Naive Bayes Classification of Phishing Websites

Notebook adapted from the 05.05 Naive Bayes notebook from the Python Data Science Handbook.
The following website was also used in making this notebook: BAIT 509
Modified by: Gábor Major
Last Modified date: 2025-02-02

Import libraries.

```
from scipy.io import arff
import pandas as pd
```

Import Data

The phishing website data was downloaded from UC Irvine Machine Learning Repository.
An explanation of the features of the dataset is available in the DOCX file provided with the data.
Load in the data, as the arff file.

```
data = arff.loadarff('../phishing_websites_data/Training
Dataset.arff')
df = pd.DataFrame(data[0])
df.head()
```

```
  having_IP_Address URL_Length Shortining_Service having_At_Symbol  \
0             b'-1'       b'1'               b'1'             b'1'
1              b'1'       b'1'               b'1'             b'1'
2              b'1'       b'0'               b'1'             b'1'
3              b'1'       b'0'               b'1'             b'1'
4              b'1'       b'0'              b'-1'             b'1'

  double_slash_redirecting Prefix_Suffix having_Sub_Domain
SSLfinal_State  \
0                    b'-1'         b'-1'             b'-1'
b'-1'
1                     b'1'         b'-1'              b'0'
b'1'
2                     b'1'         b'-1'             b'-1'
b'-1'
3                     b'1'         b'-1'             b'-1'
b'-1'
4                     b'1'         b'-1'              b'1'
b'1'

  Domain_registeration_length Favicon  ... popUpWidnow Iframe
age_of_domain  \
0                       b'-1'    b'1'  ...        b'1'   b'1'
```

```
b'-1'
1                      b'-1'    b'1'  ...        b'1'   b'1'
b'-1'
2                      b'-1'    b'1'  ...        b'1'   b'1'
b'1'
3                       b'1'    b'1'  ...        b'1'   b'1'
b'-1'
4                      b'-1'    b'1'  ...       b'-1'   b'1'
b'-1'

  DNSRecord web_traffic Page_Rank Google_Index Links_pointing_to_page
\
0     b'-1'       b'-1'     b'-1'         b'1'                    b'1'

1     b'-1'        b'0'     b'-1'         b'1'                    b'1'

2     b'-1'        b'1'     b'-1'         b'1'                    b'0'

3     b'-1'        b'1'     b'-1'         b'1'                   b'-1'

4     b'-1'        b'0'     b'-1'         b'1'                    b'1'


  Statistical_report Result
0              b'-1'  b'-1'
1               b'1'  b'-1'
2              b'-1'  b'-1'
3               b'1'  b'-1'
4               b'1'   b'1'

[5 rows x 31 columns]
```

Clean up data by changing the encoding to remove the 'b', and add 1 to each value as negative values are not allowed in the model.

```
df = df.select_dtypes([object])
df = df.stack().str.decode('utf-8').unstack()
df = df.apply(pd.to_numeric)
df = df.add(1)
df.head()

   having_IP_Address  URL_Length  Shortining_Service  having_At_Symbol
\
0                  0           2                   2                 2

1                  2           2                   2                 2

2                  2           1                   2                 2

3                  2           1                   2                 2
```

```
4                    2           1                    0                    2


   double_slash_redirecting  Prefix_Suffix  having_Sub_Domain
SSLfinal_State  \
0                         0              0                   0
0
1                         2              0                   1
2
2                         2              0                   0
0
3                         2              0                   0
0
4                         2              0                   2
2

   Domain_registeration_length  Favicon  ...  popUpWidnow  Iframe  \
0                            0        2  ...            2       2
1                            0        2  ...            2       2
2                            0        2  ...            2       2
3                            2        2  ...            2       2
4                            0        2  ...            0       2

   age_of_domain  DNSRecord  web_traffic  Page_Rank  Google_Index  \
0              0          0            0          0             2
1              0          0            1          0             2
2              2          0            2          0             2
3              0          0            2          0             2
4              0          0            1          0             2

   Links_pointing_to_page  Statistical_report  Result
0                       2                   0       0
1                       2                   2       0
2                       1                   0       0
3                       0                   2       0
4                       2                   2       2

[5 rows x 31 columns]
```

Create Data Sets

Split data into 60% training, 20% validation, and 20% testing sets.

```
data_target = df['Result']
data_features = df.drop(columns=['Result'])

from sklearn.model_selection import train_test_split
# Split off 20% test set
```

```
xTrain, xTest, yTrain, yTest = train_test_split(data_features,
data_target, test_size=0.2)
# Split 80% of full data into 60% and 20% sets
xTrain, xValidation, yTrain, yValidation = train_test_split(xTrain,
yTrain, test_size=0.25)
```

