PHISHING URL's PREDICTION

2228 INSY 5377 Web and Social Analytics

TEJESH VARMA RUDRARAJU

Table of Contents

Introduction:

Phishing is a type of fraudulent attack where the attacker pretends to be a reliable source to obtain sensitive information. A victim clicks on a hacked link that pretends to be a legitimate website in a conventional phishing assault. The victim is then prompted to submit their login information, but because it is a "false" website, the private data is sent to the hacker instead, and the victim is "hacked”. Due to its cheap effort and huge reward, phishing is a common form of assault. Most contemporary web browsers, antivirus programs, and email clients are excellent at spotting phishing websites at the source and assisting in attack prevention. The objective of this project is to use the dataset produced to predict phishing websites to train machine learning models and deep neural networks. To create a dataset from which the necessary URL- and website content-based attributes may be extracted, both phishing and benign URLs of websites are collected. Each model's performance level is assessed and contrasted.

Data Description:

We have taken our data set from Kaggle and below is the link to the dataset.

https://www.kaggle.com/datasets/taruntiwarihp/phishing-site-urls

- This dataset has 5,49,346 entries in it.
- There are two columns in the dataset.

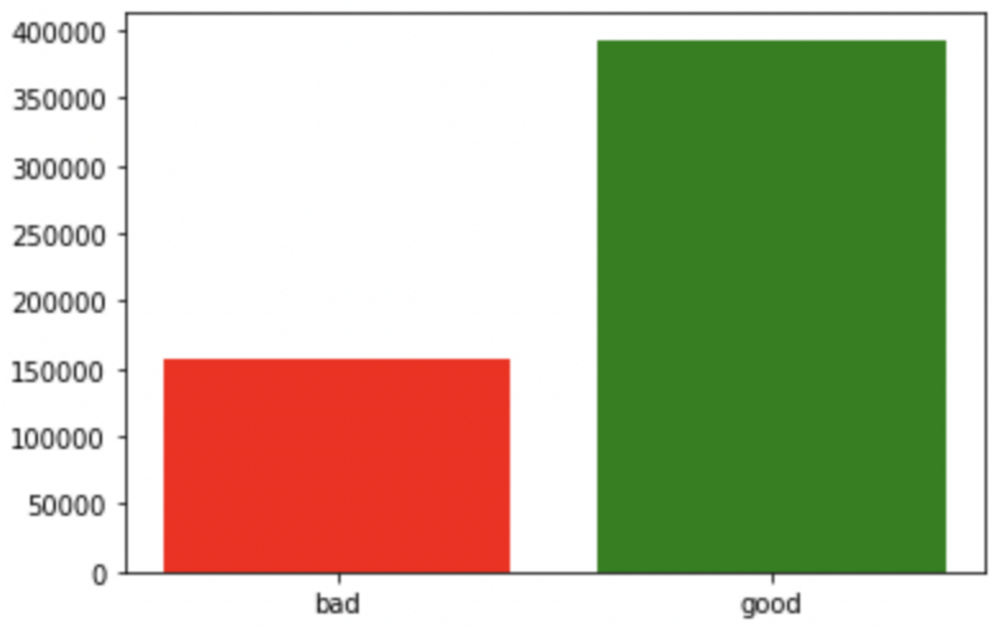
a) **URL:** This column has the name of the URL

b) **Label:** Label column is prediction col which has 2 categories

A. **Good** - which means the URLs is not containing malicious stuff and this site is not a Phishing Site.

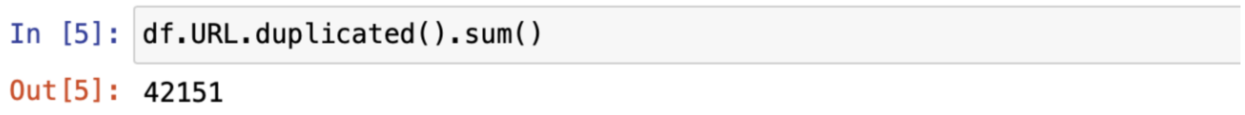
B. **Bad** - which means the URLs contains malicious stuff and this site is a Phishing Site.

Below is the bar graph representing the label column:



Data Cleaning:

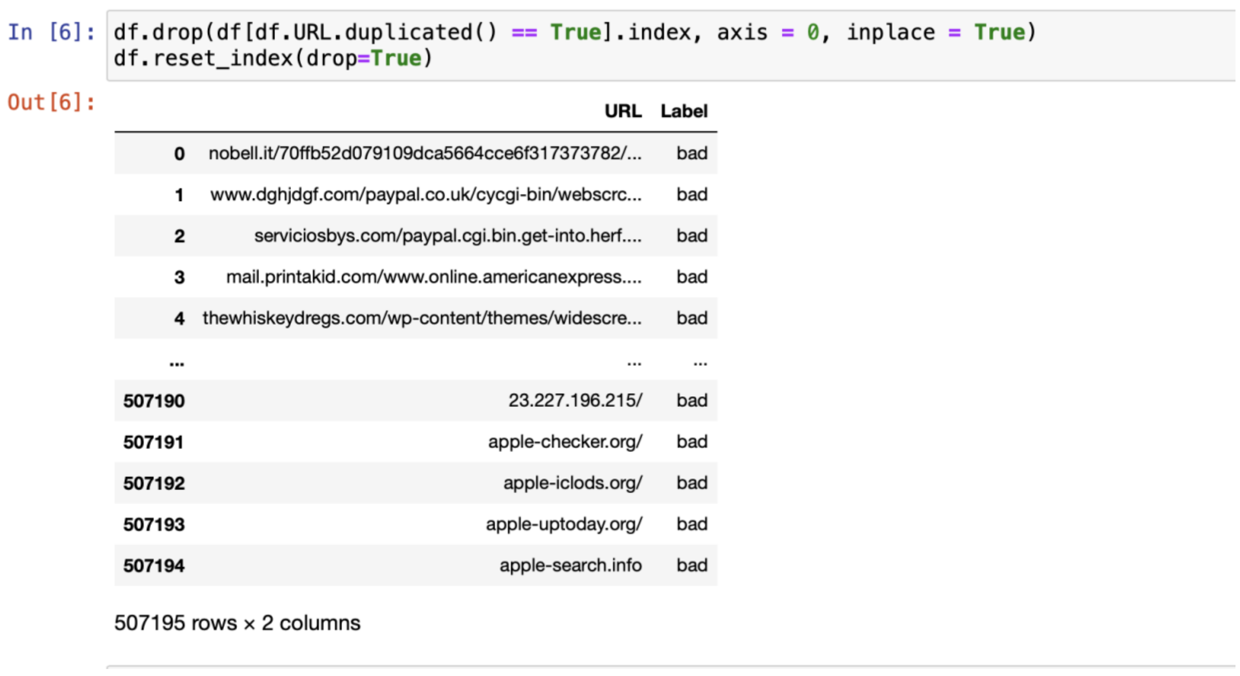
As a part of data cleaning, we have first checked for any duplicate entries present in the dataset and removed them.



```
In [5]: df.URL.duplicated().sum()
Out[5]: 42151
```

We found that there are a total of 42151 duplicate entries and dropped those entries.

