CS 584 TEAM TERM PROJECT

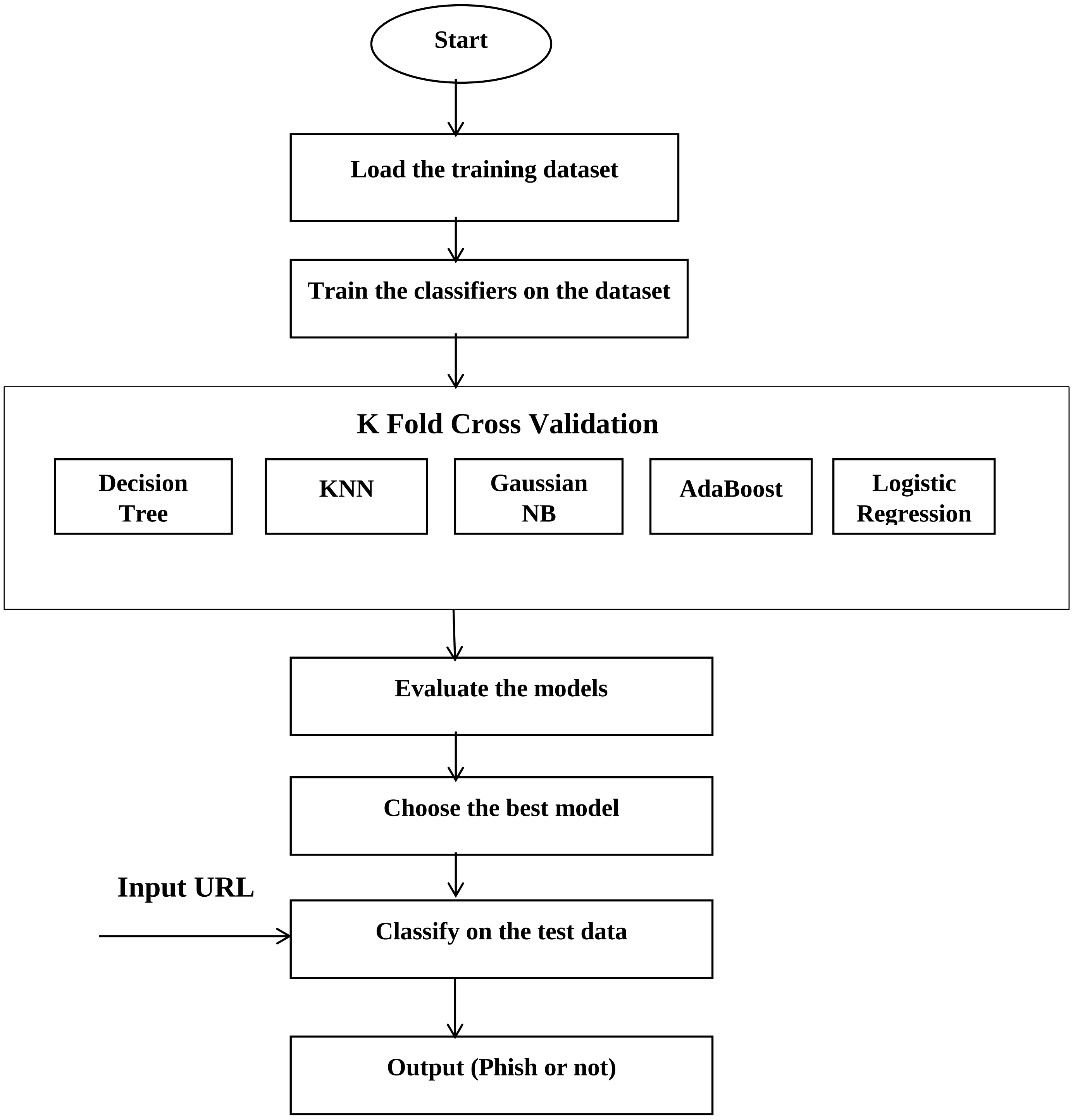
PHISHING URL DETECTION

By

SRIHASA VEJENDLA

MITHILAESH JAYAKUMAR

1. MODEL DESIGN AND ARCHITCTURE



2. DATA CAPTURE

The dataset set used in this project was obtained from two sources.

Source:

i) **Legitimate URL :** We can get a list of legitimate urls from the following link http://www.legitimate.com/?f

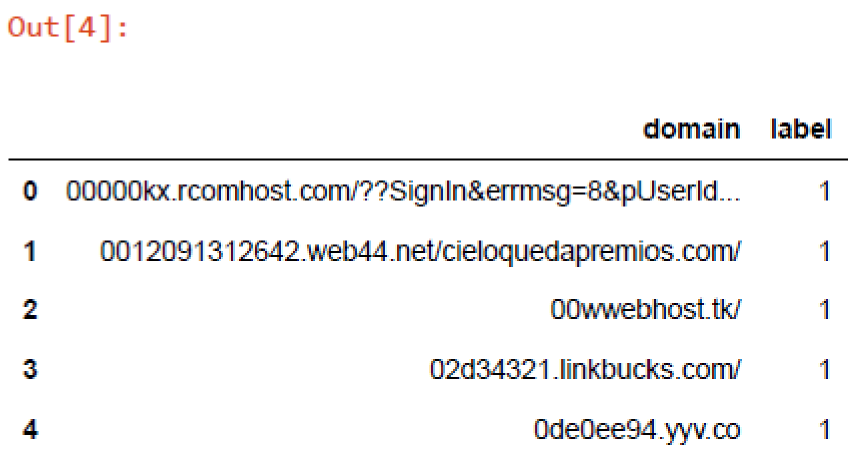
ii) **Phishing URL :** We can get a list of Phishing urls from the following link https://www.phishtank.com/

Features:

We obtained both the list of Legitimate and phishing URLs from the above mentioned sources. Both these lists were appended to a .csv file. This file contains two columns one is the url itself and another is the class label indicating 0 if it's legitimate or 1 if its phishing url. The class labels were manually assigned while combining the two categories of the URLs into the .csv file.

Size of the dataset:

The dataset contains "7498" - 0 label features and "6251" - 1 label features.



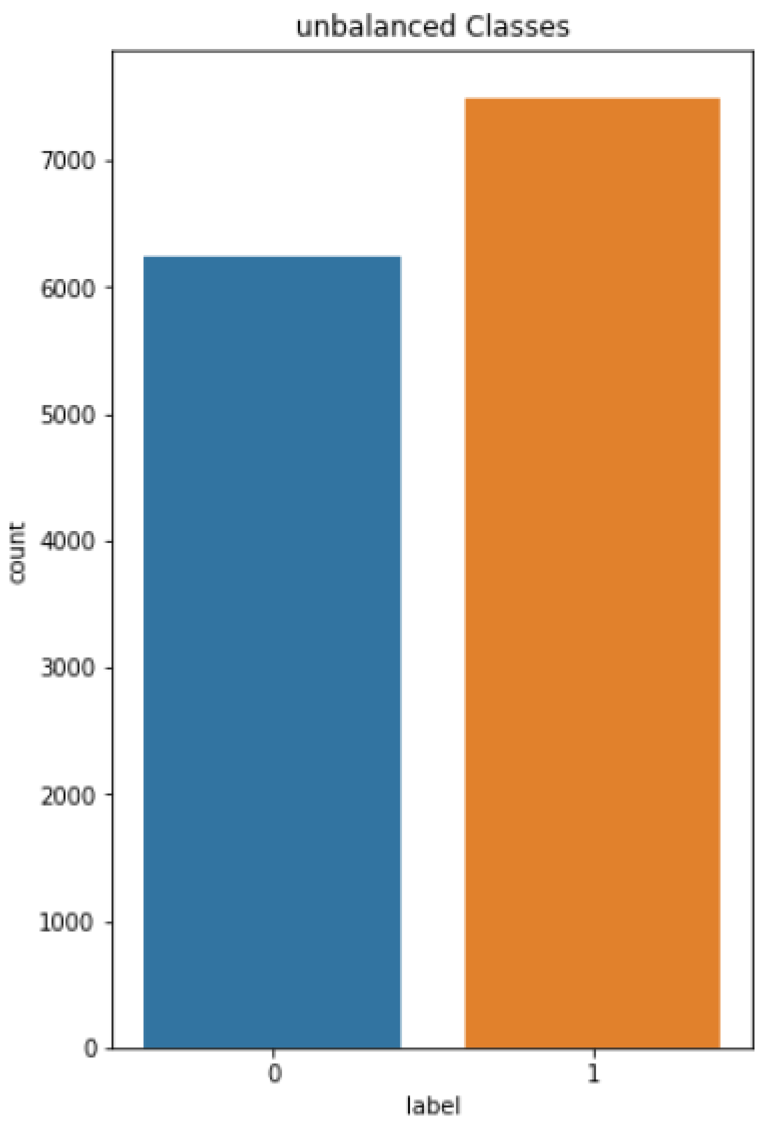
```
Out[4]:

                                          domain  label
0  00000kx.rcomhost.com/??SignIn&errmsg=8&pUserId...      1
1    0012091312642.web44.net/cieloquedapremios.com/      1
2                                    00wwebhost.tk/      1
3                             02d34321.linkbucks.com/      1
4                                    0de0ee94.yyv.co      1
```