Create Model

Use all the Bernoulli Naive Bayes model as this model performs the best on binary data, and it also takes into account not just the presence of data byt also the absence.

```
from sklearn.naive_bayes import BernoulliNB
model = BernoulliNB()
```

Train the model on the data.

```
model.fit(xTrain.values, yTrain)
```

```
BernoulliNB()
```

Make predictions for validation set.

```
yPrediction = model.predict(xValidation.values)
```

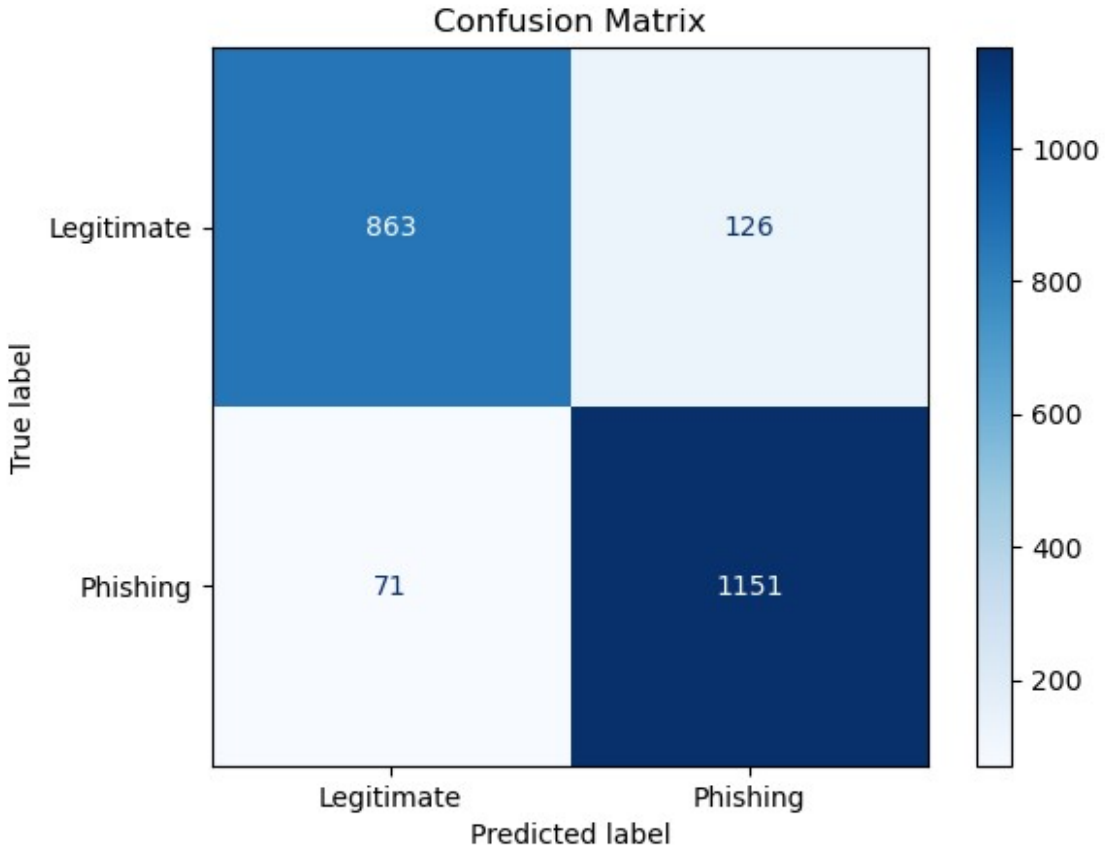
Show results of the classification for validation set.

```
from sklearn import metrics
print(metrics.classification_report(yPrediction, yValidation))
```

```
              precision    recall  f1-score   support

           0       0.87      0.92      0.90       934
           2       0.94      0.90      0.92      1277

    accuracy                           0.91      2211
   macro avg       0.91      0.91      0.91      2211
weighted avg       0.91      0.91      0.91      2211
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
target_names = ['Legitimate', 'Phishing']

matrix = confusion_matrix(yValidation, yPrediction)
display_matrix = ConfusionMatrixDisplay(confusion_matrix=matrix,
display_labels=target_names)
display_matrix.plot(cmap=plt.cm.Blues)
```

```
plt.title('Confusion Matrix')
plt.show()
```



Hyperparameter tuning using GridSearchCV.