The new dataset is formed after dropping the duplicate entries.



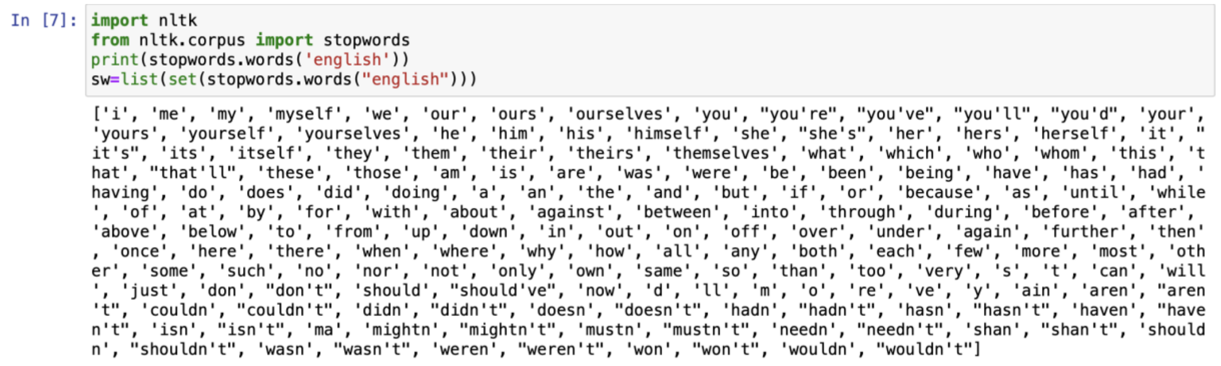
```
In [6]: df.drop(df[df.URL.duplicated() == True].index, axis = 0, inplace = True)
        df.reset_index(drop=True)
```

Out[6]:

	URL	Label
0	nobell.it/70ffb52d079109dca5664cce6f317373782/...	bad
1	www.dghjdgf.com/paypal.co.uk/cycgi-bin/webscrc...	bad
2	serviciosbys.com/paypal.cgi.bin.get-into.herf....	bad
3	mail.printakid.com/www.online.americanexpress....	bad
4	thewhiskeydregs.com/wp-content/themes/widescre...	bad
...	...	...
507190	23.227.196.215/	bad
507191	apple-checker.org/	bad
507192	apple-iclods.org/	bad
507193	apple-uptoday.org/	bad
507194	apple-search.info	bad

507195 rows × 2 columns

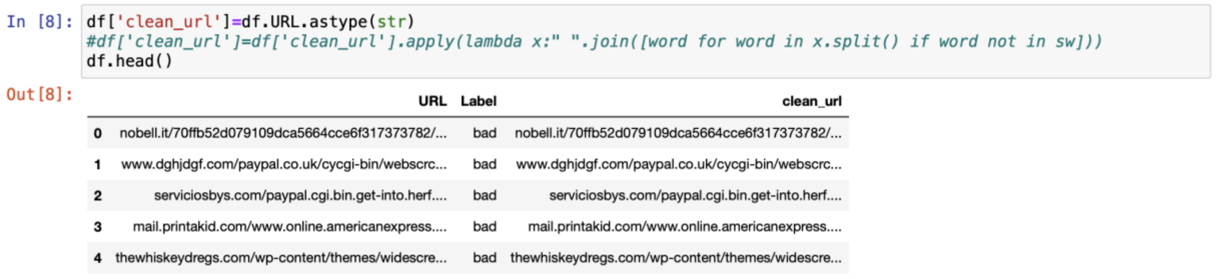
As the next step, we have removed the stopwords from the text present in the URL column of our dataset by importing nltk package.



```
In [7]: import nltk
        from nltk.corpus import stopwords
        print(stopwords.words('english'))
        sw=list(set(stopwords.words("english")))
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your',
'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "
it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 't
hat', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', '
having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while
', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',
'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then'
, 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'oth
er', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will
', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren
't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "have
n't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'should
n', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

We have then a created a new column to store the clean URL without any stopwords in it.

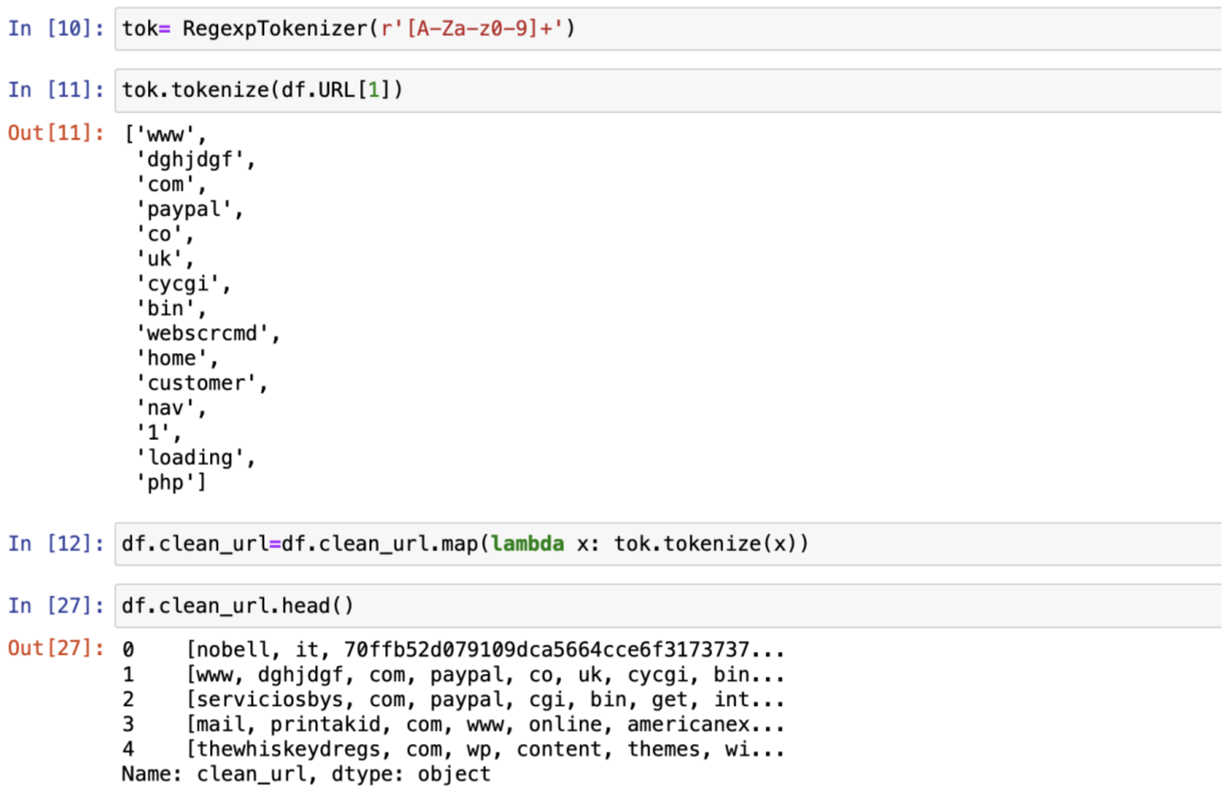


