In [1]:

```
import pandas as pd
import numpy as np
import random

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

In [2]:

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving urldata.csv to urldata.csv
```

In [3]:

```
urls_data = pd.read_csv("urldata.csv", index_col=0)
urls_data.head()
```

Out[3]:

	url	label	result
0	https://www.google.com	benign	0
1	https://www.youtube.com	benign	0
2	https://www.facebook.com	benign	0
3	https://www.baidu.com	benign	0
4	https://www.wikipedia.org	benign	0

In [4]:

```
def makeTokens(f):
    tkns_BySlash = str(f.encode('utf-8')).split('/')    # make tokens after splitting by slash
    total_Tokens = []
    for i in tkns_BySlash:
        tokens = str(i).split('-')  # make tokens after splitting by dash
        tkns_ByDot = []
        for j in range(0,len(tokens)):
            temp_Tokens = str(tokens[j]).split('.') # make tokens after splitting by dot
            tkns_ByDot = tkns_ByDot + temp_Tokens
        total_Tokens = total_Tokens + tokens + tkns_ByDot
    total_Tokens = list(set(total_Tokens))  #remove redundant tokens
    if 'com' in total_Tokens:
        total_Tokens.remove('com')  #removing .com since it occurs a lot of times and it should not be included in our features
    return total_Tokens
```

In [5]:

```
y = urls_data["result"]
```

In [6]:

```
y.unique()
```

Out[6]:

```
array([0, 1])
```

In [7]:

```
url_list = urls_data["url"]
```

In [8]:

```
vectorizer = TfidfVectorizer(tokenizer=makeTokens)
```

In [9]:

```
X = vectorizer.fit_transform(url_list)
```

In [10]:

```
X.shape
```

Out[10]:

```
(450176, 780471)
```

In [12]:

```
pip install catboost
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) https://us-python.pkg.dev/colab-wheels/public/s
imple/ (https://us-python.pkg.dev/colab-wheels/public/simple/)
Collecting catboost
  Downloading catboost-1.1.1-cp38-none-manylinux1_x86_64.whl (76.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 76.6/76.6 MB 11.2 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.8/dist-packages (from catboost) (1.21.6)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.8/dist-packages (from catboost) (3.2.2)
Requirement already satisfied: graphviz in /usr/local/lib/python3.8/dist-packages (from catboost) (0.10.1)
Requirement already satisfied: scipy in /usr/local/lib/python3.8/dist-packages (from catboost) (1.7.3)
Requirement already satisfied: plotly in /usr/local/lib/python3.8/dist-packages (from catboost) (5.5.0)
Requirement already satisfied: six in /usr/local/lib/python3.8/dist-packages (from catboost) (1.15.0)
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.8/dist-packages (from catboost) (1.3.5)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-packages (from pandas>=0.24.0->catboost)
(2022.7.1)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.8/dist-packages (from pandas>=0.24.0->c
atboost) (2.8.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.8/dist-packages (from
matplotlib->catboost) (3.0.9)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib->catboost)
(1.4.4)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dist-packages (from matplotlib->catboost) (0.1
1.0)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.8/dist-packages (from plotly->catboost) (8.1.
0)
Installing collected packages: catboost
Successfully installed catboost-1.1.1
```

In [13]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, BaggingClassifier
import xgboost as xgb
from catboost import CatBoostClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
```

In [14]:

```
from sklearn.metrics import accuracy_score, confusion_matrix, recall_score, precision_score, f1_score
```

In [15]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [ ]:

```
logit = LogisticRegression()
logit.fit(X_train, y_train)
```

Out[18]:

```
LogisticRegression()
```

In [ ]:

```
print("Accuracy ",logit.score(X_test, y_test))
```

```
Accuracy  0.9948020791683326
```

In [ ]:

```
pred2 = logit.predict(X_test)
print(confusion_matrix(y_test, pred2))
print(accuracy_score(y_test, pred2), recall_score(y_test, pred2), precision_score(y_test, pred2),  f1_score(y_test, pred2))
```

```
[[68919     2]
 [  466 20649]]
0.9948020791683326 0.97793038124556 0.9999031523897148 0.9887947134032467
```

In [ ]:

