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (1.0.257)

Requirement already satisfied: lomond in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm-watson-machine-learning) (0.3.3)

Requirement already satisfied: packaging in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm-watson-machine-learning) (21.3)

Requirement already satisfied: importlib-metadata in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm-watson-machine-learning) (4.8.2)

Requirement already satisfied: certifi in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm-watson-machine-learning) (2022.9.24)

Requirement already satisfied: requests in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm-watson-machine-learning) (2.26.0)

Requirement already satisfied: urllib3 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm-watson-machine-learning) (1.26.7)

Requirement already satisfied: ibm-cos-sdk==2.11.\* in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm-watson-machine-learning) (2.11.0)

Requirement already satisfied: pandas<1.5.0,>=0.24.2 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm-watson-machine-learning) (1.3.4)

Requirement already satisfied: tabulate in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm-watson-machine-learning) (0.8.9)

Requirement already satisfied: ibm-cos-sdk-s3transfer==2.11.0 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm-cos-sdk==2.11.\*->ibm-watson-machine-learning) (2.11.0)

Requirement already satisfied: ibm-cos-sdk-core==2.11.0 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm-cos-sdk==2.11.\*->ibm-watson-machine-learning) (2.11.0)

Requirement already satisfied: jmespath<1.0.0,>=0.7.1 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm-cos-sdk==2.11.\*->ibm-watson-machine-learning) (0.10.0)

Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm-cos-sdk-core==2.11.0->ibm-cos-sdk==2.11.\*->ibm-watson-machine-learning) (2.8.2)

Requirement already satisfied: pytz>=2017.3 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from pandas<1.5.0,>=0.24.2->ibm-watson-machine-learning) (2021.3)

Requirement already satisfied: numpy>=1.17.3 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from pandas<1.5.0,>=0.24.2->ibm-watson-machine-learning) (1.20.3)

Requirement already satisfied: six>=1.5 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from python-dateutil<3.0.0,>=2.1->ibm-cos-sdk-core==2.11.0->ibm-cos-sdk==2.11.\*->ibm-watson-machine-learning) (1.15.0)

Requirement already satisfied: charset-normalizer~=2.0.0 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from requests->ibm-watson-machine-learning) (2.0.4)

Requirement already satisfied: idna<4,>=2.5 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from requests->ibm-watson-machine-learning) (3.3)

Requirement already satisfied: zipp>=0.5 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from importlib-metadata->ibm-watson-machine-learning) (3.6.0)

Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from packaging->ibm-watson-machine-learning) (3.0.4)

In [43]:

**from** ibm\_watson\_machine\_learning **import** APIClient

**import** json

**import** numpy **as** np

**Authenticate and set space**

In [46]:

wml\_credentials **=** {

"apikey": "",

"url": "https://us-south.ml.cloud.ibm.com"

}

*#hiding apikey because of security reasons. Use your own apikey.*

In [47]:

wml\_client**=**APIClient(wml\_credentials)

wml\_client**.**spaces**.**list()

Note: 'limit' is not provided. Only first 50 records will be displayed if the number of records exceed 50

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

ID NAME CREATED

4845b8ab-cb14-4346-b586-0b27febfb500 web-phishing-detection 2022-11-11T06:53:21.297Z

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

In [48]:

SPACE\_ID **=** "4845b8ab-cb14-4346-b586-0b27febfb500"

In [49]:

wml\_client**.**set**.**default\_space(SPACE\_ID)

Out[49]:

'SUCCESS'

In [50]:

wml\_client**.**software\_specifications**.**list(500)