3. DATA PREPROCESSING

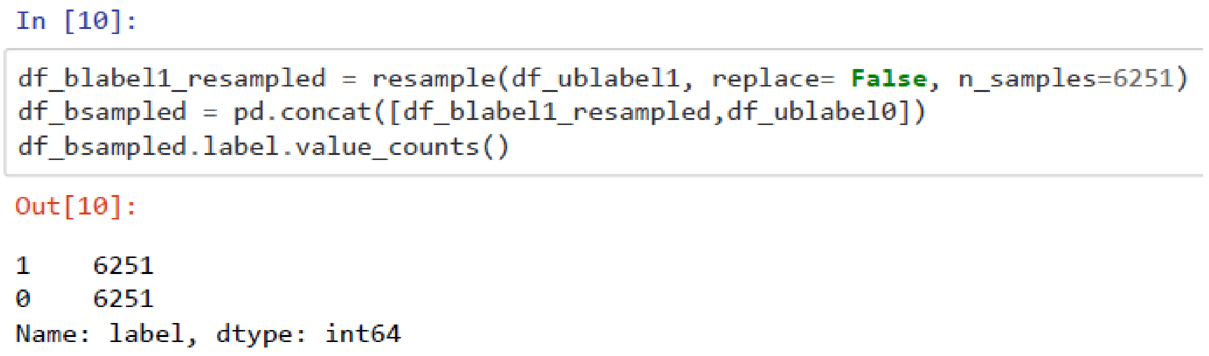
As the dataset primarily consists of a list of URLs and the labels were manually assigned to the URLs based on the source from which they were taken, the dataset was complete and no pre-processing was required on this data.

4. BALANCED VS UNBALANCED DISTRIBUTION

Our dataset contains "7498" - 0 label features and "6251" - 1 label features which is an unbalanced distribution.



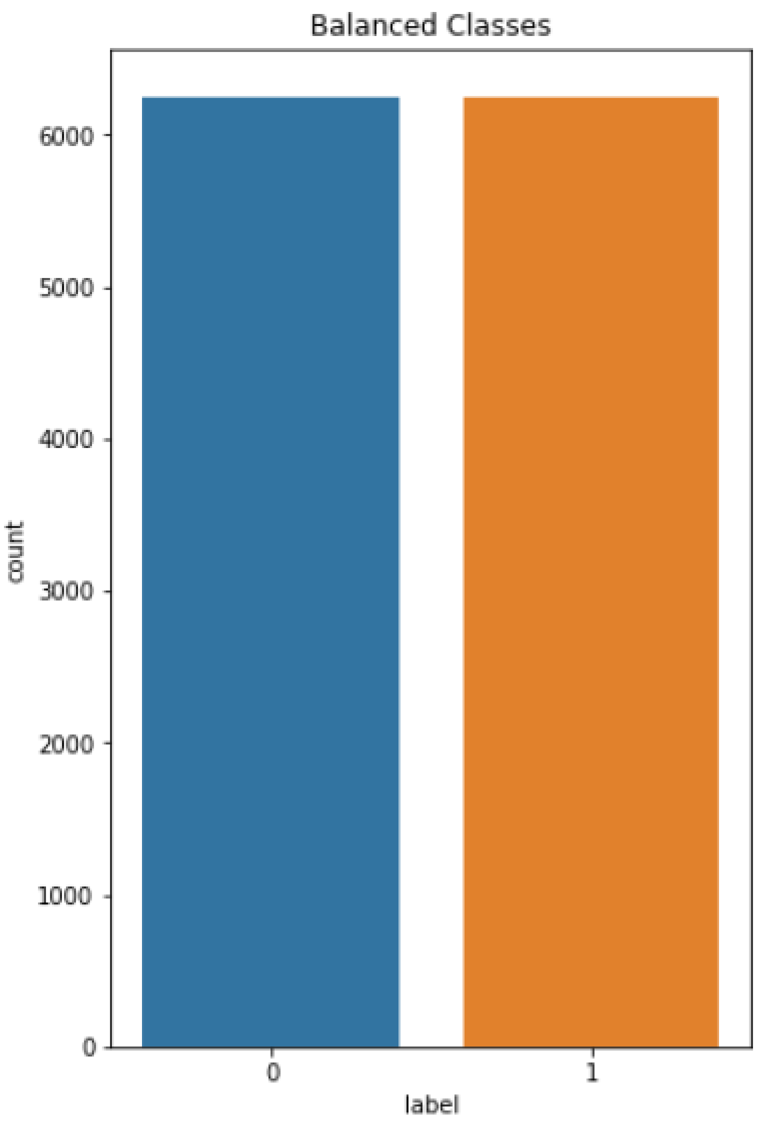
So we balanced the dataset distribution using from sklearn.utils import resample. After resampling the dataset contains “6251” - 0 label features and “6251” - 1 label features.



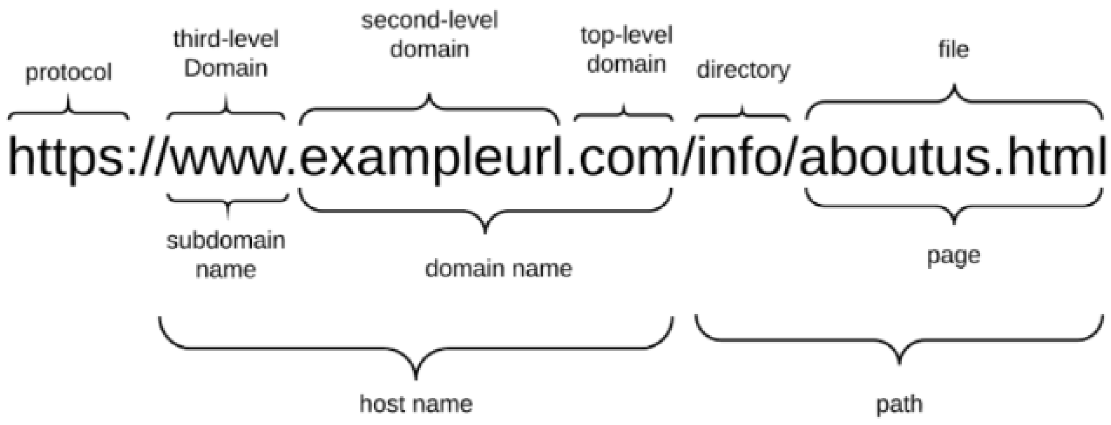
```
In [10]:
```

```
df_blabel1_resampled = resample(df_ublabel1, replace= False, n_samples=6251)
df_bsampled = pd.concat([df_blabel1_resampled,df_ublabel0])
df_bsampled.label.value_counts()
```