```
pred2 = logit.predict(X_train)
print(confusion_matrix(y_train, pred2))
print(accuracy_score(y_train, pred2), recall_score(y_train, pred2), precision_score(y_train, pred2),  f1_score(y_train, pred2))
```

```
[[276813      4]
 [  1409  81914]]
0.996076525795524 0.9830899031479904 0.999951170682878 0.9914488534927771
```

In [ ]:

```
model2 = DecisionTreeClassifier()
model2.fit(X_train, y_train)

pred2 = model2.predict(X_test)
print(confusion_matrix(y_test, pred2))
print(accuracy_score(y_test, pred2), recall_score(y_test, pred2), precision_score(y_test, pred2),  f1_score(y_test, pred2))
```

```
[[68864    57]
 [  102 21013]]
0.9982340397174464 0.9951693109164101 0.9972947318462269 0.9962308877563114
```

In [ ]:

```
pr2 = model2.predict(X_train)
print(confusion_matrix(y_train, pr2))
print(accuracy_score(y_train, pr2), recall_score(y_train, pr2), precision_score(y_train, pr2),  f1_score(y_train, pr2))
```

```
[[276817      0]
 [     0  83323]]
1.0 1.0 1.0 1.0
```

In [ ]:

```
model2 = DecisionTreeClassifier()
bag_clf = BaggingClassifier(base_estimator=model2, n_estimators=100)
bag_clf.fit(X_train, y_train)

pred_bag = bag_clf.predict(X_test)
print(confusion_matrix(y_test, pred_bag))
print(accuracy_score(y_test, pred_bag), recall_score(y_test, pred_bag), precision_score(y_test, pred_bag),  f1_score(y_test, pred
```

In [ ]:

```
pr2 = bag_clf.predict(X_train)
print(confusion_matrix(y_train, pr2))
print(accuracy_score(y_train, pr2), recall_score(y_train, pr2), precision_score(y_train, pr2),  f1_score(y_train, pr2))
```

In [16]:

```
from sklearn.model_selection import GridSearchCV
```

In [21]:

```
gscv_rf = RandomForestClassifier(n_estimators=100, n_jobs=-1)
# param_rf = {'n_estimators':[200, 400, 500, 600, 700, 800, 1000]}

# gscv_rf = GridSearchCV(rf, param_grid=param_rf, n_jobs=-1)
gscv_rf.fit(X_train, y_train)

pred_rf = gscv_rf.predict(X_test)
print(confusion_matrix(y_test, pred_rf))
print(accuracy_score(y_test, pred_rf), recall_score(y_test, pred_rf), precision_score(y_test, pred_rf),  f1_score(y_test, pred_rf
```

```
[[68889    32]
 [  119 20996]]
0.9983228930649962 0.9943641960691452 0.9984782195168347 0.9964169612984363
```

In [22]:

```
pr2 = gscv_rf.predict(X_train)
print(confusion_matrix(y_train, pr2))
print(accuracy_score(y_train, pr2), recall_score(y_train, pr2), precision_score(y_train, pr2),  f1_score(y_train, pr2))
```

```
[[276817      0]
 [     0  83323]]
1.0 1.0 1.0 1.0
```

In [17]:

```
gscv_rf = RandomForestClassifier(n_estimators=50, n_jobs=-1)
# param_rf = {'n_estimators':[200, 400, 500, 600, 700, 800, 1000]}

# gscv_rf = GridSearchCV(rf, param_grid=param_rf, n_jobs=-1)
gscv_rf.fit(X_train, y_train)

pred_rf = gscv_rf.predict(X_test)
print(confusion_matrix(y_test, pred_rf))
print(accuracy_score(y_test, pred_rf), recall_score(y_test, pred_rf), precision_score(y_test, pred_rf),  f1_score(y_test, pred_rf
```

```
[[68887    34]
 [  119 20996]]
0.9983006797281088 0.9943641960691452 0.9983832620066572 0.9963696761181635
```

In [18]:

```
pr2 = gscv_rf.predict(X_train)
print(confusion_matrix(y_train, pr2))
print(accuracy_score(y_train, pr2), recall_score(y_train, pr2), precision_score(y_train, pr2),  f1_score(y_train, pr2))
```