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

NAME ASSET\_ID TYPE

default\_py3.6 0062b8c9-8b7d-44a0-a9b9-46c416adcbd9 base

kernel-spark3.2-scala2.12 020d69ce-7ac1-5e68-ac1a-31189867356a base

pytorch-onnx\_1.3-py3.7-edt 069ea134-3346-5748-b513-49120e15d288 base

scikit-learn\_0.20-py3.6 09c5a1d0-9c1e-4473-a344-eb7b665ff687 base

spark-mllib\_3.0-scala\_2.12 09f4cff0-90a7-5899-b9ed-1ef348aebdee base

pytorch-onnx\_rt22.1-py3.9 0b848dd4-e681-5599-be41-b5f6fccc6471 base

ai-function\_0.1-py3.6 0cdb0f1e-5376-4f4d-92dd-da3b69aa9bda base

shiny-r3.6 0e6e79df-875e-4f24-8ae9-62dcc2148306 base

tensorflow\_2.4-py3.7-horovod 1092590a-307d-563d-9b62-4eb7d64b3f22 base

pytorch\_1.1-py3.6 10ac12d6-6b30-4ccd-8392-3e922c096a92 base

tensorflow\_1.15-py3.6-ddl 111e41b3-de2d-5422-a4d6-bf776828c4b7 base

autoai-kb\_rt22.2-py3.10 125b6d9a-5b1f-5e8d-972a-b251688ccf40 base

runtime-22.1-py3.9 12b83a17-24d8-5082-900f-0ab31fbfd3cb base

scikit-learn\_0.22-py3.6 154010fa-5b3b-4ac1-82af-4d5ee5abbc85 base

default\_r3.6 1b70aec3-ab34-4b87-8aa0-a4a3c8296a36 base

pytorch-onnx\_1.3-py3.6 1bc6029a-cc97-56da-b8e0-39c3880dbbe7 base

kernel-spark3.3-r3.6 1c9e5454-f216-59dd-a20e-474a5cdf5988 base

pytorch-onnx\_rt22.1-py3.9-edt 1d362186-7ad5-5b59-8b6c-9d0880bde37f base

tensorflow\_2.1-py3.6 1eb25b84-d6ed-5dde-b6a5-3fbdf1665666 base

spark-mllib\_3.2 20047f72-0a98-58c7-9ff5-a77b012eb8f5 base

tensorflow\_2.4-py3.8-horovod 217c16f6-178f-56bf-824a-b19f20564c49 base

runtime-22.1-py3.9-cuda 26215f05-08c3-5a41-a1b0-da66306ce658 base

do\_py3.8 295addb5-9ef9-547e-9bf4-92ae3563e720 base

autoai-ts\_3.8-py3.8 2aa0c932-798f-5ae9-abd6-15e0c2402fb5 base

tensorflow\_1.15-py3.6 2b73a275-7cbf-420b-a912-eae7f436e0bc base

kernel-spark3.3-py3.9 2b7961e2-e3b1-5a8c-a491-482c8368839a base

pytorch\_1.2-py3.6 2c8ef57d-2687-4b7d-acce-01f94976dac1 base

spark-mllib\_2.3 2e51f700-bca0-4b0d-88dc-5c6791338875 base

pytorch-onnx\_1.1-py3.6-edt 32983cea-3f32-4400-8965-dde874a8d67e base

spark-mllib\_3.0-py37 36507ebe-8770-55ba-ab2a-eafe787600e9 base

spark-mllib\_2.4 390d21f8-e58b-4fac-9c55-d7ceda621326 base

autoai-ts\_rt22.2-py3.10 396b2e83-0953-5b86-9a55-7ce1628a406f base

xgboost\_0.82-py3.6 39e31acd-5f30-41dc-ae44-60233c80306e base

pytorch-onnx\_1.2-py3.6-edt 40589d0e-7019-4e28-8daa-fb03b6f4fe12 base

pytorch-onnx\_rt22.2-py3.10 40e73f55-783a-5535-b3fa-0c8b94291431 base

default\_r36py38 41c247d3-45f8-5a71-b065-8580229facf0 base

autoai-ts\_rt22.1-py3.9 4269d26e-07ba-5d40-8f66-2d495b0c71f7 base

autoai-obm\_3.0 42b92e18-d9ab-567f-988a-4240ba1ed5f7 base

pmml-3.0\_4.3 493bcb95-16f1-5bc5-bee8-81b8af80e9c7 base

spark-mllib\_2.4-r\_3.6 49403dff-92e9-4c87-a3d7-a42d0021c095 base

xgboost\_0.90-py3.6 4ff8d6c2-1343-4c18-85e1-689c965304d3 base

pytorch-onnx\_1.1-py3.6 50f95b2a-bc16-43bb-bc94-b0bed208c60b base

autoai-ts\_3.9-py3.8 52c57136-80fa-572e-8728-a5e7cbb42cde base

spark-mllib\_2.4-scala\_2.11 55a70f99-7320-4be5-9fb9-9edb5a443af5 base

spark-mllib\_3.0 5c1b0ca2-4977-5c2e-9439-ffd44ea8ffe9 base

autoai-obm\_2.0 5c2e37fa-80b8-5e77-840f-d912469614ee base

spss-modeler\_18.1 5c3cad7e-507f-4b2a-a9a3-ab53a21dee8b base

cuda-py3.8 5d3232bf-c86b-5df4-a2cd-7bb870a1cd4e base