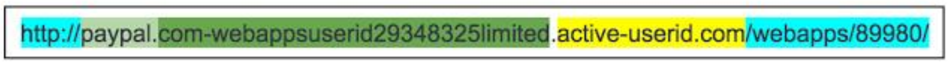
```
Out[10]:

1    6251
0    6251
Name: label, dtype: int64
```



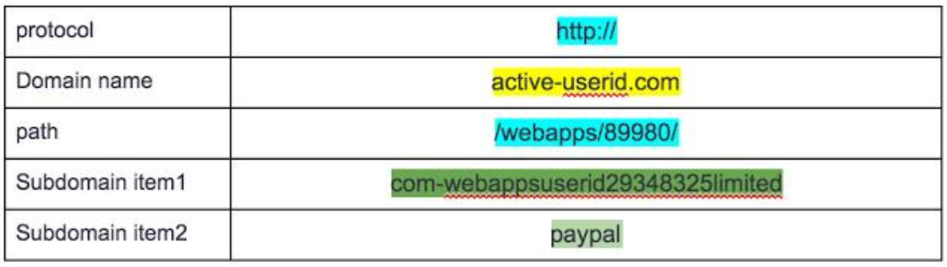
5. FEATURE CREATION



The above diagram represents the parts of a URL. The domain name portion is constrained since it has to be registered with a domain name Registrar. A Host name consists of a subdomain name and a domain name. A phisher has full control over the subdomain portions and can set any value to it. The URL may also have a path and file components which, too, can be changed by the phisher at will. The subdomain name and path are fully controllable by the phisher.



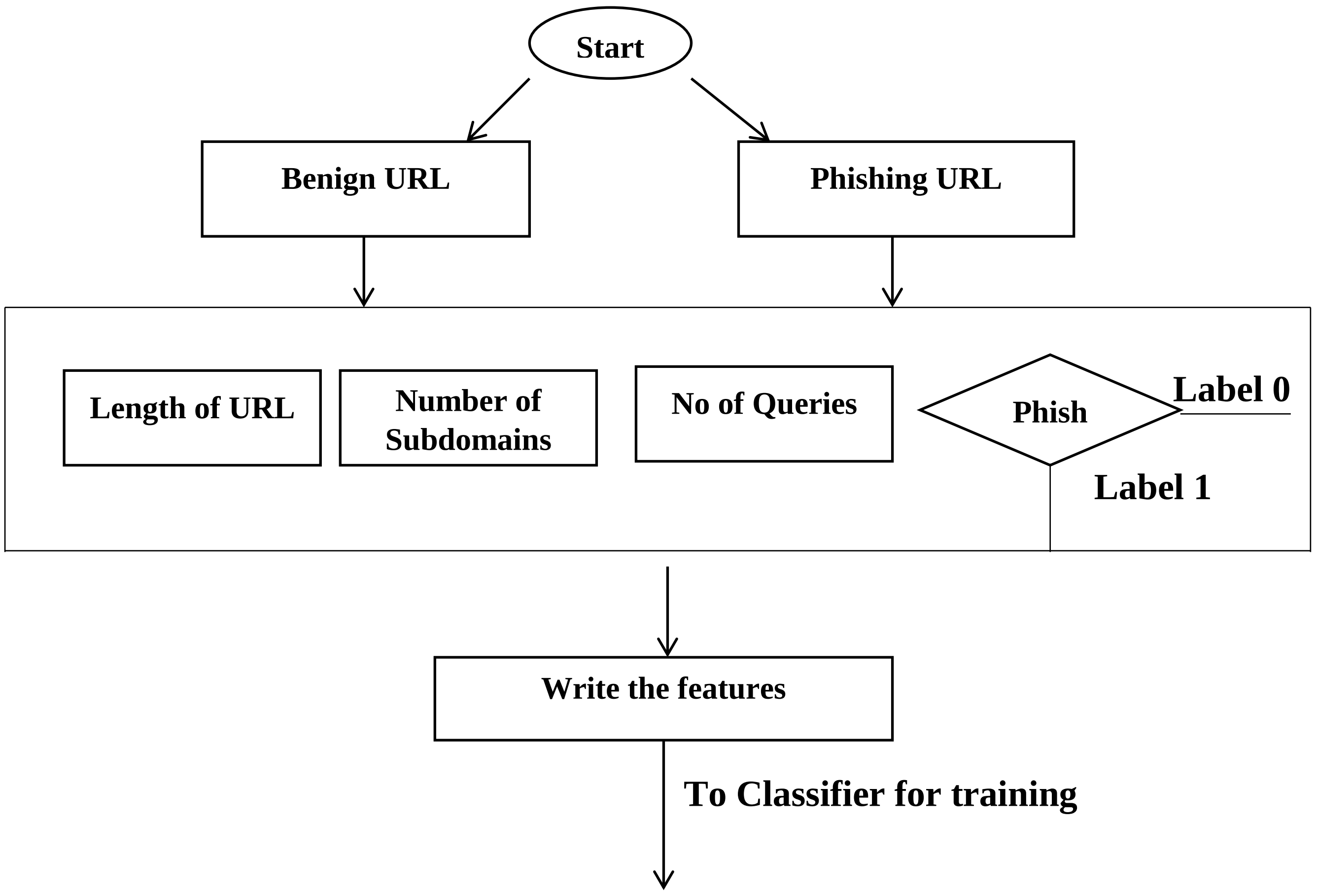
http://paypal.com-webappsuserid29348325limited.active-userid.com/webapps/89980/



protocol	http://
Domain name	active-userid.com
path	/webapps/89980/
Subdomain item1	com-webappsuserid29348325limited
Subdomain item2	paypal

In the above example, although the real domain name is active-userid.com, the attacker tried to make the domain look like paypal.com by adding subdomain as PayPal.

As the URL itself exhibits some distinctive features, we write functions to calculate the no. of dots, presence of hyphen, length of url, presence of at, presence of double slash, no of subdir, no of subdomain, label etc. from a url which can be fed to the classifier for training.



```
# Method to count number of dots
def countdots(url):
    return url.count('.')
```

In [13]:

```
# Method to count number of delimeters
def countdelim(url):
    count = 0
    delim=[';','_','?','=','&']
    for each in url:
        if each in delim:
            count = count + 1

    return count
```

```
# Is IP addr present as the hostname, let's validate

import ipaddress as ip #works only in python 3

def isip(url):
    try:
        if ip.ip_address(url):
            return 1
    except:
        return 0
```

In [15]:

```
#method to check the presence of hyphens

def isPresentHyphen(url):
    return url.count('-')
```

In [16]:

```
#method to check the presence of @

def isPresentAt(url):
    return url.count('@')
```

In [17]:

```
def isPresentDSlash(url):
    return url.count('//')
```

```
def countSubDir(url):
    return url.count('/')
```

In [19]:

```
def get_ext(url):

    root, ext = splitext(url)
    return ext
```

In [20]:

```
def countSubDomain(subdomain):
    if not subdomain:
        return 0
    else:
        return len(subdomain.split('.'))
```

```
def countQueries(query):
    if not query:
        return 0
    else:
        return len(query.split('&'))
```

In [26]:

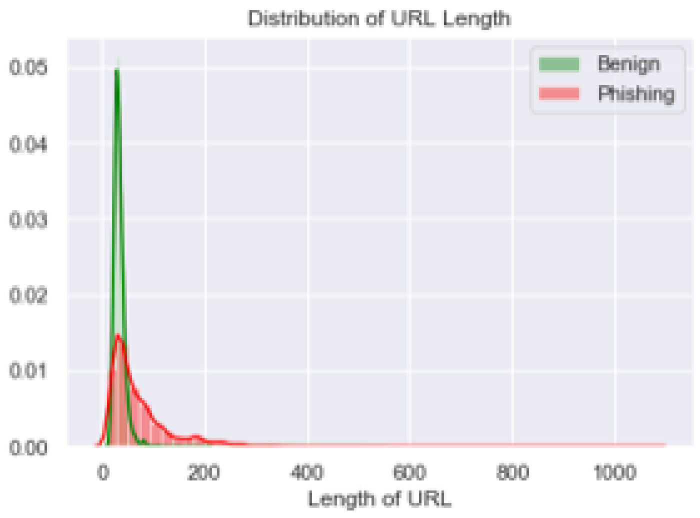
```
featureSet.head()
```

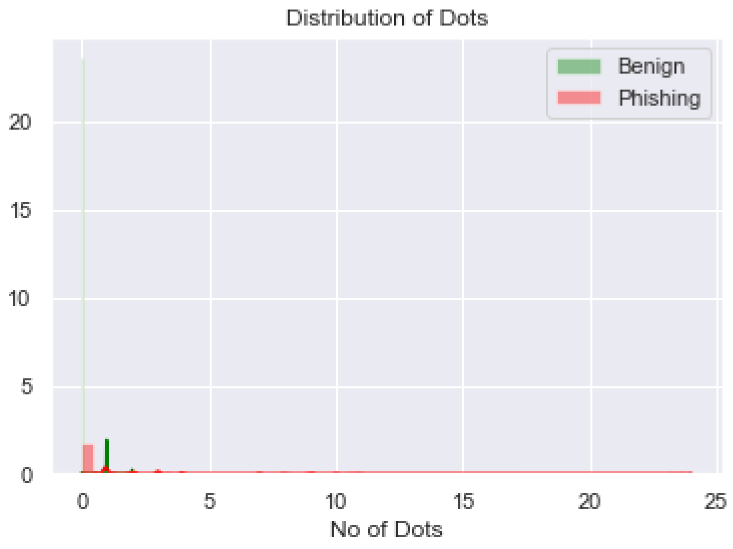
Out[26]:

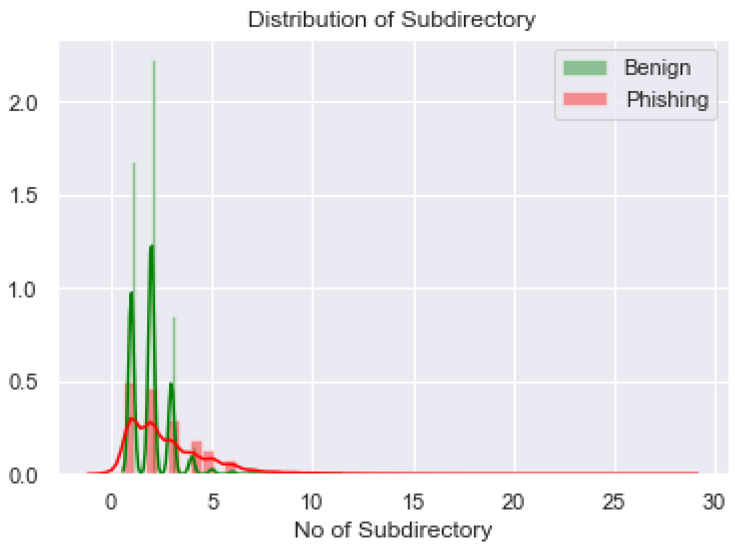
	url	no of dots	presence of hyphen	len of url	presence of at	presence of double slash
0	00000kx.rcomhost.com/??SignIn&errmsg=8&pUserId...	0	0	164	0	0
1	0012091312642.web44.net/cieloquedapremios.com/	0	0	46	0	0
2	00wwebhost.tk/	0	0	14	0	0
3	02d34321.linkbucks.com/	0	0	23	0	0
4	0de0ee94.yyv.co	0	0	15	0	0

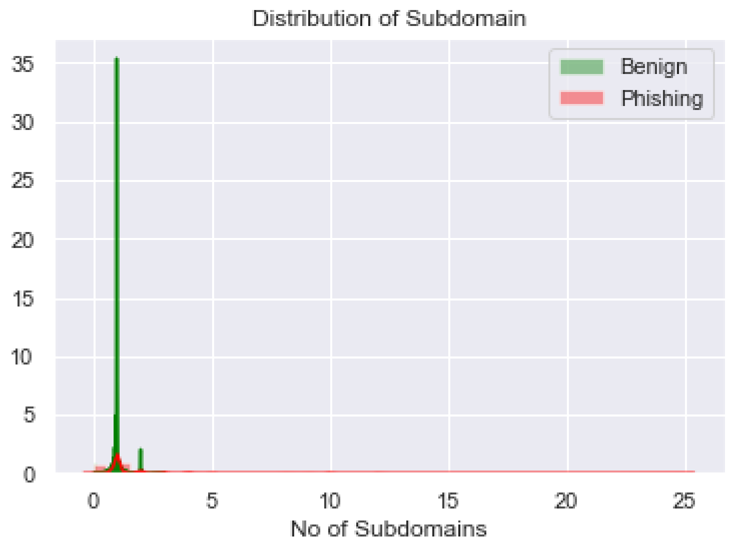
6. VISUALIZATION

We visually represent the features created from the url to have an insight on the variation between Phishing and legitimate URLs with respect to these features.



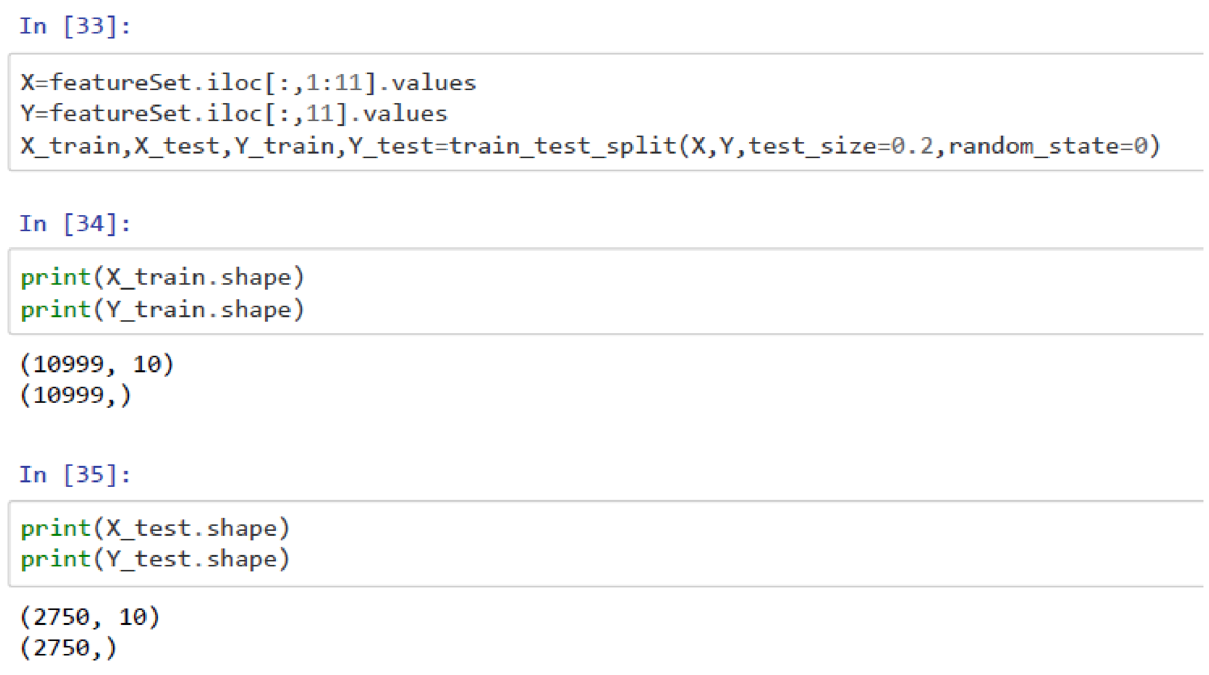






7. TEST / TRAIN SPLIT

The features created acts a dataset which is split into train and test datasets in an 80:20 ratio. The train dataset is used to train different classification models. The test data is used to finally evaluate the best model obtained. Here we do not use any data normalization methods such as MinMaxScaler or StandardScaler as the features such as len of url, no of subdomains, no of queries etc. plays a distinctive role in highlighting between legitimate and phishing urls and normalizing the values impacts the classifiers accuracy during prediction.



```
In [33]:

X=featureSet.iloc[:,1:11].values
Y=featureSet.iloc[:,11].values
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=0)

In [34]:

print(X_train.shape)
print(Y_train.shape)

(10999, 10)
(10999,)

In [35]:

print(X_test.shape)
print(Y_test.shape)

(2750, 10)
(2750,)
```

8. CLASSIFICATION MODELS

Here we use the below models for classifying a url into legitimate or phishing based on the features.