```
In [8]: df['clean_url']=df.URL.astype(str)
        #df['clean_url']=df['clean_url'].apply(lambda x:" ".join([word for word in x.split() if word not in sw]))
        df.head()
```

Out[8]:

	URL	Label	clean_url
0	nobell.it/70ffb52d079109dca5664cce6f317373782/...	bad	nobell.it/70ffb52d079109dca5664cce6f317373782/...
1	www.dghjdgf.com/paypal.co.uk/cycgi-bin/webscrc...	bad	www.dghjdgf.com/paypal.co.uk/cycgi-bin/webscrc...
2	serviciosbys.com/paypal.cgi.bin.get-into.herf....	bad	serviciosbys.com/paypal.cgi.bin.get-into.herf....
3	mail.printakid.com/www.online.americanexpress....	bad	mail.printakid.com/www.online.americanexpress....
4	thewhiskeydregs.com/wp-content/themes/widescre...	bad	thewhiskeydregs.com/wp-content/themes/widescre...

Moving forward with the data cleaning process, we have also used regular expressions tokenizer to remove all the unnecessary characters apart from alphabets and numbers. We have taken the clean URL column to use the regular expressions tokenizer.



```
In [10]: tok= RegexpTokenizer(r'[A-Za-z0-9]+')

In [11]: tok.tokenize(df.URL[1])

Out[11]: ['www',
          'dghjdgf',
          'com',
          'paypal',
          'co',
          'uk',
          'cycgi',
          'bin',
          'webscrcmd',
          'home',
          'customer',
          'nav',
          '1',
          'loading',
          'php']

In [12]: df.clean_url=df.clean_url.map(lambda x: tok.tokenize(x))

In [27]: df.clean_url.head()

Out[27]: 0    [nobell, it, 70ffb52d079109dca5664cce6f3173737...
         1    [www, dghjdgf, com, paypal, co, uk, cycgi, bin...
         2    [serviciosbys, com, paypal, cgi, bin, get, int...
         3    [mail, printakid, com, www, online, americanex...
         4    [thewhiskeydregs, com, wp, content, themes, wi...
         Name: clean_url, dtype: object
```

Research Questions:

We have done our analysis on detecting whether a URL is good or bad to answer the following questions.

- To create a classification model to identify whether a URL is a good or a bad URL.
- To find whether there are any similarities in all bad URL's.

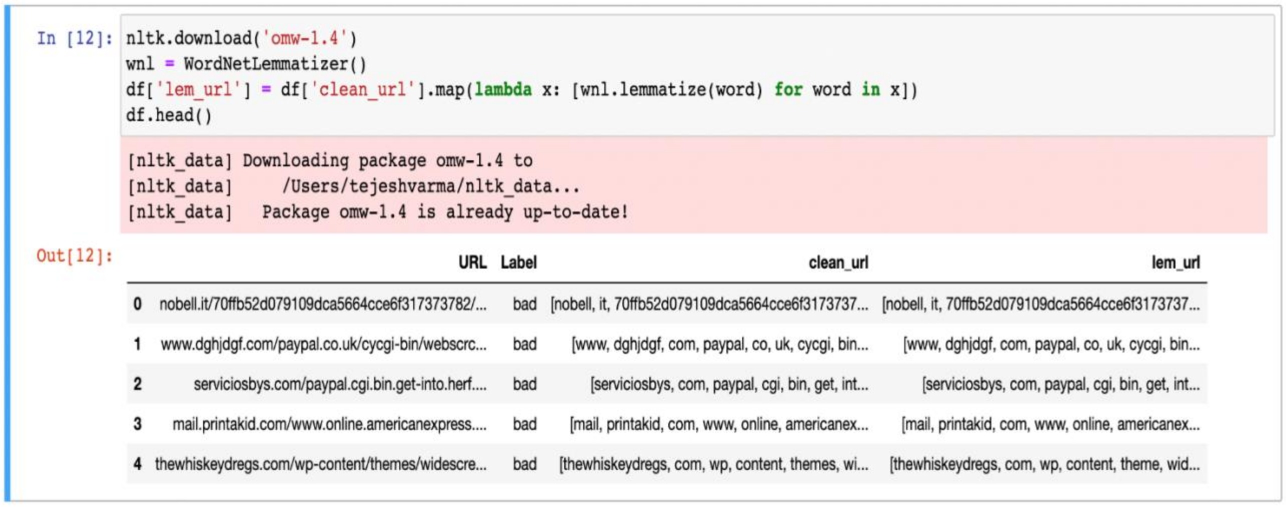
Methodologies:

We have followed the below steps in detecting whether the URL is a good or bad.

- **Word Lemmatization**
- **Vectorizing Using TF-IDF**
- **Vectorizing Using Count Vectorizer**
- **Generating Word Clouds**
- **Clustering**
- **Using Various Machine Learning Models**

<u>Word lemmatization:</u>

After Cleaning the URL's, we have used word lemmatization to group the similar words of URLS so that we can group similar type of words in URL's together and remove unwanted noise from the URL's.