autoai-kb\_3.1-py3.7 632d4b22-10aa-5180-88f0-f52dfb6444d7 base

pytorch-onnx\_1.7-py3.8 634d3cdc-b562-5bf9-a2d4-ea90a478456b base

spark-mllib\_2.3-r\_3.6 6586b9e3-ccd6-4f92-900f-0f8cb2bd6f0c base

tensorflow\_2.4-py3.7 65e171d7-72d1-55d9-8ebb-f813d620c9bb base

spss-modeler\_18.2 687eddc9-028a-4117-b9dd-e57b36f1efa5 base

pytorch-onnx\_1.2-py3.6 692a6a4d-2c4d-45ff-a1ed-b167ee55469a base

spark-mllib\_2.3-scala\_2.11 7963efe5-bbec-417e-92cf-0574e21b4e8d base

spark-mllib\_2.4-py37 7abc992b-b685-532b-a122-a396a3cdbaab base

caffe\_1.0-py3.6 7bb3dbe2-da6e-4145-918d-b6d84aa93b6b base

pytorch-onnx\_1.7-py3.7 812c6631-42b7-5613-982b-02098e6c909c base

cuda-py3.6 82c79ece-4d12-40e6-8787-a7b9e0f62770 base

tensorflow\_1.15-py3.6-horovod 8964680e-d5e4-5bb8-919b-8342c6c0dfd8 base

hybrid\_0.1 8c1a58c6-62b5-4dc4-987a-df751c2756b6 base

pytorch-onnx\_1.3-py3.7 8d5d8a87-a912-54cf-81ec-3914adaa988d base

caffe-ibm\_1.0-py3.6 8d863266-7927-4d1e-97d7-56a7f4c0a19b base

spss-modeler\_17.1 902d0051-84bd-4af6-ab6b-8f6aa6fdeabb base

do\_12.10 9100fd72-8159-4eb9-8a0b-a87e12eefa36 base

do\_py3.7 9447fa8b-2051-4d24-9eef-5acb0e3c59f8 base

spark-mllib\_3.0-r\_3.6 94bb6052-c837-589d-83f1-f4142f219e32 base

cuda-py3.7-opence 94e9652b-7f2d-59d5-ba5a-23a414ea488f base

nlp-py3.8 96e60351-99d4-5a1c-9cc0-473ac1b5a864 base

cuda-py3.7 9a44990c-1aa1-4c7d-baf8-c4099011741c base

hybrid\_0.2 9b3f9040-9cee-4ead-8d7a-780600f542f7 base

spark-mllib\_3.0-py38 9f7a8fc1-4d3c-5e65-ab90-41fa8de2d418 base

autoai-kb\_3.3-py3.7 a545cca3-02df-5c61-9e88-998b09dc79af base

spark-mllib\_3.0-py39 a6082a27-5acc-5163-b02c-6b96916eb5e0 base

runtime-22.1-py3.9-do a7e7dbf1-1d03-5544-994d-e5ec845ce99a base

default\_py3.8 ab9e1b80-f2ce-592c-a7d2-4f2344f77194 base

tensorflow\_rt22.1-py3.9 acd9c798-6974-5d2f-a657-ce06e986df4d base

kernel-spark3.2-py3.9 ad7033ee-794e-58cf-812e-a95f4b64b207 base

autoai-obm\_2.0 with Spark 3.0 af10f35f-69fa-5d66-9bf5-acb58434263a base

default\_py3.7\_opence c2057dd4-f42c-5f77-a02f-72bdbd3282c9 base

tensorflow\_2.1-py3.7 c4032338-2a40-500a-beef-b01ab2667e27 base

do\_py3.7\_opence cc8f8976-b74a-551a-bb66-6377f8d865b4 base

spark-mllib\_3.3 d11f2434-4fc7-58b7-8a62-755da64fdaf8 base

autoai-kb\_3.0-py3.6 d139f196-e04b-5d8b-9140-9a10ca1fa91a base

spark-mllib\_3.0-py36 d82546d5-dd78-5fbb-9131-2ec309bc56ed base

autoai-kb\_3.4-py3.8 da9b39c3-758c-5a4f-9cfd-457dd4d8c395 base

kernel-spark3.2-r3.6 db2fe4d6-d641-5d05-9972-73c654c60e0a base

autoai-kb\_rt22.1-py3.9 db6afe93-665f-5910-b117-d879897404d9 base

tensorflow\_rt22.1-py3.9-horovod dda170cc-ca67-5da7-9b7a-cf84c6987fae base

autoai-ts\_1.0-py3.7 deef04f0-0c42-5147-9711-89f9904299db base

tensorflow\_2.1-py3.7-horovod e384fce5-fdd1-53f8-bc71-11326c9c635f base

default\_py3.7 e4429883-c883-42b6-87a8-f419d64088cd base

do\_22.1 e51999ba-6452-5f1f-8287-17228b88b652 base

autoai-obm\_3.2 eae86aab-da30-5229-a6a6-1d0d4e368983 base

tensorflow\_rt22.2-py3.10 f65bd165-f057-55de-b5cb-f97cf2c0f393 base

do\_20.1 f686cdd9-7904-5f9d-a732-01b0d6b10dc5 base

pytorch-onnx\_rt22.2-py3.10-edt f8a05d07-e7cd-57bb-a10b-23f1d4b837ac base

scikit-learn\_0.19-py3.6 f963fa9d-4bb7-5652-9c5d-8d9289ef6ad9 base

