jupyter Spam Mail Prediction Last Checkpoint: 5 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.tree import DT
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import PL,score
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [2]:
```
df = pd.read_csv('data/Machine_Learning_Projects-mailchmp/spam predictionmail_data - mail_data.csv')
```

execution time: Kernel 0:00:00.001 CE

Out[2]:

	Category	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will ì_ b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So any other s...
5570	ham	The guy did some bidding but said he is ...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns

In [3]:
```
df.describe()
```

execution time: Kernel 0:00:00.002 CE

Out[3]:

	Category	Message
count	5572	5572
unique	2	5157
top	ham	Sorry, I'll call later
freq	4825	30

In [4]:
```
df.info()
```

execution time: Kernel 0:00:00.002 CE

```
<class'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Category 5572 non-null   object
 1   Message  5572 non-null   object
dtypes: object(2)
memory usage: 87.1+ KB
```

In [5]:
```
df.isnull()
```

execution time: Kernel 0:00:00.002 CE

Out[5]:

	Category	Message
0	False	False
1	False	False
2	False	False
3	False	False
4	False	False
...	...	...
5567	False	False
5568	False	False
5569	False	False
5570	False	False
5571	False	False

5572 rows × 2 columns

In [6]:
```
sns.countplot('Category', data=df)
```

execution time: Kernel 0:00:00.002 CE

Out[6]:
```
<AxesSubplot:xlabel='Category', ylabel='count'>
```

In [7]:
```
df.Category.value_counts()
```

execution time: Kernel 0:00:00.002 CE

Out[7]:
```
ham    4825
spam    747
Name: Category, dtype: int64
```

In [8]:
```
df.replace(to_int, inplace=True)
df.fillna(0,0, inplace=True)
```

execution time: Kernel 0:00:00.002 CE

In [9]:
```
df.shape_dimension(2)
df.Category.value_counts().get_dataframe(), 'df needs to be a pd.DataFrame'
df.drop(columns='ham')
df.Category.value_counts().get_dataframe(, ng.inf).apply()
    .sort_values(ascending=False)
```

execution time: Kernel 0:00:00.003 CE

In [10]:
```
np.isnan(df.spam) #not gets false
np.isfinite(df.spam) #not gets true

np.any(df.isnan(df))

np.all(np.isfinite(df))
np.where(np.isfinite(df))
```

Out[10]:
```
False
```

In [11]:
```
df.replace(ham, 1)
for column in df.columns:
    if df[column].dtype == 'object':
        df[column] = LE.fit_transform(df[column])
```

execution time: Kernel 0:00:00.003 CE

In [12]:
```
[ham] = LabelEncoder()
df.Category = LE.fit_transform(df.cat)
df[column] = LabelEncoder().fit_transform(df[column])
```

execution time: Kernel 0:00:00.003 CE

In [13]:
```
from sklearn.model_selection import train_test_split

X = df.drop('category', axis=1)
Y = df.Category

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

execution time: Kernel 0:00:00.002 CE

In [14]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

def print_score(clf, X_train, y_train, X_test, y_test, train=True):
    if train:
        pred = clf.predict(X_train)
        clf_report = pd.DataFrame(classification_report(y_train, pred, output_dict=True))
        print("Train Result:\n================================================")
        print("Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
        print("_______________________________________________")
        print("CLASSIFICATION REPORT:\n{clf_report}")
        print("_______________________________________________")
        print("Confusion Matrix: \n {confusion_matrix(y_train, pred)}")

    elif train:
        pred = clf.predict(X_test)
        clf_report = pd.DataFrame(classification_report(y_test, pred, output_dict=True))
        print("Test Result:\n================================================")
        print("Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
        print("_______________________________________________")
        print("CLASSIFICATION REPORT:\n{clf_report}")
        print("_______________________________________________")
        print("Confusion Matrix: \n {confusion_matrix(y_test, pred)}")
```