```
In [12]: nltk.download('omw-1.4')
         wnl = WordNetLemmatizer()
         df['lem_url'] = df['clean_url'].map(lambda x: [wnl.lemmatize(word) for word in x])
         df.head()

         [nltk_data] Downloading package omw-1.4 to
         [nltk_data]     /Users/tejeshvarma/nltk_data...
         [nltk_data]   Package omw-1.4 is already up-to-date!
```

Out[12]:

	URL	Label	clean_url	lem_url
0	nobell.it/70ffb52d079109dca5664cce6f317373782/...	bad	[nobell, it, 70ffb52d079109dca5664cce6f3173737...	[nobell, it, 70ffb52d079109dca5664cce6f3173737...
1	www.dghjdgf.com/paypal.co.uk/cycgi-bin/webscrc...	bad	[www, dghjdgf, com, paypal, co, uk, cycgi, bin...	[www, dghjdgf, com, paypal, co, uk, cycgi, bin...
2	serviciosbys.com/paypal.cgi.bin.get-into.herf....	bad	[serviciosbys, com, paypal, cgi, bin, get, int...	[serviciosbys, com, paypal, cgi, bin, get, int...
3	mail.printakid.com/www.online.americanexpress....	bad	[mail, printakid, com, www, online, americanex...	[mail, printakid, com, www, online, americanex...
4	thewhiskeydregs.com/wp-content/themes/widescre...	bad	[thewhiskeydregs, com, wp, content, themes, wi...	[thewhiskeydregs, com, wp, content, theme, wid...

<u>**Vectorization using TF-IDF:**</u>

After lemmatizing the URL's, we have calculated the TF-IDF score of every word in the URL's and added in a vector form using the using tf-idf vectorizer.

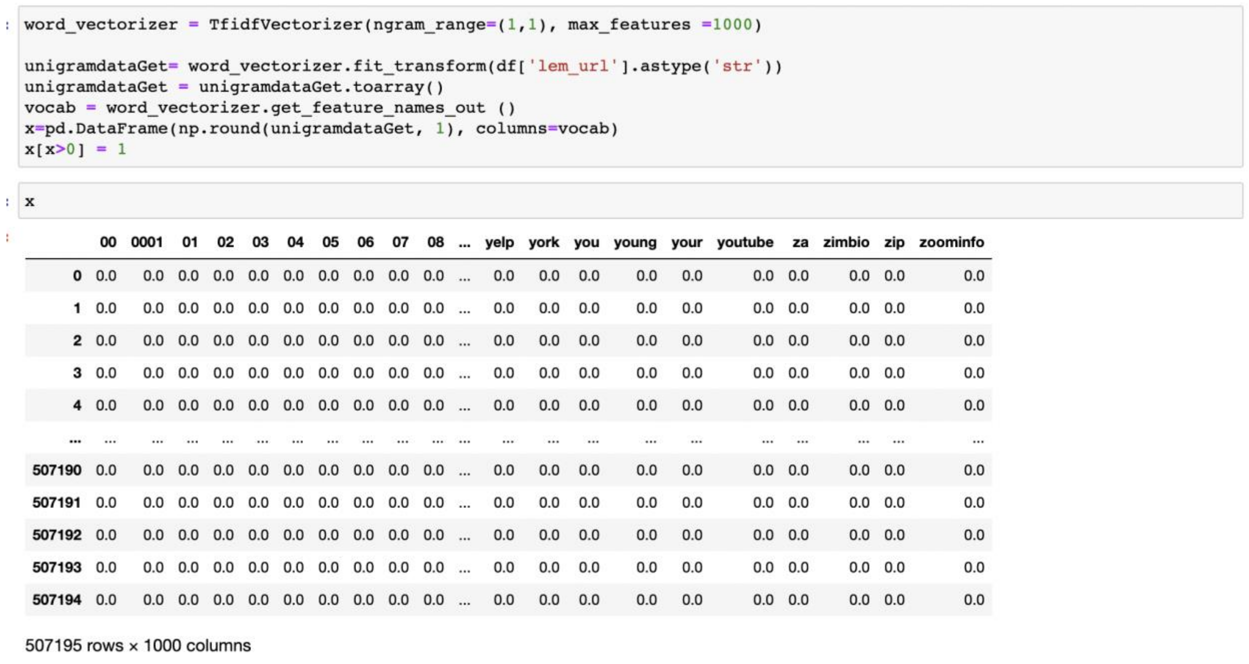
The TF-IDF score is calculated using the following formula

$$TF(w,d) = \frac{occurences\ of\ w\ in\ document\ d}{total\ number\ of\ words\ in\ document\ d}$$

$$IDF(w,D) = \ln(\frac{Total\ number\ of\ documents\ (N)\ in\ corpus\ D}{number\ of\ documents\ containing\ w})$$

$$TFIDF\ (w,d,D) = TF(w,d) * IDF(w,D)$$

After using the TF-IDF vectorizer this is the result vector we got



```
word_vectorizer = TfidfVectorizer(ngram_range=(1,1), max_features =1000)

unigramdataGet= word_vectorizer.fit_transform(df['lem_url'].astype('str'))
unigramdataGet = unigramdataGet.toarray()
vocab = word_vectorizer.get_feature_names_out ()
x=pd.DataFrame(np.round(unigramdataGet, 1), columns=vocab)
x[x>0] = 1
```

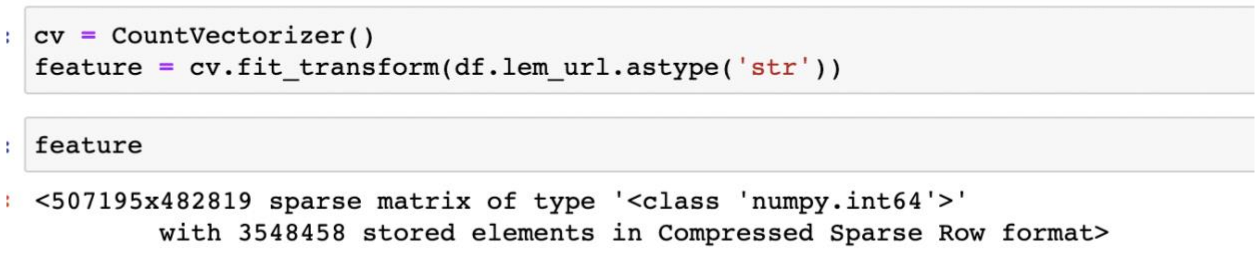
```
x
```

	00	0001	01	02	03	04	05	06	07	08	...	yelp	york	you	young	your	youtube	za	zimbio	zip	zoominfo
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
507190	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
507191	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
507192	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
507193	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
507194	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

507195 rows × 1000 columns

Vectorization using Count Vectorizer:

This Process is also called feature extraction. By using count vectorizer to transform a given text into a vector based on the frequency of each word occurs in the entire text. The difference between count vectorizer and tf-idf vectorizer is that count vectorizer performs task of tokenization and counting while tf-idf usually used for normalization of data.



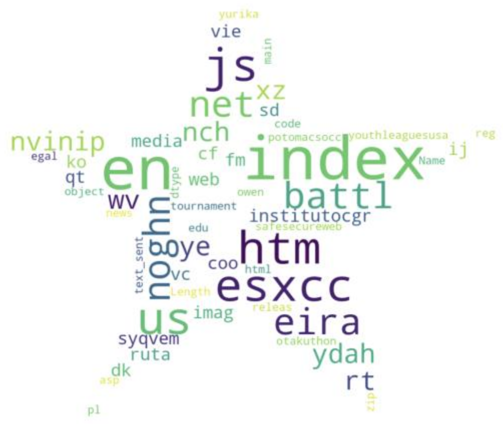
```
cv = CountVectorizer()
feature = cv.fit_transform(df.lem_url.astype('str'))
```