1. Decision Tree
2. AdaBoost
3. Logistic Regression
4. Gaussian NB
5. KNN

9. FREEZE AND TEST

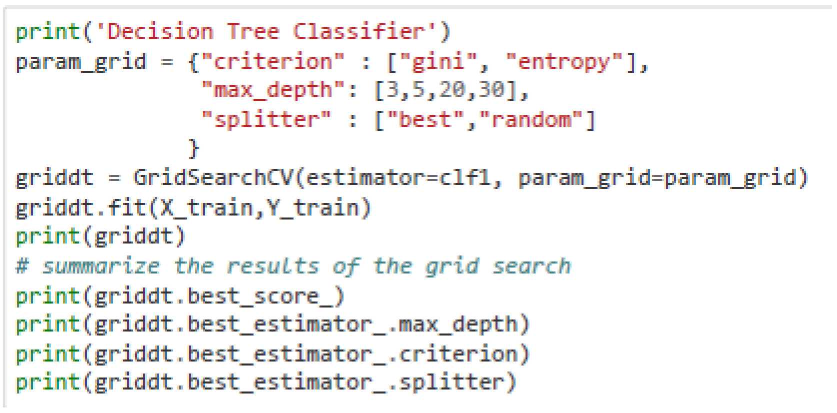
Models can have many hyper parameters and finding the best combination of parameters can be treated as a search problem. Initially all the models were trained on the Feature set obtained from the url without tuning any hyper parameters and were tested for accuracy.

Classifiers	**Before Tuning**
Decision Tree	0.807636
AdaBoost	0.799636
Logistic Regression	0.765455
Gaussian NB	0.658545
KNN	0.786545

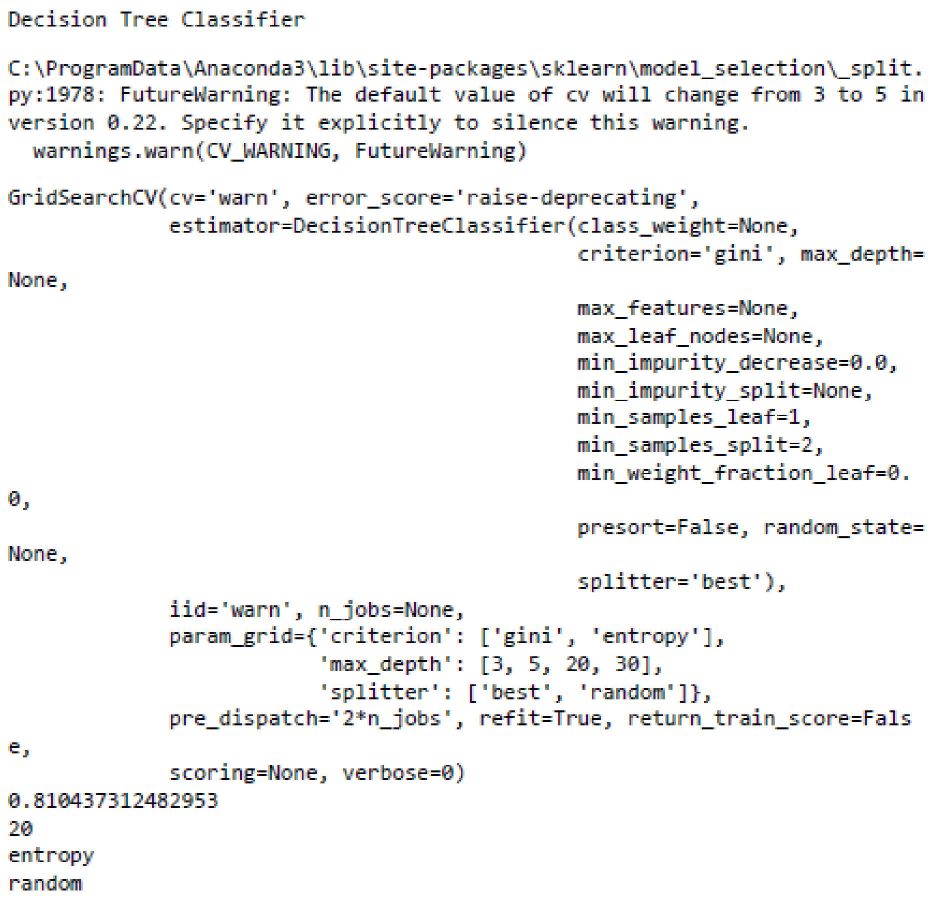
Then we tune the hyper parameters for each model. We use GridSearchCV to find the best hyper parameters that can increase the accuracy for the same dataset. The following hyper parameters were tested using GridSearchCV to get the one with the best values.

1. Decision Tree
    - Criterion, max_depth, splitter hyper parameters were tuned to find the best matching.
2. AdaBoost
    - learning_rate, n_estimators hyper parameters were tuned to find the best matching.
3. Logistic Regression
    - penalty, C hyper parameters were tuned to find the best matching.
4. KNN
    - n_neighbors, weights hyper parameters were tuned to find the best matching.

The below is a sample screenshot of results given by GridSearchCV for Decision Tree model.

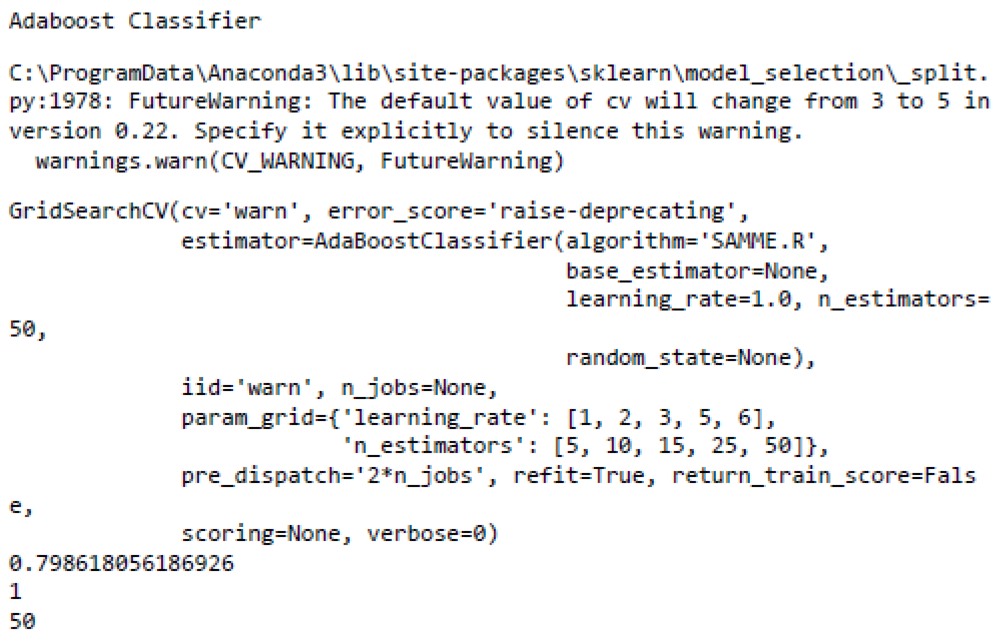


```
print('Decision Tree Classifier')
param_grid = {"criterion" : ["gini", "entropy"],
              "max_depth": [3,5,20,30],
              "splitter" : ["best","random"]
             }
griddt = GridSearchCV(estimator=clf1, param_grid=param_grid)
griddt.fit(X_train,Y_train)
print(griddt)
# summarize the results of the grid search
print(griddt.best_score_)
print(griddt.best_estimator_.max_depth)
print(griddt.best_estimator_.criterion)
print(griddt.best_estimator_.splitter)
```