execution time: Kernel 0:00:00.002 CE

In [15]:
```
tree_clf = DecisionTreeClassifier(random_state=100)
tree_clf.fit(X_train, y_train)

print_score(tree_clf, X_train, y_train, X_test, y_test, train=True)
print_score(tree_clf, X_train, y_train, X_test, y_test, train=False)
```

execution time: Kernel 0:00:00.012 CE

```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
CLASSIFICATION REPORT:
              0.0  1.0  accuracy  macro avg  weighted avg
precision   1.0  1.0       1.0        1.0           1.0
recall      1.0  1.0       1.0        1.0           1.0
f1-score    1.0  1.0       1.0        1.0           1.0
support   3859  613       1.0     4472.0        4472.0
_______________________________________________
Confusion Matrix:
 [[3859    0]
 [   0  613]]

Test Result:
================================================
Accuracy Score: 98.25%
_______________________________________________
CLASSIFICATION REPORT:
              0.0  1.0  accuracy  macro avg  weighted avg
precision   0.99  0.93  0.982522   0.961111      0.982522
recall      0.99  0.93  0.982522   0.961111      0.982522
f1-score    0.99  0.93  0.982522   0.961111      0.982522
support   1203.  150  0.982522  1353.000000   1353.000000
_______________________________________________
Confusion Matrix:
 [[1190   13]
 [  11  139]]
```

In [16]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

params = {
    'criterion':['gini', 'entropy'],
    'splitter':['best', 'random'],
    'max_depth':[1,2,3,4,5,6,7],
    'max_features':['auto', 'sqrt', 'log2'],
    'min_samples_leaf':[1,2,3,4,5,6,7,8,9,10],
    'min_samples_split':[2,3,4,5,6,7,8,9,10],
    }

tree_clf = DecisionTreeClassifier(random_state=42)
tree_cv = GridSearchCV(tree_clf, params, scoring='accuracy', n_jobs=-1, verbose=1, cv=3)
tree_cv.fit(X_train, y_train)
best_params = tree_cv.best_params_
print("Best paramters: {best_params}")

tree_clf = DecisionTreeClassifier(**best_params)
tree_clf.fit(X_train, y_train)
print_score(tree_clf, X_train, y_train, X_test, y_test, train=True)
print_score(tree_clf, X_train, y_train, X_test, y_test, train=False)
```

execution time: Kernel 0:00:00.003 CE

```
Fitting 3 folds for each of 140 candidates, totalling 420 Fits
Best paramters: {'criterion': 'entropy', 'max_depth': 7, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 3, 'splitter': 'best'
Train Result:
================================================
Accuracy Score: 98.82%
_______________________________________________
CLASSIFICATION REPORT:
              0.0       1.0  accuracy  macro avg  weighted avg
precision   0.99  0.990196  0.988162   0.994598      0.988162
recall      1.00  0.919739  0.988162   0.959870      0.988162
f1-score    0.99  0.953888  0.988162   0.976944      0.988162
support   3859  613.000000  0.988162  4472.000000   4472.000000
_______________________________________________
Confusion Matrix:
 [[3856    3]
 [  49  564]]

Test Result:
================================================
Accuracy Score: 98.45%
_______________________________________________
CLASSIFICATION REPORT:
              0.0       1.0  accuracy  macro avg  weighted avg
precision   0.99  0.990196  0.984455   0.994598      0.984455
recall      1.00  0.919739  0.984455   0.959870      0.984455
f1-score    0.99  0.953888  0.984455   0.976944      0.984455
support   1203.  150.000000  0.984455  1353.000000   1353.000000
_______________________________________________
Confusion Matrix:
 [[1195    3]
 [  12  138]]
```

In [17]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomSearchCV

n_estimators = [int(x) for x in np.linspace(start=200, stop=2000, num=10)]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(10, 110, num=11)]
min_samples_split = [2, 5, 10]
min_samples_split = [1, 2, 4]
bootstrap = [True, False]
bootstrap = [True, False]

random_grid = {'n_estimators': n_estimators, 'max_features': max_features,
               'max_depth': max_depth, 'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf, 'bootstrap': bootstrap}

rf = RandomForestClassifier(random_state=42)

rf_cv = RandomSearchCV(estimator=rf,param_grid=random_grid,n_iter=100,cv=3,
                       verbose=2, random_state=42, n_jobs=-1)

rf_cv.fit(X_train, y_train)
best_params = rf_cv.best_params_
print("Best paramters: {best_params}")

rf = RandomForestClassifier(**best_params)
rf.fit(X_train, y_train)
print_score(rf, X_train, y_train, X_test, y_test, train=True)
print_score(rf, X_train, y_train, X_test, y_test, train=False)
```