```
[[276817      0]
 [     4  83319]]
0.9999888932081968 0.9999519940472619 1.0 0.9999759964474743
```

In [ ]:

```
#gscv_rf.best_params_
```

In [ ]:

```
gscv_gb = GradientBoostingClassifier(n_estimators=50, n_jobs=-1)
# param_gb = {'n_estimators':[700, 800, 900, 1000, 1100, 1200, 1300]}

# gscv_gb = GridSearchCV(gb, param_grid=param_gb, n_jobs=-1)
gscv_gb.fit(X_train, y_train)

pred_gb = gscv_gb.predict(X_test)
print(confusion_matrix(y_test, pred_gb))
print(accuracy_score(y_test, pred_gb), recall_score(y_test, pred_gb), precision_score(y_test, pred_gb),  f1_score(y_test, pred_gb
```

In [ ]:

```
pr2 = gscv_gb.predict(X_train)
print(confusion_matrix(y_train, pr2))
print(accuracy_score(y_train, pr2), recall_score(y_train, pr2), precision_score(y_train, pr2),  f1_score(y_train, pr2))
```

In [ ]:

```
#gscv_gb.best_params_
```

In [ ]:

```
gscv_ab = AdaBoostClassifier(n_estimators=50, n_jobs=-1)
# param_ab = {'n_estimators':[700, 800, 900, 1000, 1100, 1200, 1300]}

# gscv_ab = GridSearchCV(ab, param_grid=param_ab, n_jobs=-1)
gscv_ab.fit(X_train, y_train)

pred_ab = gscv_ab.predict(X_test)
print(confusion_matrix(y_test, pred_ab))
print(accuracy_score(y_test, pred_ab), recall_score(y_test, pred_ab), precision_score(y_test, pred_ab),  f1_score(y_test, pred_ab
```

In [ ]:

```
pr2 = gscv_ab.predict(X_train)
print(confusion_matrix(y_train, pr2))
print(accuracy_score(y_train, pr2), recall_score(y_train, pr2), precision_score(y_train, pr2),  f1_score(y_train, pr2))
```

In [ ]:

```
#gscv_ab.best_params_
```

In [ ]:

```
gscv_xb = xgb.XGBClassifier(n_estimators=500)
# param_xb = {'n_estimators':[700, 800, 900, 1000, 1100, 1200, 1300]}

# gscv_xb = GridSearchCV(xb, param_grid=param_xb, n_jobs=-1)
gscv_xb.fit(X_train, y_train)

pred_xb = gscv_xb.predict(X_test)
print(confusion_matrix(y_test, pred_xb))
print(accuracy_score(y_test, pred_xb), recall_score(y_test, pred_xb), precision_score(y_test, pred_xb),  f1_score(y_test, pred_xb
```

```
D:\AnacondaNavigator\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in XGBClassifier
is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_lab
el_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.
e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[15:13:29] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoo
st 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
[[68875    46]
 [  134 20981]]
0.998000799680128 0.9936538006156761 0.9978123365197128 0.9957287266859666
```

In [ ]:

```
pr2 = gscv_xb.predict(X_train)
print(confusion_matrix(y_train, pr2))
print(accuracy_score(y_train, pr2), recall_score(y_train, pr2), precision_score(y_train, pr2),  f1_score(y_train, pr2))
```

```
[[276710    107]
 [   355  82968]]
0.9987171655467318 0.9957394716944901 0.9987120072223894 0.9972235243212058
```

In [ ]:

```
#gscv_xb.best_params_
```

In [ ]:

```
cb = CatBoostClassifier(n_estimators=500)
cb.fit(X_train, y_train)

pred_cb = cb.predict(X_test)
print(confusion_matrix(y_test, pred_cb))
print(accuracy_score(y_test, pred_cb), recall_score(y_test, pred_cb), precision_score(y_test, pred_cb),  f1_score(y_test, pred_cb
```