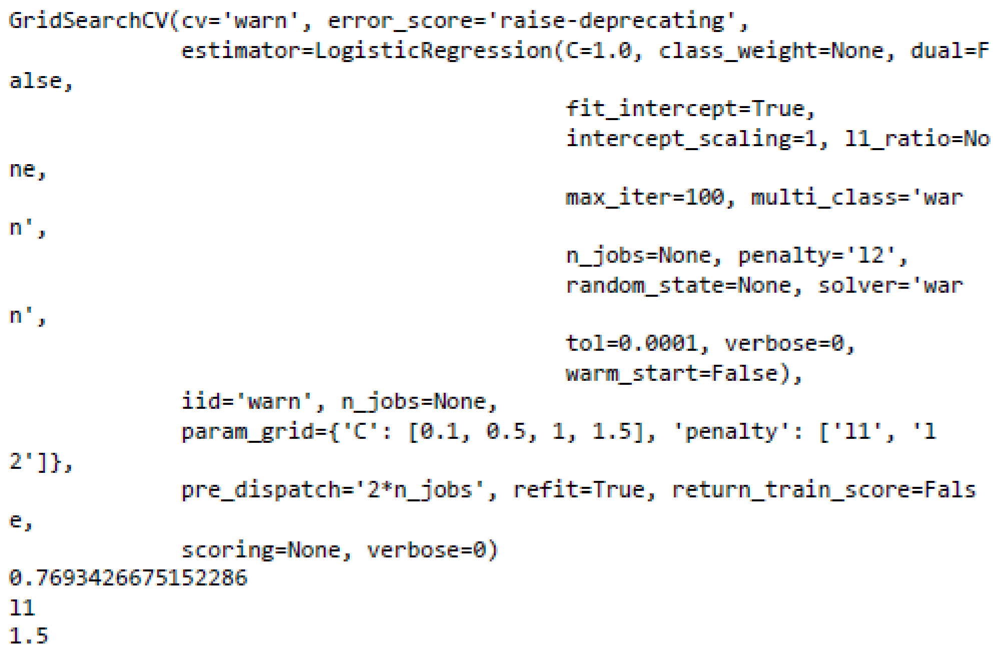


```
Decision Tree Classifier

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_split.
py:1978: FutureWarning: The default value of cv will change from 3 to 5 in
version 0.22. Specify it explicitly to silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)

GridSearchCV(cv='warn', error_score='raise-deprecating',
             estimator=DecisionTreeClassifier(class_weight=None,
                                              criterion='gini', max_depth=
None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.
0,
                                              presort=False, random_state=
None,
                                              splitter='best'),
             iid='warn', n_jobs=None,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [3, 5, 20, 30],
                         'splitter': ['best', 'random']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=Fals
e,
             scoring=None, verbose=0)
0.810437312482953
20
entropy
random
```

The results obtained for the other models used are as follows.



```
Adaboost Classifier

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_split.
py:1978: FutureWarning: The default value of cv will change from 3 to 5 in
version 0.22. Specify it explicitly to silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)

GridSearchCV(cv='warn', error_score='raise-deprecating',
             estimator=AdaBoostClassifier(algorithm='SAMME.R',
                                          base_estimator=None,
                                          learning_rate=1.0, n_estimators=
50,
                                          random_state=None),
             iid='warn', n_jobs=None,
             param_grid={'learning_rate': [1, 2, 3, 5, 6],
                         'n_estimators': [5, 10, 15, 25, 50]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=Fals
e,
             scoring=None, verbose=0)
0.798618056186926
1
50
```



```
GridSearchCV(cv='warn', error_score='raise-deprecating',
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=F
alse,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=No
ne,
                                          max_iter=100, multi_class='war
n',
                                          n_jobs=None, penalty='l2',
                                          random_state=None, solver='war
n',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'C': [0.1, 0.5, 1, 1.5], 'penalty': ['l1', 'l
2']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=Fals
e,
             scoring=None, verbose=0)
0.7693426675152286
l1
1.5
```

```
KNN Classifier
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=3
0,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=Non
e,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='warn', n_jobs=None,
             param_grid={'n_neighbors': [1, 2, 3, 4, 5],
                         'weights': ['uniform', 'distance']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=Fals
e,
             scoring=None, verbose=0)
0.793163014819529
4
distance
```

After training the models on the tuned hyper parameters the below three models showed an increase in the accuracy.

Classifiers	**After tuning**
Decision Tree	0.81252
Logistic Regression	0.769524
KNN	0.793163

10. K-FOLD CROSS-VALIDATION

Now we train the models on the tuned hyper parameters and also use 5 fold cross validation for evaluating each model. Using cross_val_score, the dataset is folded into 5 splits using cv attribute and we get the validation score for each fold. The package is from sklearn.model_selection import cross_val_score.

```
print('Decision Tree Classifier')
clf1 = tree.DecisionTreeClassifier(max_depth =20 , criterion = 'entropy', splitter = 'r
andom')
clf1.fit(X_train,Y_train)
scores_1 = cross_val_score(clf1, X_train, Y_train, cv=5)
print("Accuracy: %f (+/- %0.2f)" % (scores_1.mean(), scores_1.std() * 2))
```

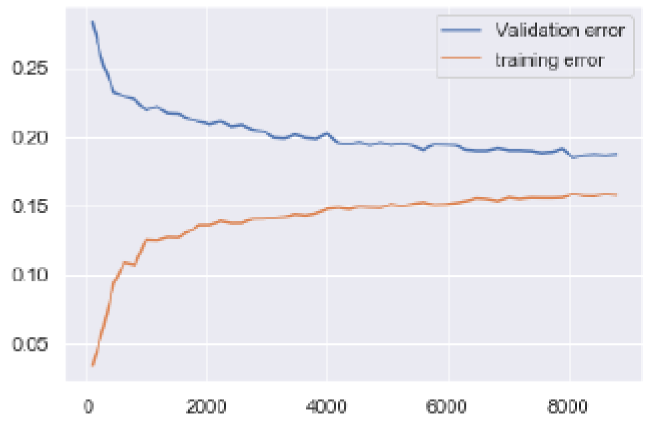
```
Decision Tree Classifier
Accuracy: 0.811891 (+/- 0.02)
```

11.LEARNING CURVES

We use learning curves to evaluate the performance of the different models during training and validation. We plot a curve between the training error and validation error which helps us to understand better about the bias and variance of the model.