execution time: Kernel 0:00:00.003 CE

```
Fitting 3 folds for each of 100 candidates, totalling 300 Fits
Best paramters: {'bootstrap': False, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_features': 'auto', 'max_depth':
Train Result:
================================================
Accuracy Score: 99.40%
_______________________________________________
CLASSIFICATION REPORT:
              0.0       1.0  accuracy  macro avg  weighted avg
precision   0.99  0.994898  0.994407   0.994949      0.994407
recall      1.00  0.967427  0.994407   0.983713      0.994407
f1-score    0.99  0.980952  0.994407   0.990476      0.994407
support   3859  613.000000  0.994407  4472.000000   4472.000000
_______________________________________________
Confusion Matrix:
 [[3857    2]
 [  20  593]]

Test Result:
================================================
Accuracy Score: 98.45%
_______________________________________________
CLASSIFICATION REPORT:
              0.0       1.0  accuracy  macro avg  weighted avg
precision   0.99  0.990196  0.984455   0.994598      0.984455
recall      1.00  0.919739  0.984455   0.959870      0.984455
f1-score    0.99  0.953888  0.984455   0.976944      0.984455
support   1203.  150.000000  0.984455  1353.000000   1353.000000
_______________________________________________
Confusion Matrix:
 [[1195    3]
 [  12  138]]
```

In [18]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomSearchCV

n_estimators = [int(x) for x in np.linspace(start=200, stop=2000, num=10)]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(10, 110, num=11)]
min_samples_split = [2, 5, 10]
min_samples_split = [1, 2, 4]
bootstrap = [True, False]
bootstrap = [True, False]

random_grid = {'n_estimators': n_estimators, 'max_features': max_features,
               'max_depth': max_depth, 'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf, 'bootstrap': bootstrap}

rf = RandomForestClassifier(random_state=42)

rf_cv = RandomSearchCV(estimator=rf,param_grid=random_grid,n_iter=100,cv=3,
                       verbose=2, random_state=42, n_jobs=-1)

rf_cv.fit(X_train, y_train)
best_params = rf_cv.best_params_
print("Best paramters: {best_params}")

rf = RandomForestClassifier(**best_params)
rf.fit(X_train, y_train)
print_score(rf, X_train, y_train, X_test, y_test, train=True)
print_score(rf, X_train, y_train, X_test, y_test, train=False)
```

execution time: Kernel 0:00:00.003 CE

```
Fitting 3 folds for each of 100 candidates, totalling 300 Fits
Best paramters: {'bootstrap': True, 'max_depth': 90, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split
Train Result:
================================================
Accuracy Score: 99.40%
_______________________________________________
CLASSIFICATION REPORT:
              0.0       1.0  accuracy  macro avg  weighted avg
precision   0.99  0.994898  0.994407   0.994949      0.994407
recall      1.00  0.967427  0.994407   0.983713      0.994407
f1-score    0.99  0.980952  0.994407   0.990476      0.994407
support   3859  613.000000  0.994407  4472.000000   4472.000000
_______________________________________________
Confusion Matrix:
 [[3857    2]
 [  20  593]]