```
feature
```

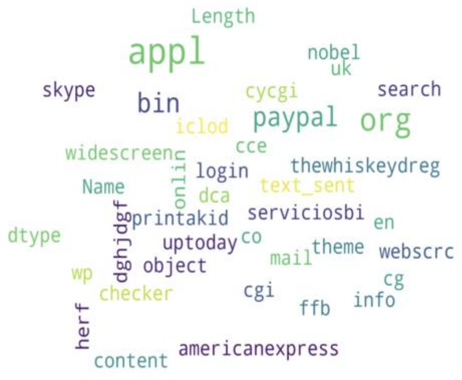
```
<507195x482819 sparse matrix of type '<class 'numpy.int64'>'
        with 3548458 stored elements in Compressed Sparse Row format>
```

Word Clouds:

Later we formed word clouds to see which words are highly repeated or less repeated. If the words in the word cloud are large it represents that they are repeated more times in the text or the URLs in our case.



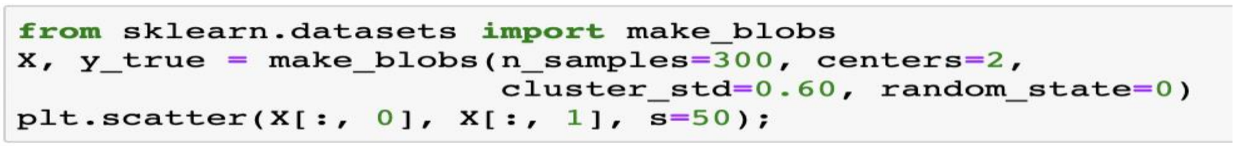
Good URL Word Cloud



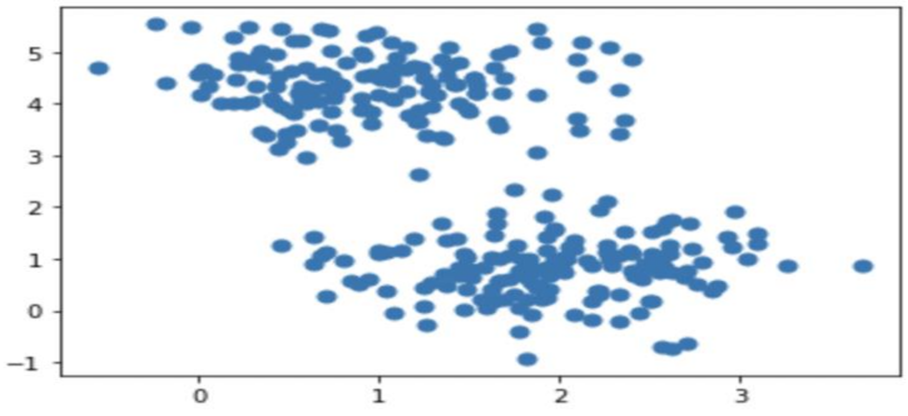
Bad URL Word

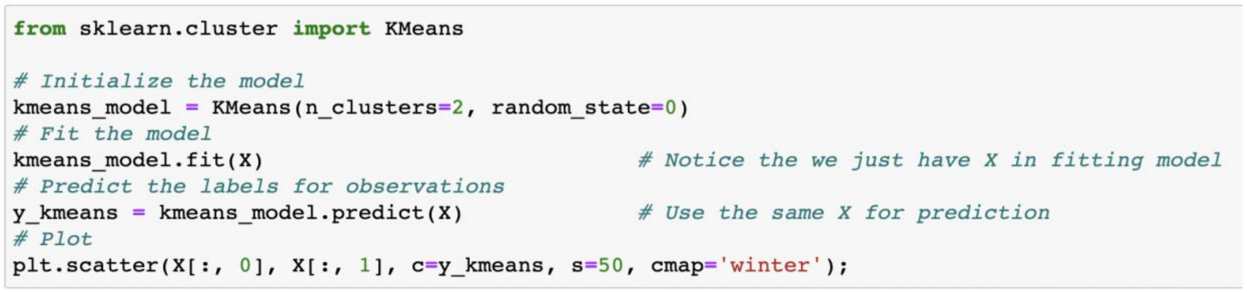
Clustering:

Since professor asked us to remove the label column, we removed the label column and we wanted to see with the use of vectorizers we can cluster the data without using the label column, so we applied K-means clustering for the new data without the label column and this we what we found.

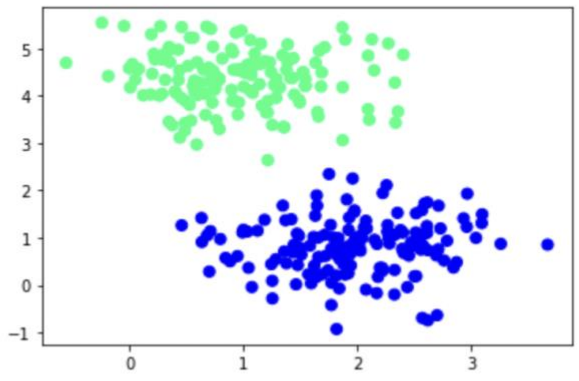


```
from sklearn.datasets import make_blobs
X, y_true = make_blobs(n_samples=300, centers=2,
                       cluster_std=0.60, random_state=0)
plt.scatter(X[:, 0], X[:, 1], s=50);
```





```
from sklearn.cluster import KMeans

# Initialize the model
kmeans_model = KMeans(n_clusters=2, random_state=0)
# Fit the model
kmeans_model.fit(X)                          # Notice the we just have X in fitting model
# Predict the labels for observations
y_kmeans = kmeans_model.predict(X)           # Use the same X for prediction
# Plot
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='winter');
```

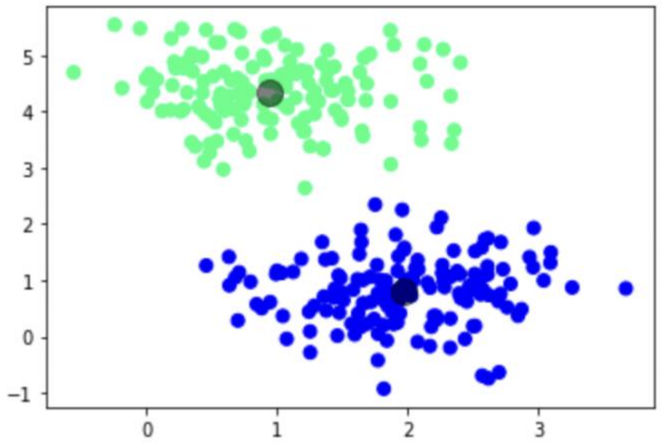


Later we found cluster centers for both the clusters and marked it on the scatter plot