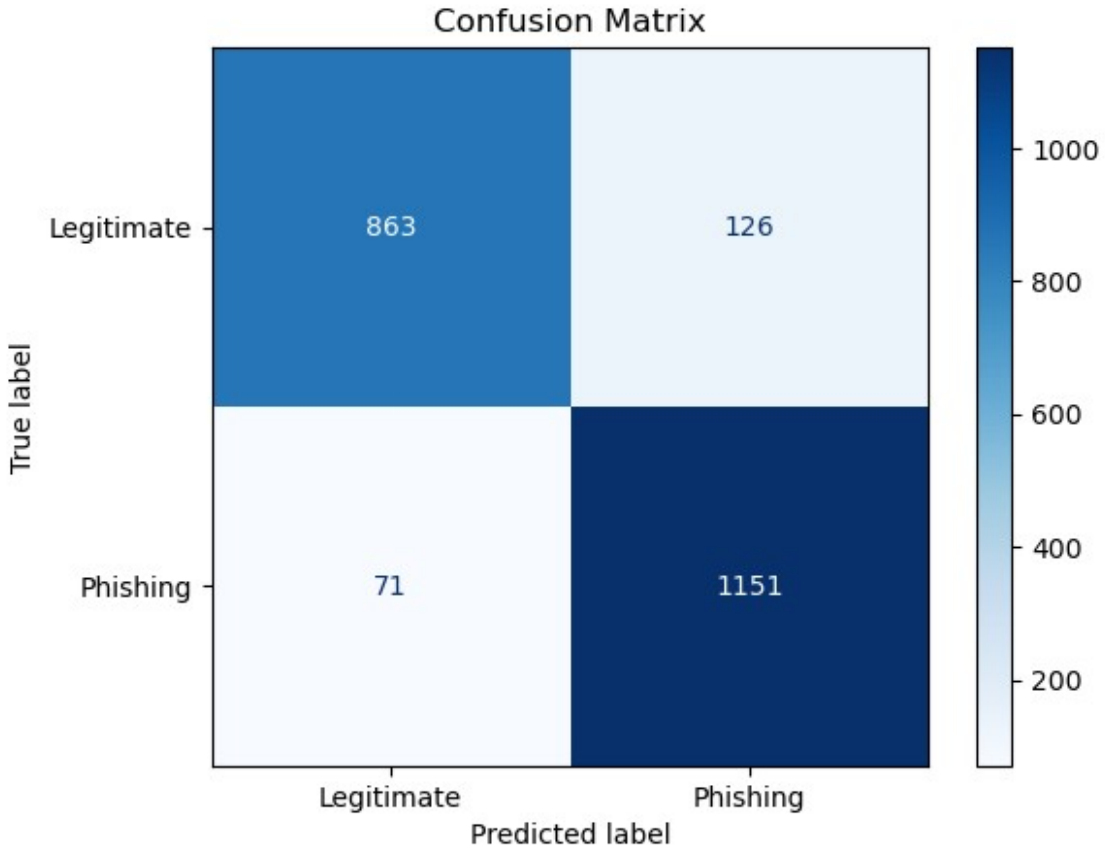
```
from sklearn.model_selection import GridSearchCV
param_grid = {"alpha": [0.1, 1.0, 10, 100]}
grid_search = GridSearchCV(BernoulliNB(), param_grid, verbose=2)
grid_search.fit(xTrain.values, yTrain)

Fitting 5 folds for each of 4 candidates, totalling 20 fits
[CV] END .....................................alpha=0.1; total
time=   0.0s
[CV] END .....................................alpha=0.1; total
time=   0.0s
[CV] END .....................................alpha=0.1; total
time=   0.0s
[CV] END .....................................alpha=0.1; total
time=   0.0s
[CV] END .....................................alpha=0.1; total
time=   0.0s
[CV] END .....................................alpha=1.0; total
time=   0.0s
```

```
[CV] END .....................................alpha=1.0; total
time=   0.0s
[CV] END .....................................alpha=1.0; total
time=   0.0s
[CV] END .....................................alpha=1.0; total
time=   0.0s
[CV] END .....................................alpha=1.0; total
time=   0.0s
[CV] END .....................................alpha=10; total
time=   0.0s
[CV] END .....................................alpha=10; total
time=   0.0s
[CV] END .....................................alpha=10; total
time=   0.0s
[CV] END .....................................alpha=10; total
time=   0.0s
[CV] END .....................................alpha=10; total
time=   0.0s
[CV] END ....................................alpha=100; total
time=   0.0s
[CV] END ....................................alpha=100; total
time=   0.0s
[CV] END ....................................alpha=100; total
time=   0.0s
[CV] END ....................................alpha=100; total
time=   0.0s
[CV] END ....................................alpha=100; total
time=   0.0s

GridSearchCV(estimator=BernoulliNB(), param_grid={'alpha': [0.1, 1.0,
10, 100]},
             verbose=2)
```