Test Result:
================================================
Accuracy Score: 98.52%
_______________________________________________
CLASSIFICATION REPORT:
              0.0       1.0  accuracy  macro avg  weighted avg
precision   0.99  0.990196  0.985227   0.994598      0.985227
recall      1.00  0.919739  0.985227   0.959870      0.985227
f1-score    0.99  0.954198  0.985227   0.976944      0.985227
support   1203.  150.000000  0.985227  1353.000000   1353.000000
_______________________________________________
Confusion Matrix:
 [[1195    3]
 [  11  139]]
```

In [19]:
```
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)
```

execution time: Kernel 0:00:00.002 CE

In [20]:
```
from sklearn.tree import DecisionTreeClassifier
tc = DecisionTreeClassifier(random_state=42)
tc.fit(X_train, y_train)
```

execution time: Kernel 0:00:00.003 CE

Out[20]:
```
DecisionTreeClassifier(random_state=42)
```

In [20]:
```
tc.score(X_train, y_train)*100
```

execution time: Kernel 0:00:00.002 CE

Out[20]:
```
100.0
```

In [21]:
```
y_pred = tc.predict(X_test)
accuracy_score(y_pred, y_test)*100
```

execution time: Kernel 0:00:00.002 CE

Out[21]:
```
95.37666666666666
```

In [22]:
```
tc = DecisionTreeClassifier(criterion='entropy')
tc.fit(X_train, y_train)
```

execution time: Kernel 0:00:00.002 CE

Out[22]:
```
DecisionTreeClassifier(criterion='entropy')
```

In [23]:
```
y_pred = tc.predict(X_test)
accuracy_score(y_pred, y_test)*100
```

execution time: Kernel 0:00:00.002 CE

Out[23]:
```
95.37666666666666
```

In [24]:
```
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)
```

execution time: Kernel 0:00:00.002 CE

In [25]:
```
from sklearn.tree import DecisionTreeClassifier
tc = DecisionTreeClassifier(random_state=42)
tc.fit(X_train, y_train)
```

execution time: Kernel 0:00:00.002 CE

Out[25]:
```
DecisionTreeClassifier(random_state=42)
```

In [26]:
```
tc.score(X_train, y_train)*100
```

execution time: Kernel 0:00:00.002 CE

Out[26]:
```
100.0
```

In [27]:
```
y_pred = tc.predict(X_test)
accuracy_score(y_pred, y_test)*100
```

execution time: Kernel 0:00:00.002 CE

Out[27]:
```
95.37666666666666
```

In [28]:
```
tc = DecisionTree(criterion='entropy')
tc.fit(X_train, y_train)
```

execution time: Kernel 0:00:00.002 CE

Out[28]:
```
DecisionTree(criterion='entropy')
```

In [29]:
```
y_pred = tc.predict(X_test)
accuracy_score(y_pred, y_test)*100
```

execution time: Kernel 0:00:00.002 CE

Out[29]:
```
95.37666666666666
```

In [30]:
```
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)
```

execution time: Kernel 0:00:00.002 CE

In [31]:
```
from sklearn.tree import DecisionTreeClassifier
tc = DecisionTreeClassifier(random_state=42)
tc.fit(X_train, y_train)
```

execution time: Kernel 0:00:00.002 CE

Out[31]:
```
DecisionTreeClassifier(random_state=42)
```

In [32]:
```
tc.score(X_train, y_train)*100
```

execution time: Kernel 0:00:00.002 CE

Out[32]:
```
100.0
```

In [33]:
```
y_pred = tc.predict(X_test)
accuracy_score(y_pred, y_test)*100
```

execution time: Kernel 0:00:00.002 CE

Out[33]:
```
95.37666666666666
```

In [34]:
```
tc = DecisionTreeClassifier(criterion='entropy')
tc.fit(X_train, y_train)
```

execution time: Kernel 0:00:00.002 CE

Out[34]:
```
DecisionTreeClassifier(criterion='entropy')
```

In [35]:
```
y_pred = tc.predict(X_test)
accuracy_score(y_pred, y_test)*100
```

execution time: Kernel 0:00:00.002 CE

Out[35]:
```
95.37666666666666
```

In [36]:
```
ac = neighbors.KNeighborsClassifier(n_neighbors=10)
ac.fit(X_train, y_train)
```

execution time: Kernel 0:00:00.002 CE

Out[36]:
```
KNeighborsClassifier(n_neighbors=10)
```

In [37]:
```
y_pred = ac.predict(X_test)
accuracy_score(y_pred, y_test)*100
```

execution time: Kernel 0:00:00.002 CE