```
Learning rate set to 0.24021
0:      learn: 0.1891841        total: 4.92s    remaining: 40m 54s
1:      learn: 0.0704305        total: 8.63s    remaining: 35m 49s
2:      learn: 0.0357784        total: 11.8s    remaining: 32m 40s
3:      learn: 0.0234963        total: 15.3s    remaining: 31m 40s
4:      learn: 0.0185126        total: 18.8s    remaining: 30m 57s
5:      learn: 0.0163698        total: 22.1s    remaining: 30m 19s
6:      learn: 0.0158579        total: 25.3s    remaining: 29m 43s
7:      learn: 0.0155386        total: 28.5s    remaining: 29m 9s
8:      learn: 0.0151340        total: 31.6s    remaining: 28m 46s
9:      learn: 0.0149476        total: 34.9s    remaining: 28m 28s
10:     learn: 0.0148453        total: 38.2s    remaining: 28m 17s
11:     learn: 0.0139928        total: 41.6s    remaining: 28m 10s
12:     learn: 0.0139131        total: 44.7s    remaining: 27m 55s
13:     learn: 0.0138438        total: 47.9s    remaining: 27m 41s
14:     learn: 0.0137769        total: 51s      remaining: 27m 28s
15:     learn: 0.0137220        total: 54.1s    remaining: 27m 16s
16:     learn: 0.0136613        total: 57.2s    remaining: 27m 5s
17:     learn: 0.0136081        total: 1m       remaining: 26m 56s
```

In [ ]:

```
pr2 = cb.predict(X_train)
print(confusion_matrix(y_train, pr2))
print(accuracy_score(y_train, pr2), recall_score(y_train, pr2), precision_score(y_train, pr2),  f1_score(y_train, pr2))
```

```
[[276582    235]
 [   465  82858]]
0.9980563114344422 0.9944193079941913 0.9971718435969336 0.9957936736852225
```

In [ ]:

```
import math
n = (int(math.sqrt(X_train.shape[0]))//2)*2+1
```

In [ ]:

```
knn = KNeighborsClassifier(n_neighbors=n, weights='distance')
knn.fit(X_train, y_train)

pred_knn = knn.predict(X_test)
print(confusion_matrix(y_test, pred_knn))
print(accuracy_score(y_test, pred_knn), recall_score(y_test, pred_knn), precision_score(y_test, pred_knn),  f1_score(y_test, pred
```

```
[[68078   843]
 [ 2477 18638]]
0.9631258607668044 0.882690030783803 0.9567270673990041 0.9182185436988866
```

In [ ]:

```
sv = SVC()
sv.fit(X_train, y_train)

pred_sv = sv.predict(X_test)
print(confusion_matrix(y_test, pred_sv))
print(accuracy_score(y_test, pred_sv), recall_score(y_test, pred_sv), precision_score(y_test, pred_sv),  f1_score(y_test, pred_sv
```

In [ ]:

```
from sklearn.naive_bayes import MultinomialNB, GaussianNB, BernoulliNB
```

In [ ]:

```
mnb = MultinomialNB()
mnb.fit(X_train, y_train)

pred_nb = mnb.predict(X_test)
print(confusion_matrix(y_test, pred_nb))
print(accuracy_score(y_test, pred_nb), recall_score(y_test, pred_nb), precision_score(y_test, pred_nb),  f1_score(y_test, pred_nb
```

```
[[68860    61]
 [  855 20260]]
0.98982629170554 0.9595074591522614 0.9969981792234635 0.9778936190752003
```

In [ ]:

```
bnb = BernoulliNB()
bnb.fit(X_train, y_train)

pred_nb = bnb.predict(X_test)
print(confusion_matrix(y_test, pred_nb))
print(accuracy_score(y_test, pred_nb), recall_score(y_test, pred_nb), precision_score(y_test, pred_nb),  f1_score(y_test, pred_nb
```

```
[[68914     7]
 [  327 20788]]
0.9962903727397929 0.9845133791143736 0.9996633806203414 0.9920305416368408
```

In [ ]:

```
pred_nb = bnb.predict(X_train)
print(confusion_matrix(y_train, pred_nb))
print(accuracy_score(y_train, pred_nb), recall_score(y_train, pred_nb), precision_score(y_train, pred_nb),  f1_score(y_train, pre
```

```
[[276805     12]
 [   989  82334]]
0.9972205253512523 0.988130528185495 0.9998542734316178 0.993957831579837
```

In [ ]:

```
lr_m = [99.48]
dt_m = [99.82]
xgb_m = [99.80]
knn_m = [96.31]
cb_m = [99.75]
mnb_m = [98.98]
bnb_m = [99.63]
```

In [ ]:

```
for i in [53, 72, 96, 21]:
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=i, stratify=y)

    model = LogisticRegression()
    model.fit(X_train, y_train)

    pred = model.predict(X_test)
    print("LR:")
    print(confusion_matrix(y_test, pred))
    print(accuracy_score(y_test, pred), recall_score(y_test, pred), precision_score(y_test, pred),  f1_score(y_test, pred))
    lr_m.append(accuracy_score(y_test, pred))


    model2 = DecisionTreeClassifier()
    model2.fit(X_train, y_train)

    pred2 = model2.predict(X_test)
    print("DTC:")
    print(confusion_matrix(y_test, pred2))
    print(accuracy_score(y_test, pred2), recall_score(y_test, pred2), precision_score(y_test, pred2),  f1_score(y_test, pred2))
    dt_m.append(accuracy_score(y_test, pred2))


    xb = xgb.XGBClassifier(n_estimators=700)
    xb.fit(X_train, y_train)

    pred_xb = xb.predict(X_test)
    print("XGB:")
    print(confusion_matrix(y_test, pred_xb))
    print(accuracy_score(y_test, pred_xb), recall_score(y_test, pred_xb), precision_score(y_test, pred_xb),  f1_score(y_test, pre
    xgb_m.append(accuracy_score(y_test, pred_xb))


    n = (int(math.sqrt(X_train.shape[0]))//2)*2+1

    knn = KNeighborsClassifier(n_neighbors=n, weights='distance')
    knn.fit(X_train, y_train)

    pred_knn = knn.predict(X_test)
    print("KNN:")
    print(confusion_matrix(y_test, pred_knn))
    print(accuracy_score(y_test, pred_knn), recall_score(y_test, pred_knn), precision_score(y_test, pred_knn),  f1_score(y_test,
    knn_m.append(accuracy_score(y_test, pred_knn))


    cb = CatBoostClassifier(n_estimators=500)
    cb.fit(X_train, y_train)

    pred_cb = cb.predict(X_test)
    print("CB:")
    print(confusion_matrix(y_test, pred_cb))
    print(accuracy_score(y_test, pred_cb), recall_score(y_test, pred_cb), precision_score(y_test, pred_cb),  f1_score(y_test, pre
    cb_m.append(accuracy_score(y_test, pred_cb))


    mnb = MultinomialNB()
    mnb.fit(X_train, y_train)

    pred_nb = mnb.predict(X_test)
    print("MNB:")
    print(confusion_matrix(y_test, pred_nb))
    print(accuracy_score(y_test, pred_nb), recall_score(y_test, pred_nb), precision_score(y_test, pred_nb),  f1_score(y_test, pre
    mnb_m.append(accuracy_score(y_test, pred_nb))


    bnb = BernoulliNB()
    bnb.fit(X_train, y_train)

    pred_nb = bnb.predict(X_test)
    print("BNB:")
    print(confusion_matrix(y_test, pred_nb))
    print(accuracy_score(y_test, pred_nb), recall_score(y_test, pred_nb), precision_score(y_test, pred_nb),  f1_score(y_test, pre
    bnb_m.append(accuracy_score(y_test, pred_nb))
```

```
D:\AnacondaNavigator\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to co
nverge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessin
g.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/m
odules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
LR:
[[69147     1]
 [  435 20453]]
0.9951574925585321 0.9791746457296056 0.9999511098073727 0.9894538241981521
DTC:
[[69082    66]
 [  101 20787]]
```

In [ ]:

```
for model in [lr_m, dt_m, xgb_m, cb_m, knn_m, mnb_m, bnb_m]:
    print(np.mean(model), np.std(model))
```

```
99.478 0.019390719429666442
99.818 0.007483314773545635
99.792 0.007483314773545634
99.76599999999999 0.010198039027184655
96.28 0.060991802727905275
98.95 0.029664793948382496
99.636 0.012000000000002349
```

In [ ]:

```

```