AI-spam\_classifier

**Phase 4: Development Part 2**

**Model building**

**from** sklearn.feature\_extraction.text **import** CountVectorizer,TfidfVectorizer

cv **=** CountVectorizer()

tfidf **=** TfidfVectorizer(max\_features**=**3000)

X **=** tfidf**.**fit\_transform(df['transformed\_text'])**.**toarray()

X**.**shape

y **=** df['target']**.**values

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

X\_train,X\_test,y\_train,y\_test **=** train\_test\_split(X,y,test\_size**=**0.2,random\_state**=**2)

**from** sklearn.naive\_bayes **import** GaussianNB,MultinomialNB,BernoulliNB

**from** sklearn.metrics **import** accuracy\_score,confusion\_matrix,precision\_score

gnb **=** GaussianNB()

mnb **=** MultinomialNB()

bnb **=** BernoulliNB()

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

y\_pred1 **=** gnb**.**predict(X\_test)

print(accuracy\_score(y\_test,y\_pred1))

print(confusion\_matrix(y\_test,y\_pred1))

print(precision\_score(y\_test,y\_pred1))

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

y\_pred2 **=** mnb**.**predict(X\_test)

print(accuracy\_score(y\_test,y\_pred2))

print(confusion\_matrix(y\_test,y\_pred2))

print(precision\_score(y\_test,y\_pred2))

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

y\_pred3 **=** bnb**.**predict(X\_test)

print(accuracy\_score(y\_test,y\_pred3))

print(confusion\_matrix(y\_test,y\_pred3))

print(precision\_score(y\_test,y\_pred3))

**Model Training**

**from** sklearn.linear\_model **import** LogisticRegression

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

**from** sklearn.naive\_bayes **import** MultinomialNB

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

**from** sklearn.neighbors **import** KNeighborsClassifier

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

**from** sklearn.ensemble **import** AdaBoostClassifier

**from** sklearn.ensemble **import** BaggingClassifier

**from** sklearn.ensemble **import** ExtraTreesClassifier

**from** sklearn.ensemble **import** GradientBoostingClassifier

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

svc **=** SVC(kernel**=**'sigmoid', gamma**=**1.0)

knc **=** KNeighborsClassifier()

mnb **=** MultinomialNB()

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

lrc **=** LogisticRegression(solver**=**'liblinear', penalty**=**'l1')

rfc **=** RandomForestClassifier(n\_estimators**=**50, random\_state**=**2)

abc **=** AdaBoostClassifier(n\_estimators**=**50, random\_state**=**2)

bc **=** BaggingClassifier(n\_estimators**=**50, random\_state**=**2)

etc **=** ExtraTreesClassifier(n\_estimators**=**50, random\_state**=**2)

gbdt **=** GradientBoostingClassifier(n\_estimators**=**50,random\_state**=**2)

xgb **=** XGBClassifier(n\_estimators**=**50,random\_state**=**2)

clfs **=** {

'SVC' : svc,

'KN' : knc,

'NB': mnb,

'DT': dtc,

'LR': lrc,

'RF': rfc,

'AdaBoost': abc,

'BgC': bc,

'ETC': etc,

'GBDT':gbdt,

'xgb':xgb

}

**def** train\_classifier(clf,X\_train,y\_train,X\_test,y\_test):

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

y\_pred **=** clf**.**predict(X\_test)

accuracy **=** accuracy\_score(y\_test,y\_pred)

precision **=** precision\_score(y\_test,y\_pred)

**return** accuracy,precision

train\_classifier(svc,X\_train,y\_train,X\_test,y\_test)

accuracy\_scores **=** []

precision\_scores **=** []