```
D:\Programs\Jupyter\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction functions
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
D:\Programs\Jupyter\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction functions
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

Out[37]:
```
94.61267222222223
```

In [38]:
```
rf = RandomForestClassifier(n_estimators=1)
rf.fit(X_train, y_train)
```

execution time: Kernel 0:00:00.002 CE

Out[38]:
```
RandomForestClassifier(n_estimators=1)
```

In [39]:
```
y_pred = rf.predict(X_test)
accuracy_score(y_pred, y_test)*100
```

execution time: Kernel 0:00:00.002 CE

Out[39]:
```
93.43833333333333
```

In [40]:
```
score = cross_val_score(rf, X, y, cv = 10)
print(score.mean()*100)
```

execution time: Kernel 0:00:00.002 CE

```
93.16666666666667
```

In [41]:
```
cm= heatmap(confusion_matrix(rf.predict(X_test), y_test),annot=True)
PL.savefig('confusion_matrix_y_test.jpg')
```

execution time: Kernel 0:00:00.002 CE

Out[41]:
```
0.5166666666666666
```

In [42]:
```
df.loc[df['Category'] == 'spam', 'Category'] = 0
df.loc[df['Category'] == 'ham', 'Category'] = 1
```

execution time: Kernel 0:00:00.002 CE

In [43]:
```
# 'df' Message:
df.shape_dimension(2)
```

execution time: Kernel 0:00:00.002 CE

In [44]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y ,train_size = 0.3 , shuffle=True , random_state=1)
```

execution time: Kernel 0:00:00.002 CE

In [45]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
```

execution time: Kernel 0:00:00.002 CE

```
(1672,)
(3900,)
(1672,)
```

In [46]:
```
feature_extraction = TfidfVectorizer(lowercase=True)
```

In [47]:
```
X_train_features = feature_extraction.fit_transform(X_train)
X_test_features = feature_extraction.transform(X_test)

X_train = y_train.astype('int')
X_test = y_test.astype('int')
```

execution time: Kernel 0:00:00.002 CE

In [48]:
```
model = LogisticRegression()
model.fit(X_train_features, X_train)
```

execution time: Kernel 0:00:00.002 CE

Out[48]:
```
LogisticRegression()
```

In [49]:
```
model_training_prediction = model.predict(X_train_features)
training_accuracy = accuracy_score( X_train , model_training_prediction )
print('Accuracy on training data : ', training_accuracy )
```

execution time: Kernel 0:00:00.002 CE

```
Accuracy on training data :  0.9522727272727273
```

In [50]:
```
model_testing_prediction = model.predict(X_test_features)
test_accuracy = accuracy_score( X_test , model_testing_prediction )
print('Accuracy on test data : ', test_accuracy )
```

execution time: Kernel 0:00:00.002 CE

```
Accuracy on test data :  0.9522727272727273
```

In [51]:
```
input_mail = [] # enter searching for the right gives to thank you for this breather. I promise i wont take your help for g
input_data_features = feature_extraction.transform([input_mail])

prediction = model.predict(input_data_features)

if (prediction[0]==1):
    print("ham mail")
else:
    print("spam mail")
```

execution time: Kernel 0:00:00.002 CE

```
spam mail
```

In [ ]:
```
```