tensorflow\_2.4-py3.8 fe185c44-9a99-5425-986b-59bd1d2eda46 base

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

**Save and deploy the model**

In [51]:

**import** sklearn

sklearn**.**\_\_version\_\_

Out[51]:

'1.0.2'

In [52]:

MODEL\_NAME **=** 'WebPhishingDetection'

DEPLOYMENT\_LAYER **=** 'WebPhishingDetectionDeployment'

DEMO\_MODEL **=** xgb

In [53]:

*#set python version*

software\_spec\_uid **=** wml\_client**.**software\_specifications**.**get\_id\_by\_name('runtime-22.1-py3.9')

In [54]:

*#setup model meta*

model\_props **=** {

wml\_client**.**repository**.**ModelMetaNames**.**NAME: MODEL\_NAME,

wml\_client**.**repository**.**ModelMetaNames**.**TYPE: 'scikit-learn\_1.0',

wml\_client**.**repository**.**ModelMetaNames**.**SOFTWARE\_SPEC\_UID: software\_spec\_uid

}

In [55]:

*#save model*

model\_details **=** wml\_client**.**repository**.**store\_model(

model**=**DEMO\_MODEL,

meta\_props**=**model\_props,

training\_data**=**X\_train,

training\_target**=**y\_train

)

In [56]:

model\_details

Out[56]:

{'entity': {'hybrid\_pipeline\_software\_specs': [],

'label\_column': 'l1',

'schemas': {'input': [{'fields': [{'name': 'f0', 'type': 'int'},

{'name': 'f1', 'type': 'int'},

{'name': 'f2', 'type': 'int'},

{'name': 'f3', 'type': 'int'},

{'name': 'f4', 'type': 'int'},

{'name': 'f5', 'type': 'int'},

{'name': 'f6', 'type': 'int'},

{'name': 'f7', 'type': 'int'},

{'name': 'f8', 'type': 'int'},

{'name': 'f9', 'type': 'int'},

{'name': 'f10', 'type': 'int'},

{'name': 'f11', 'type': 'int'},

{'name': 'f12', 'type': 'int'}],

'id': '1',

'type': 'struct'}],

'output': []},

'software\_spec': {'id': '12b83a17-24d8-5082-900f-0ab31fbfd3cb',

'name': 'runtime-22.1-py3.9'},

'type': 'scikit-learn\_1.0'},

'metadata': {'created\_at': '2022-11-11T07:09:42.002Z',

'id': '3645aafc-b5f9-412d-8e48-819c8d9bfdc4',

'modified\_at': '2022-11-11T07:09:44.807Z',

'name': 'WebPhishingDetection',

'owner': 'IBMid-667000EZJI',

'resource\_key': 'dde01532-da6b-419b-90e7-76b012669ee5',

'space\_id': '4845b8ab-cb14-4346-b586-0b27febfb500'},

'system': {'warnings': []}}

In [57]:

model\_id **=** wml\_client**.**repository**.**get\_model\_id(model\_details)

In [58]:

model\_id

Out[58]:

'3645aafc-b5f9-412d-8e48-819c8d9bfdc4'

In [60]:

*#set meta*

deployment\_props **=** {

wml\_client**.**deployments**.**ConfigurationMetaNames**.**NAME:DEPLOYMENT\_LAYER,

wml\_client**.**deployments**.**ConfigurationMetaNames**.**ONLINE: {}

}

In [62]:

*#deploy*

deployment **=** wml\_client**.**deployments**.**create(

artifact\_uid**=**model\_id,

meta\_props**=**deployment\_props

)

#######################################################################################

Synchronous deployment creation for uid: '3645aafc-b5f9-412d-8e48-819c8d9bfdc4' started

#######################################################################################

initializing

Note: online\_url is deprecated and will be removed in a future release. Use serving\_urls instead.

ready

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

Successfully finished deployment creation, deployment\_uid='859ae568-d692-4958-9dbe-60431a8a0918'