```
kmeans_model.cluster_centers_
```

```
array([[1.9674274 , 0.81501647],
       [0.94060243, 4.34916815]])
```

```
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='winter');
centers = kmeans_model.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5);
```



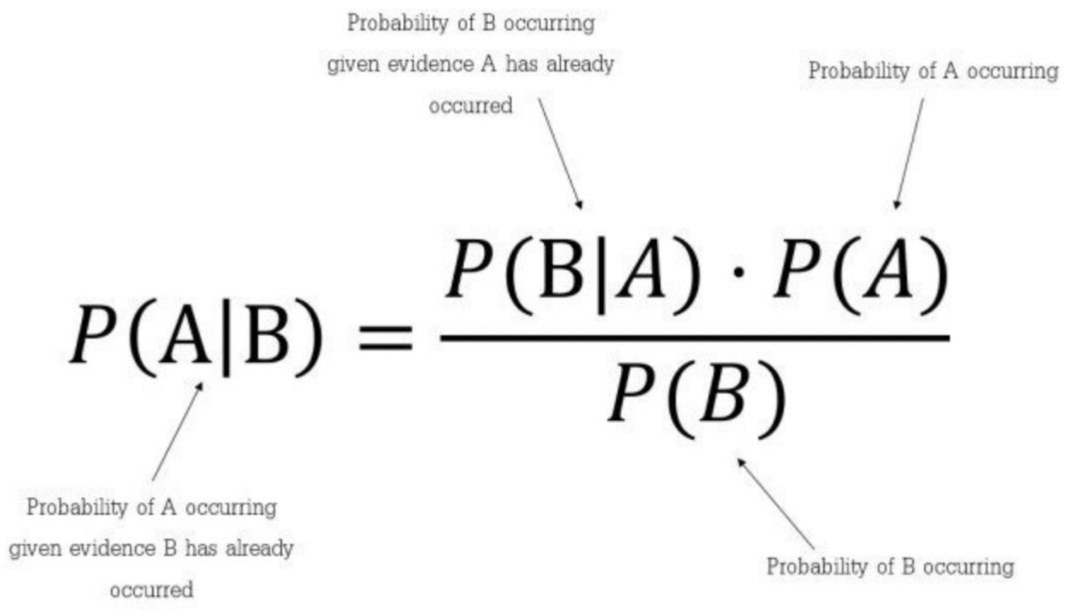
Machine Learning Models

Once the clustering is done, we then performed classification models to check the accuracy and other metrics to determine if the URL is good or bad. The models we chose are

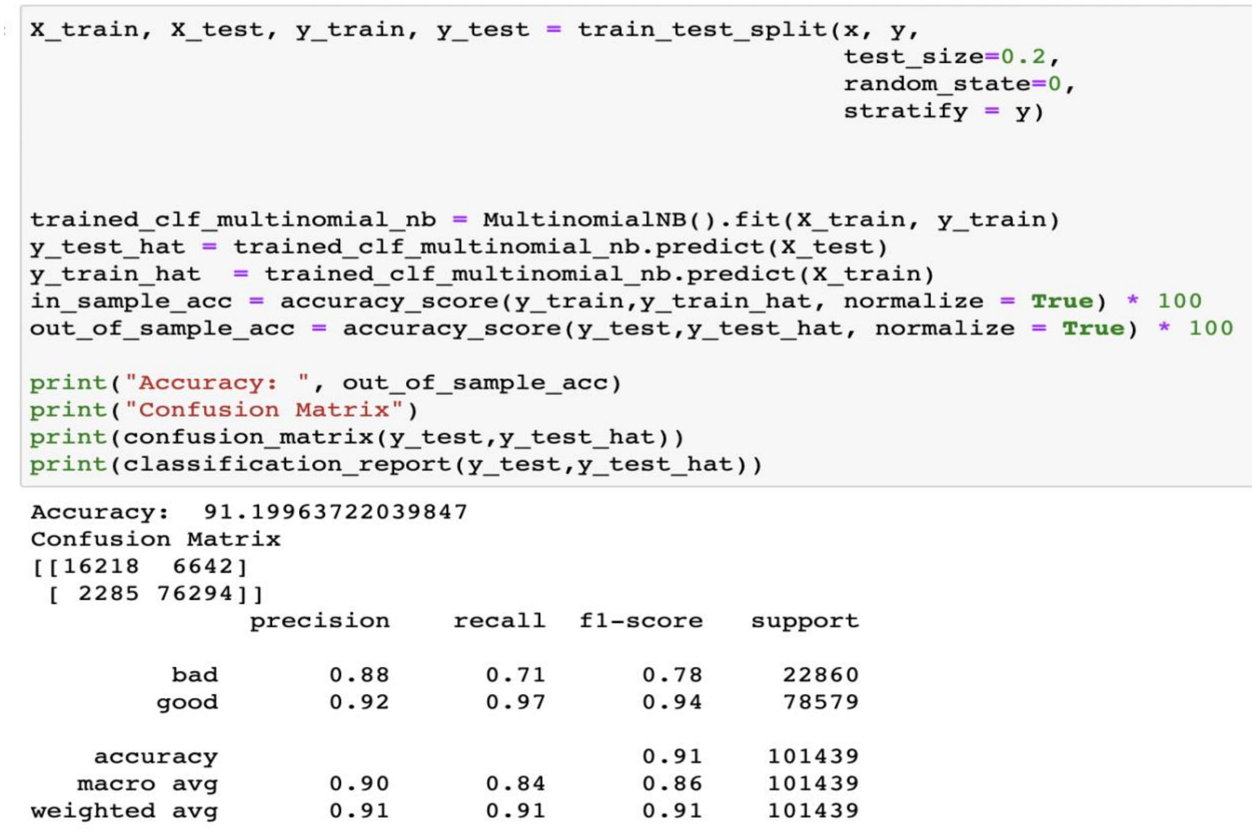
1. Naïve Bayes
2. Logistic Regression
3. Decision Tree

Naïve Bayes

Naïve Bayes is generally used for classification tasks, and it is one of the simplest models. It is fast, accurate and reliable and works great for Natural Language Processing (NLP) problems. The formula is


Probability of B occurring given evidence A has already occurred
Probability of A occurring
$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$
Probability of A occurring given evidence B has already occurred
Probability of B occurring

We performed this model for our dataset and got the below results.



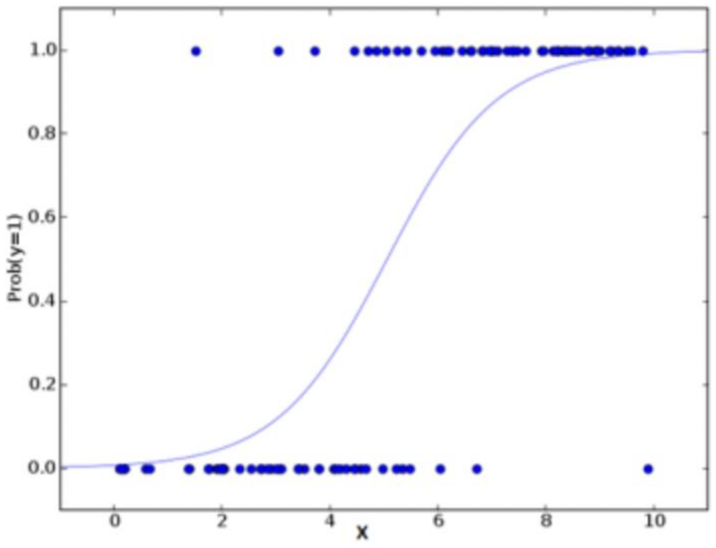
```
X_train, X_test, y_train, y_test = train_test_split(x, y,
                                                    test_size=0.2,
                                                    random_state=0,
                                                    stratify = y)