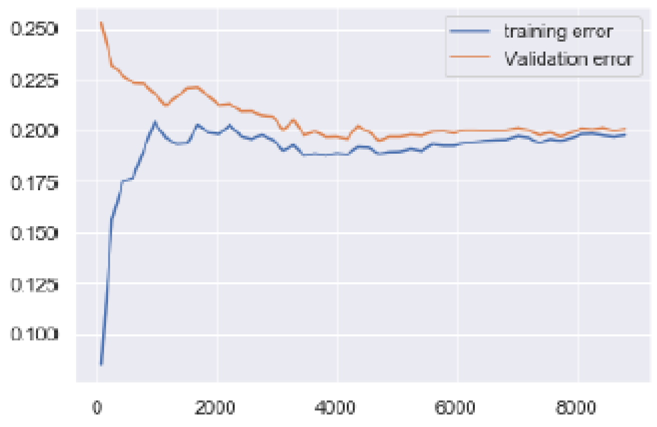
```
train_mean = np.mean(1-train_scores, axis=1)
validation_mean = np.mean(1-validation_scores, axis=1)
plt.plot(train_sizes, validation_mean, label = 'Validation error')
plt.plot(train_sizes, train_mean, label = 'training error')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x1984fb8b198>
```



Decision Tree Learning Curve

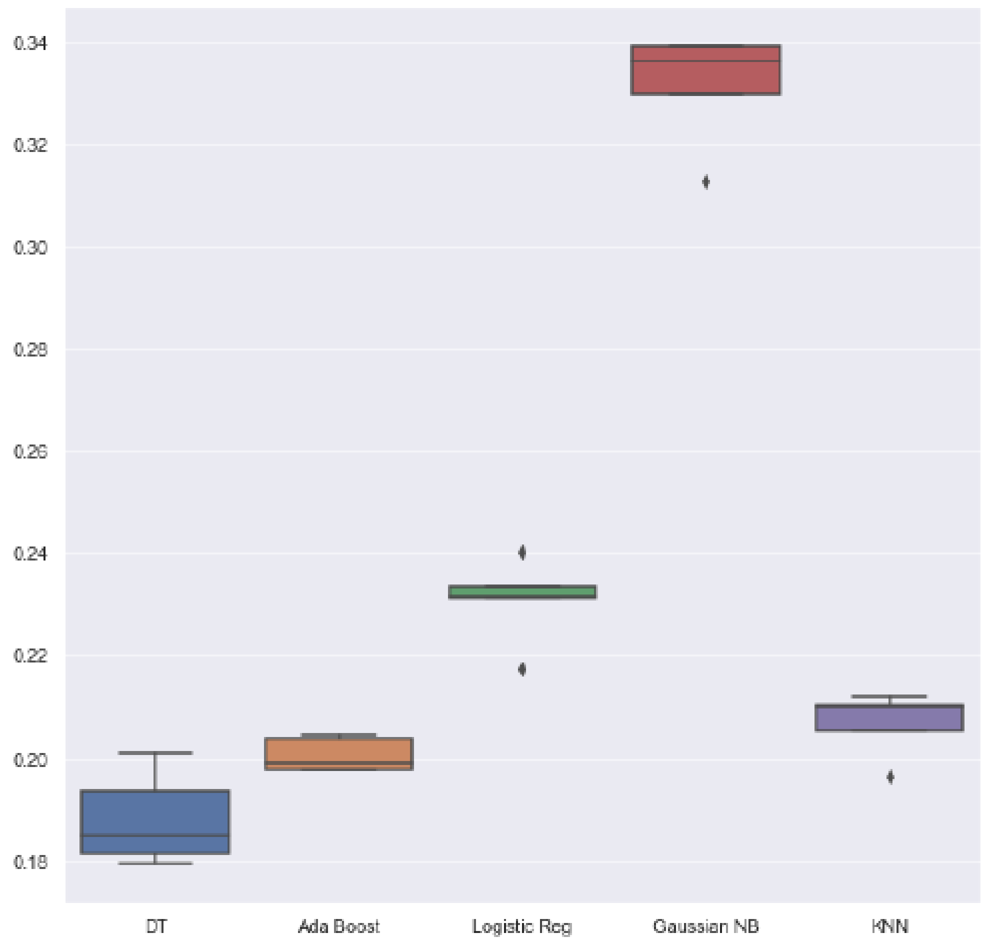
```
<matplotlib.legend.Legend at 0x19850160550>
```



AdaBoost Learning Curve

12.ERROR VS METHOD

The below represents a boxplot of the errors obtained for each model after training on tuned hyper parameters. The Decision tree model seems to have a reduced error which becomes our best selection.



13.BIAS VS VARIANCE

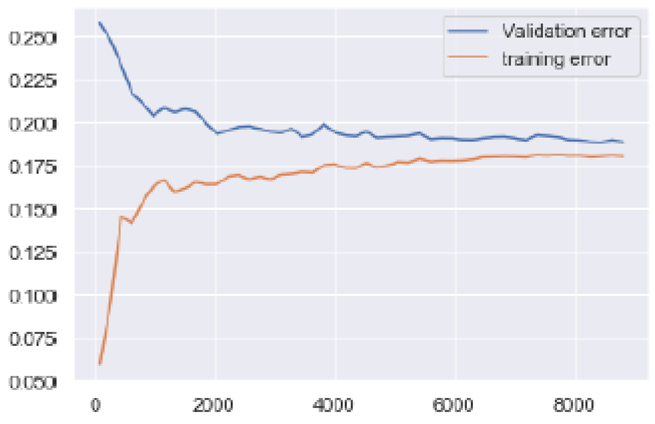
From the learning curve we could see that the Decision Tree has a considerably more variance. So we try to reduce the variance. As there is a trade-off between variance and error, we try to reduce the variance without increasing the error. A considerable decrease in the variance with the same error was achieved in this case for the following hyper parameters.

```
print('Decision Tree Classifier')
clf1 = tree.DecisionTreeClassifier(max_depth =7, criterion = 'gini', splitter = 'best')
clf1.fit(X_train,Y_train)
scores_1 = cross_val_score(clf1, X_train, Y_train, cv=5)
print("Accuracy: %f (+/- %0.2f)" % (scores_1.mean(), scores_1.std() * 2))
train_sizes, train_scores,validation_scores = learning_curve(clf1,X_train,Y_train,cv=5,
scoring='accuracy', n_jobs=-1,train_sizes=np.linspace(0.01, 1.0, 50)
)
train_mean = np.mean(1-train_scores, axis=1)
validation_mean = np.mean(1-validation_scores, axis=1)
plt.plot(train_sizes, validation_mean, label = 'Validation error')
plt.plot(train_sizes, train_mean, label = 'training error')
plt.legend()
```

```
Decision Tree Classifier
Accuracy: 0.811164 (+/- 0.01)
```

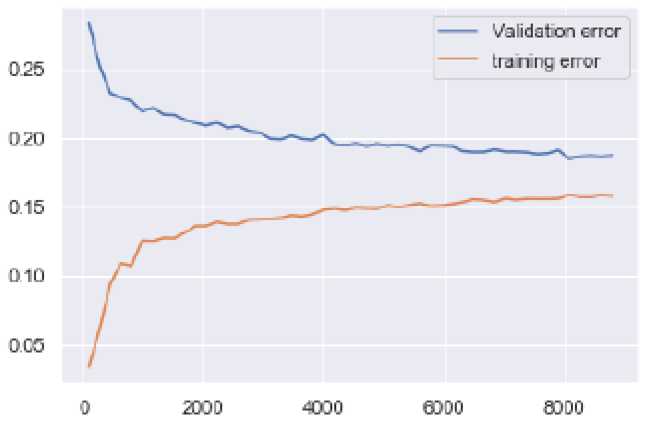
```
Out[54]:
```

```
<matplotlib.legend.Legend at 0x1984ff15a20>
```



$\longrightarrow$ Reduced variance

```
<matplotlib.legend.Legend at 0x1984fb8b198>
```



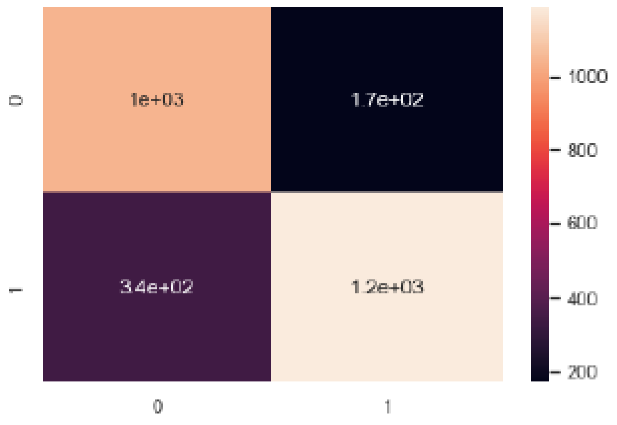
$\longrightarrow$ Initial variance

14. PERFORMANCE EVALUATION

We tested the final trained Decision Tree on the test data and below were the results obtained.

```
print('Decision Tree Classifier')
Y_pred = clf1.predict(X_test)
from sklearn import metrics
print('Accuraccy : %f' % metrics.accuracy_score(Y_test, Y_pred))
cm = confusion_matrix(Y_test,Y_pred)
cr = classification_report(Y_test,Y_pred)
sns.heatmap(cm,annot=True,cbar=True,xticklabels='auto',yticklabels='auto')
print(cr)
```

```
Decision Tree Classifier
Accuraccy : 0.814545
              precision    recall  f1-score   support

           0       0.75      0.86      0.80      1216
           1       0.88      0.78      0.82      1534

    accuracy                           0.81      2750
   macro avg       0.82      0.82      0.81      2750
weighted avg       0.82      0.81      0.82      2750
```