**for** name,clf **in** clfs**.**items():

current\_accuracy,current\_precision **=** train\_classifier(clf, X\_train,y\_train,X\_test,y\_test)

print("For ",name)

print("Accuracy - ",current\_accuracy)

print("Precision - ",current\_precision)

accuracy\_scores**.**append(current\_accuracy)

precision\_scores**.**append(current\_precision)

performance\_df **=** pd**.**DataFrame({'Algorithm':clfs**.**keys(),'Accuracy':accuracy\_scores,'Precision':precision\_scores})**.**sort\_values('Precision',ascending**=False**)

performance\_df

performance\_df1 **=** pd**.**melt(performance\_df, id\_vars **=** "Algorithm")

performance\_df1

sns**.**catplot(x **=** 'Algorithm', y**=**'value',

hue **=** 'variable',data**=**performance\_df1, kind**=**'bar',height**=**5)

plt**.**ylim(0.5,1.0)

plt**.**xticks(rotation**=**'vertical')

plt**.**show()

temp\_df **=** pd**.**DataFrame({'Algorithm':clfs**.**keys(),'Accuracy\_max\_ft\_3000':accuracy\_scores,'Precision\_max\_ft\_3000':precision\_scores})**.**sort\_values('Precision\_max\_ft\_3000',ascending**=False**)

temp\_df **=** pd**.**DataFrame({'Algorithm':clfs**.**keys(),'Accuracy\_scaling':accuracy\_scores,'Precision\_scaling':precision\_scores})**.**sort\_values('Precision\_scaling',ascending**=False**)

new\_df **=** performance\_df**.**merge(temp\_df,on**=**'Algorithm')

new\_df\_scaled **=** new\_df**.**merge(temp\_df,on**=**'Algorithm')

temp\_df **=** pd**.**DataFrame({'Algorithm':clfs**.**keys(),'Accuracy\_num\_chars':accuracy\_scores,'Precision\_num\_chars':precision\_scores})**.**sort\_values('Precision\_num\_chars',ascending**=False**)

new\_df\_scaled**.**merge(temp\_df,on**=**'Algorithm')

svc **=** SVC(kernel**=**'sigmoid', gamma**=**1.0,probability**=True**)

mnb **=** MultinomialNB()

etc **=** ExtraTreesClassifier(n\_estimators**=**50, random\_state**=**2)

**from** sklearn.ensemble **import** VotingClassifier

voting **=** VotingClassifier(estimators**=**[('svm', svc), ('nb', mnb), ('et', etc)],voting**=**'soft')

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

y\_pred **=** voting**.**predict(X\_test)

print("Accuracy",accuracy\_score(y\_test,y\_pred))

print("Precision",precision\_score(y\_test,y\_pred))

estimators**=**[('svm', svc), ('nb', mnb), ('et', etc)]

final\_estimator**=**RandomForestClassifier()

**from** sklearn.ensemble **import** StackingClassifier

clf **=** StackingClassifier(estimators**=**estimators, final\_estimator**=**final\_estimator)

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

y\_pred **=** clf**.**predict(X\_test)

print("Accuracy",accuracy\_score(y\_test,y\_pred))

print("Precision",precision\_score(y\_test,y\_pred))

**import** pickle

pickle**.**dump(tfidf,open('vectorizer.pkl','wb'))

pickle**.**dump(mnb,open('model.pkl','wb'))

**Evaluation**

from flask import Flask, render\_template,request

import pickle

import numpy as np

app=Flask(\_\_name\_\_)

word\_list=pickle.load(open('mystrings.pkl','rb'))

clf=pickle.load(open('model.pkl','rb'))

@app.route('/')

def home():

return render\_template('index.html')

@app.route('/predict', methods=['POST'])

def predict():

email=request.form.get('email')

input\_mail = []

for i in word\_list:

input\_mail.append(email.count(i[0]))

x=clf.predict(np.array(input\_mail).reshape(1, 3000))

x=x[0]

return render\_template('index.html', label=str(x))

if \_\_name\_\_=="\_\_main\_\_":

app.run(debug=True)

**Display**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Email Spam Classifier</title>

</head>

<body>

<h2>Enter your Email</h2>

{% if label %}

{% if label==0 %}

<p style="color:green">Not Spam</p>

{% else %}

<p style="color:red">Spam</p>

{% endif %}

{% endif %}

<form method="POST" action="/predict">

<textarea name="email"></textarea><br>

<input type="submit" value="Predict"/>

</form>

</body>

</html>