trained_clf_multinomial_nb = MultinomialNB().fit(X_train, y_train)
y_test_hat = trained_clf_multinomial_nb.predict(X_test)
y_train_hat  = trained_clf_multinomial_nb.predict(X_train)
in_sample_acc = accuracy_score(y_train,y_train_hat, normalize = True) * 100
out_of_sample_acc = accuracy_score(y_test,y_test_hat, normalize = True) * 100

print("Accuracy: ", out_of_sample_acc)
print("Confusion Matrix")
print(confusion_matrix(y_test,y_test_hat))
print(classification_report(y_test,y_test_hat))
```

```
Accuracy:  91.19963722039847
Confusion Matrix
[[16218  6642]
 [ 2285 76294]]
              precision    recall  f1-score   support

         bad       0.88      0.71      0.78     22860
        good       0.92      0.97      0.94     78579

    accuracy                           0.91    101439
   macro avg       0.90      0.84      0.86    101439
weighted avg       0.91      0.91      0.91    101439
```

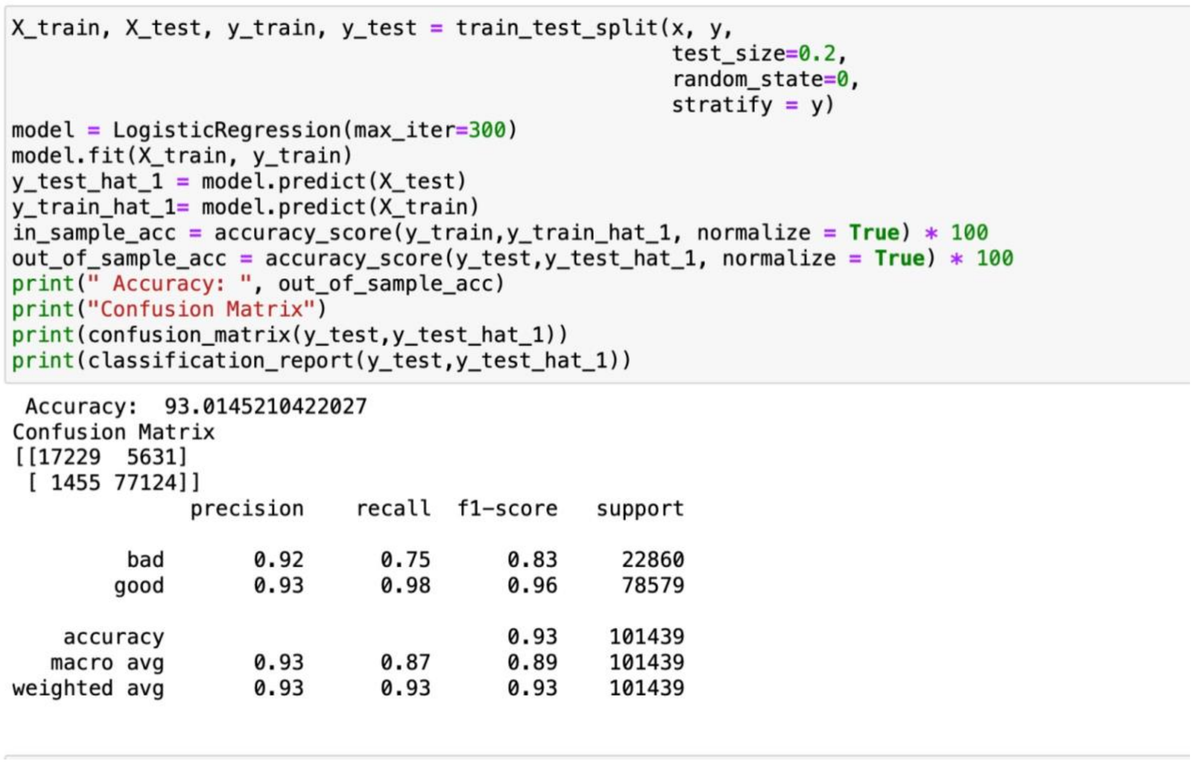
The accuracy was 91.19, precision was 0.92, recall was 0.97 and f1 score as 0.94 respectively.

Logistic Regression

Later we tried Logistic Regression, it is one of the most efficient methods for binary and linear classification models. It would be easy and simple as the results would be achieved very well because of its great performance with linearly separable classes. It not only limits to binary classification but also can be used for multiclass classification. The probability will never go below 0 and above 1. Below is the model plot for reference.



Logistic Regression is performed for our dataset and below is the outcome



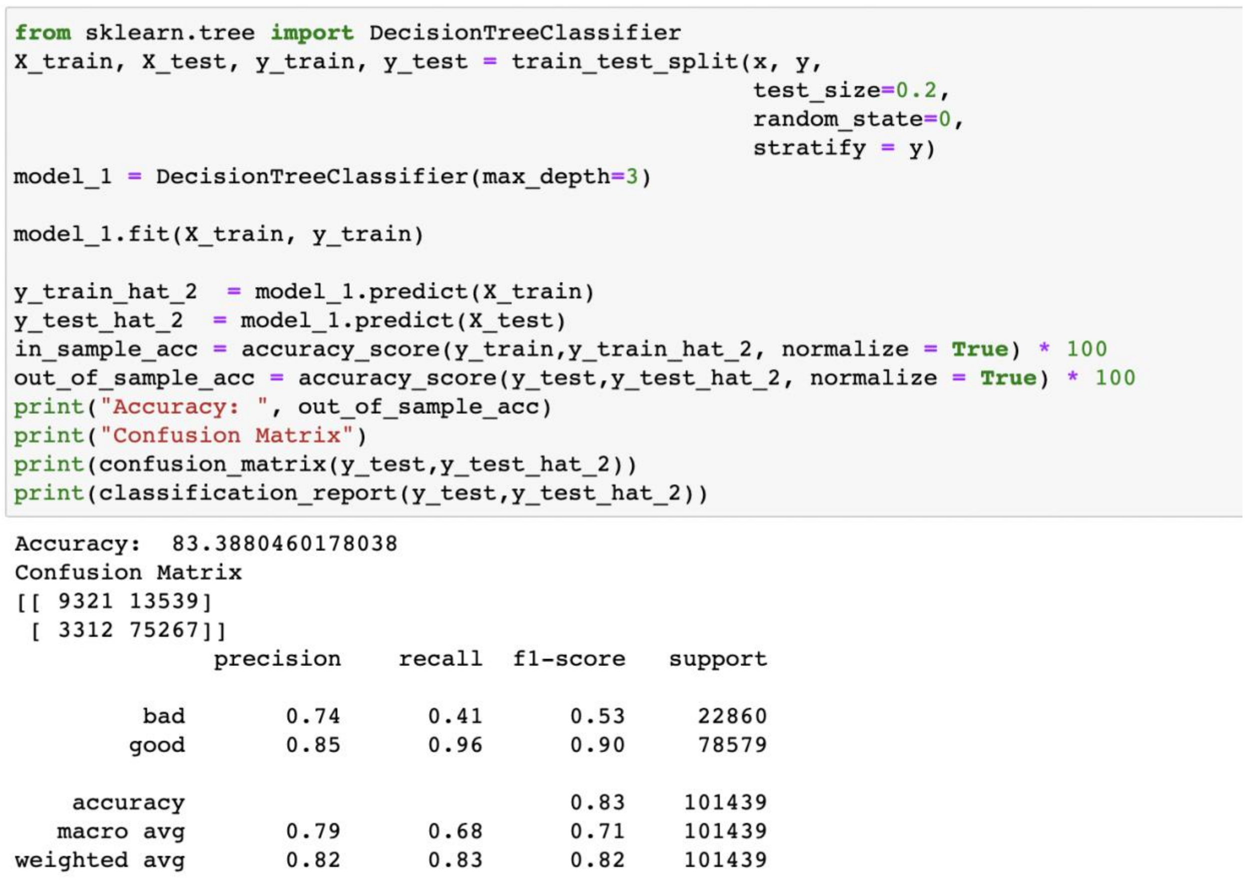
```
X_train, X_test, y_train, y_test = train_test_split(x, y,
                                                    test_size=0.2,
                                                    random_state=0,
                                                    stratify = y)
model = LogisticRegression(max_iter=300)
model.fit(X_train, y_train)
y_test_hat_1 = model.predict(X_test)
y_train_hat_1= model.predict(X_train)
in_sample_acc = accuracy_score(y_train,y_train_hat_1, normalize = True) * 100
out_of_sample_acc = accuracy_score(y_test,y_test_hat_1, normalize = True) * 100
print(" Accuracy: ", out_of_sample_acc)
print("Confusion Matrix")
print(confusion_matrix(y_test,y_test_hat_1))
print(classification_report(y_test,y_test_hat_1))
```