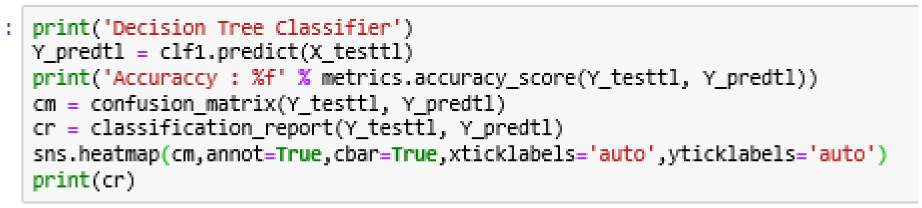


15. TRANSFER LEARNING

The same trained Decision tree model was used to predict on another list of URLs and below were the accuracy achieved by the model. This dataset contained 6500 URLs labelled as 0 or 1. The same feature creation technique was used on these URL's and prediction was made by the model on these obtained features.

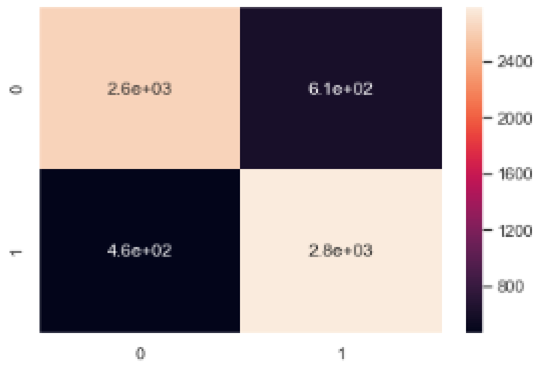
```
X_testtl=featureSettl.iloc[:,1:11].values
Y_testtl=featureSettl.iloc[:,11].values
print(X_testtl.shape)
print(Y_testtl.shape)
```

```
(6500, 10)
(6500,)
```



```
print('Decision Tree Classifier')
Y_predtl = clf1.predict(X_testtl)
print('Accuraccy : %f' % metrics.accuracy_score(Y_testtl, Y_predtl))
cm = confusion_matrix(Y_testtl, Y_predtl)
cr = classification_report(Y_testtl, Y_predtl)
sns.heatmap(cm,annot=True,cbar=True,xticklabels='auto',yticklabels='auto')
print(cr)
```

```
Decision Tree Classifier
Accuraccy : 0.835077
              precision    recall  f1-score   support

           0       0.85      0.81      0.83      3251
           1       0.82      0.86      0.84      3249

    accuracy                           0.84      6500
   macro avg       0.84      0.84      0.83      6500
weighted avg       0.84      0.84      0.83      6500
```



During transfer learning when the model was exposed to new data set the accuracy increase compared to one obtained during testing because of considerable decrease in the variance which helped to generalize the model well.

16. HARDWARE AND SOFTWARE REQUIREMENTS

Hardware Requirements:

RAM: 4GB and Higher

Processor: Intel i3 and above

Hard Disk: 500GB: Minimum

Software Requirements:

Python IDE: python 2.7.x and above

Anaconda

Setup tools and pip to be installed for 3.6 and above

Language: Python Scripting

OS: Windows or Linux

17. DISCUSSION

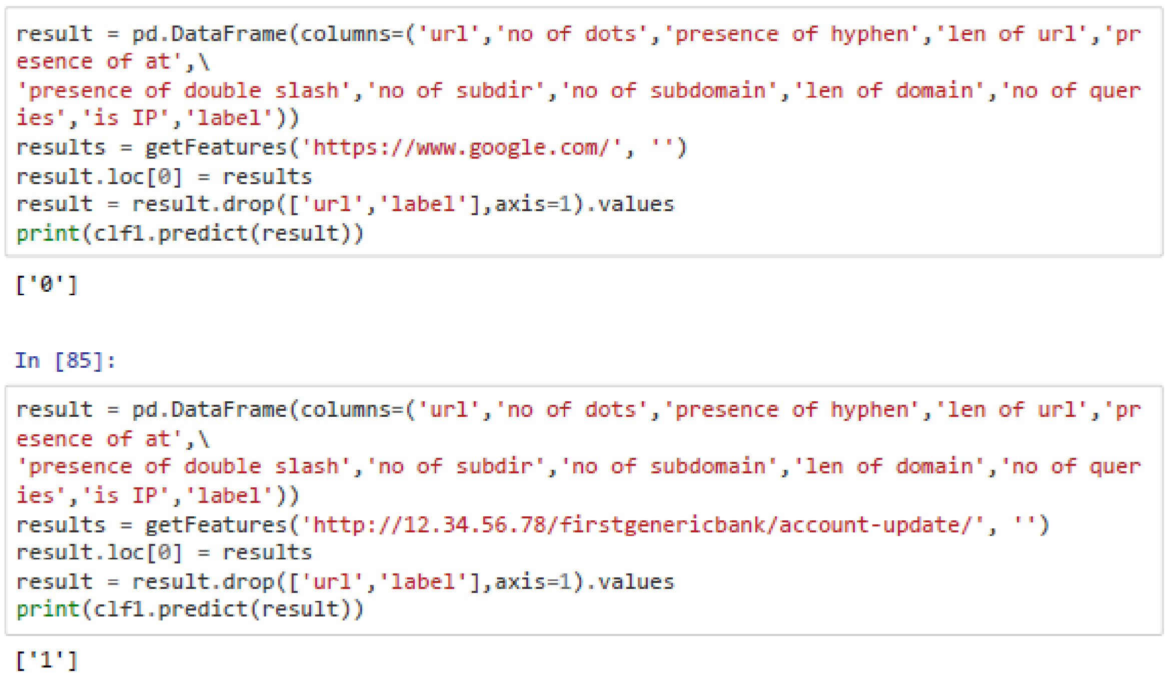
The Decision tree works well for our dataset because the number of classes to predict is less (label 0/1). Also the performance of a decision tree increase with more number of training examples. As our dataset have more instances the accuracy is relatively high for this classifier.

The Gaussian NB did not work well on this dataset as the features created from the url are not independent and they are interlinked in predicting a class label (0/1).

Rest of the classifiers like Adaboost, Linear Regression and KNN have a considerably close accuracy.

18. DEMO

The below shows a working demo of the Decision Tree model in predicting the class of a given label.



```
result = pd.DataFrame(columns=('url','no of dots','presence of hyphen','len of url','pr
esence of at',\
'presence of double slash','no of subdir','no of subdomain','len of domain','no of quer
ies','is IP','label'))
results = getFeatures('https://www.google.com/', '')
result.loc[0] = results
result = result.drop(['url','label'],axis=1).values
print(clf1.predict(result))
```

```
['0']
```

```
In [85]:
```

```
result = pd.DataFrame(columns=('url','no of dots','presence of hyphen','len of url','pr
esence of at',\
'presence of double slash','no of subdir','no of subdomain','len of domain','no of quer
ies','is IP','label'))
results = getFeatures('http://12.34.56.78/firstgenericbank/account-update/', '')
result.loc[0] = results
result = result.drop(['url','label'],axis=1).values
print(clf1.predict(result))
```

```
['1']
```