Show the results of the hyperparameter tuning.

```
print(grid_search.best_params_)
print(grid_search.best_score_)

yPrediction = grid_search.predict(xValidation.values)
print(metrics.classification_report(yPrediction, yValidation))
matrix = confusion_matrix(yValidation, yPrediction)
display_matrix = ConfusionMatrixDisplay(confusion_matrix=matrix,
display_labels=target_names)
display_matrix.plot(cmap=plt.cm.Blues)

plt.title('Confusion Matrix')
plt.show()

{'alpha': 0.1}
0.9140686359756354
```

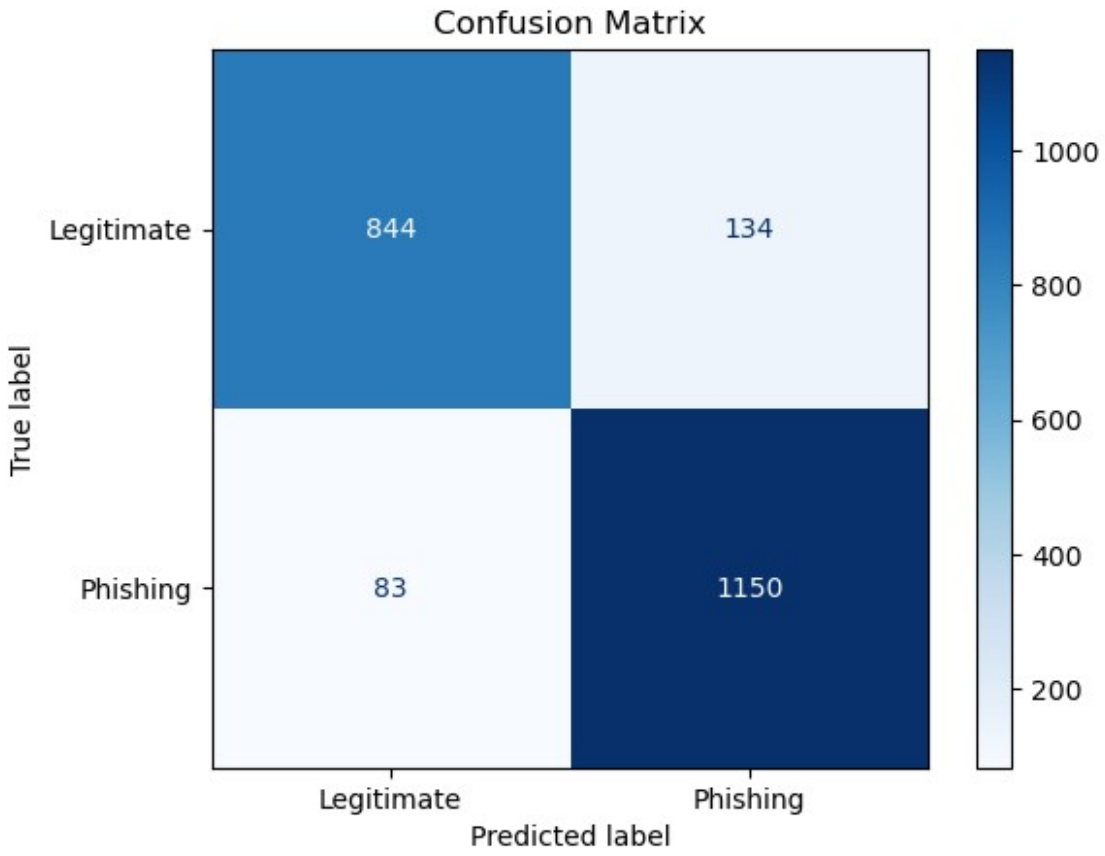
```
              precision    recall  f1-score   support

           0       0.87      0.92      0.90       934
           2       0.94      0.90      0.92      1277

    accuracy                           0.91      2211
   macro avg       0.91      0.91      0.91      2211
weighted avg       0.91      0.91      0.91      2211
```