```
 Accuracy:  93.0145210422027
Confusion Matrix
[[17229  5631]
 [ 1455 77124]]
              precision    recall  f1-score   support

         bad       0.92      0.75      0.83     22860
        good       0.93      0.98      0.96     78579

    accuracy                           0.93    101439
   macro avg       0.93      0.87      0.89    101439
weighted avg       0.93      0.93      0.93    101439
```

The accuracy is 93.01, precision is 0.93, recall is 0.98 and f1 score as 0.96 respectively.

Decision Tree

It is a classification model that is used to predict or draw conclusions about a set of observations and is also known as Classification tree. This is a non-parametric supervised learning method used for both classification and regression tasks. We performed the below code for decision tree and got the output accordingly.



```
from sklearn.tree import DecisionTreeClassifier
X_train, X_test, y_train, y_test = train_test_split(x, y,
                                                    test_size=0.2,
                                                    random_state=0,
                                                    stratify = y)
model_1 = DecisionTreeClassifier(max_depth=3)

model_1.fit(X_train, y_train)

y_train_hat_2  = model_1.predict(X_train)
y_test_hat_2  = model_1.predict(X_test)
in_sample_acc = accuracy_score(y_train,y_train_hat_2, normalize = True) * 100
out_of_sample_acc = accuracy_score(y_test,y_test_hat_2, normalize = True) * 100
print("Accuracy: ", out_of_sample_acc)
print("Confusion Matrix")
print(confusion_matrix(y_test,y_test_hat_2))
print(classification_report(y_test,y_test_hat_2))
```

```
Accuracy:  83.3880460178038
Confusion Matrix
[[ 9321 13539]
 [ 3312 75267]]
              precision    recall  f1-score   support

         bad       0.74      0.41      0.53     22860
        good       0.85      0.96      0.90     78579

    accuracy                           0.83    101439
   macro avg       0.79      0.68      0.71    101439
weighted avg       0.82      0.83      0.82    101439
```

The accuracy is 83.38, precision is 0.85, recall is 0.96 and f1 score as 0.90 respectively.

Results & Conclusions:

In our research about the Phishing URLs and the similarities between them, we have discovered that Bad Phishing URL's have common type words/texts used for example they contain most of the payment links such as PayPal or credit card links in them and seem to procrastinate some legit websites. Further we after removing the label column and applied the various machine learning models. We have found that logistic regression was able to give the highest accuracy out of all three models as its one of the most efficient models that can be used for binary data. We have considered accuracy as our main preference as we wanted the machine learning algorithm to separate bad and good URL.

Acknowledgements and References:

The users can also identify the bad URLs by following the below protocols

- Consider the Source
- Spellcheck the address
- Vet that URL before you click
- Investigate the website
- Proofread that site
- Dataset: **https://www.kaggle.com/datasets/taruntiwarihp/phishing-site-urls**
- Data science Text Analysis - PI, PII.ipynb file for reference
- Data science Supervised Learning PI, PII, PIII .ipynbfile for reference
- To check and report phishing websites. Here's the link **phishing-report@us-cert.gov**
- UTA students can also report the bad URLs to the university at **phish@uta.edu**