Finally testing using the Test set.

```
yPrediction = grid_search.predict(xTest.values)
print(metrics.classification_report(yPrediction, yTest))
matrix = confusion_matrix(yTest, yPrediction)
display_matrix = ConfusionMatrixDisplay(confusion_matrix=matrix,
display_labels=target_names)
display_matrix.plot(cmap=plt.cm.Blues)

plt.title('Confusion Matrix')
plt.show()
```

```
                precision    recall  f1-score   support

           0       0.86      0.91      0.89       927
           2       0.93      0.90      0.91      1284

    accuracy                           0.90      2211
   macro avg       0.90      0.90      0.90      2211
weighted avg       0.90      0.90      0.90      2211
```



Save Model

```
import pickle
with open(f'models/bernoulli_nb.pkl', 'wb') as f:
    pickle.dump(grid_search, f)
```

Use Model

Import the model and use it.

```
with open(f'models/bernoulli_nb.pkl', 'rb') as f:
    loaded_model = pickle.load(f)
```

```
def take_input_and_convert_for_model(question):
    while(True):
        answer = input(question + ": ").upper()
        if answer == "Y":
            return 0
        elif answer == "N":
            return 2
        elif answer == "M":
            return 1
        print("Could not get answer, try again.")

print("Answer the following questions with Y for yes, N for no, and
for some questions M for the middle answer.")
data_list = [
    take_input_and_convert_for_model(x) for x in [
        "Does the URL have an IPv4 or IPv6 address? Example:
http://125.98.3.123/fake.html (Y, N)",
        "Is the URL shorter than 54 characters, in between 54 and 75
or longer? (Y, M, N)",
        "Is it a shortened URL? Example: bit.ly/19DXSk4 (Y, N)",
        "Does the URL have the @ symbol? (Y, N)",
        "Is a double foreward slash // present in the URL? Example:
http://www.legitimate.com//http://www.phishing.com (Y, N)",
        "Is there a dash - in the URL? Example: http://www.Confirme-
paypal.com/ (Y, N)",
        "Excluding the www. are there 1 dots, 2 dots, or more in the
URL? (Y, M, N)",
        "Is the website using https and the issuer is trusted and the
certificate is over 1 year old, or is it using https but issuer is not
trusted, or not using https? (Y, M, N)",
        "Does the domain expire is less that a year? (Y, N)",
        "Is the favicon loaded from a different domain? (Y, N)",
        "Are common ports in their preffered states? \n21 closed, \n22
closed, \n23 closed, \n80 open, \n443 open, \n445 closed, \n1433
closed, \n1521 closed, \n3306 closed, \n3389 closed \n(Y, N)",
        "Does https show up in the URL? Example: http://https-www-
paypal-it-webapps-mpp-home.soft-hair.com/ (Y, N)",
        "What percentage of resources are loaded from another URL,
less than 22, between 22 and 61 or more than 61? (Y, M, N)",
        "What percentage of <a> tags are pointing to another URL, less
than 22, between 22 and 61 or more than 61? (Y, M, N)",
        "What percentage of links in <meta>, <script> and <link> tags
are pointing to another URL, less than 22, between 22 and 61 or more
than 61? (Y, M, N)",
        "Is the Server Form Handler about:blank or empty, or it refers
to a different domain, or something else? (Y, M, N)",
        "Is mail() or mailto present on the page? (Y, N)",
        "Is the identity in WHOIS database part of the URL? (Y, N)",
        "Number of times redirected? Less than 1, between 2 and 4 or
more? (Y, M, N)",
```

```
        "Does onMouseOver change the URL? (Y, N)",
        "Is right click disabled? (Y, N)",
        "Is a popup window used? (Y, N)",
        "Do iframes use frameBorder? (Y, N)",
        "In WHOIS database is the age of the domain over 6 months? (Y,
N)",
        "Is there no DNS record for the domain? (Y, N)",
        "What website rank is it in the Alexa database? Less than
100,000, more than 100,000, or not found? (Y, M, N)",
        "Is the PageRank of the website less than 0.2? (Y, N)",
        "Is the website indexed by Google? (Y, N)",
        "What is the number of links pointing to this website? 0,
between 0 and 2, or more? (Y, M, N)",
        "Is the IP or domain in the PHishTank or StopBadware top 50
list? (Y, N)"
    ]
]
```

Answer the following questions with Y for yes, N for no, and for some questions M for the middle answer.

```
import numpy as np
prediction = loaded_model.predict(np.array(data_list).reshape(1, -1))
if prediction == 0:
    print("Website prediction is legitimate!")
elif prediction == 2:
    print("Website prediction is phishing!")
```