ex8-support-vector-machine

August 12, 2024

1 Import the dataset

[1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

[2]:
```
spam = pd.read_csv("datasets/spam.csv")
spam.head()
```

[2]:

	Label	EmailText
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…

2 Check the shape of the dataset

[3]:
```
spam.shape
```

[3]:
```
(5572, 2)
```

3 Check the columns present in the dataset

[4]:
```
spam.columns
```

[4]:
```
Index(['Label', 'EmailText'], dtype='object')
```

4 Check the descriptive statistics of the dataset

[5]:
```
spam.describe()
```

[5]:

	Label	EmailText
count	5572	5572

```
unique     2                    5169
top      ham  Sorry, I'll call later
freq    4825                      30
```

5 Check the info of the dataset

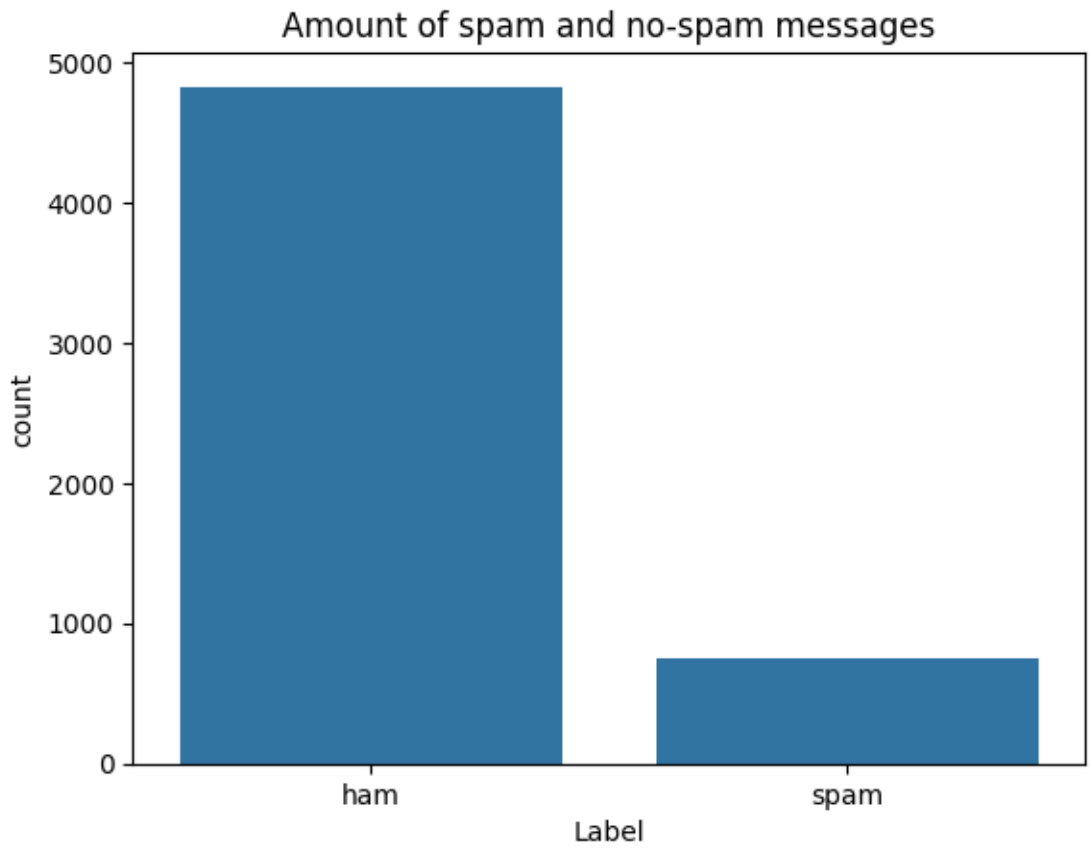
[6]:
```
spam.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Label      5572 non-null   object
 1   EmailText  5572 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB
```

[7]:
```
spam["Label"].value_counts()
```

[7]:
```
Label
ham     4825
spam     747
Name: count, dtype: int64
```

[8]:
```
sns.countplot(data=spam, x=spam["Label"]).set_title(
    "Amount of spam and no-spam messages"
)
plt.show()
```



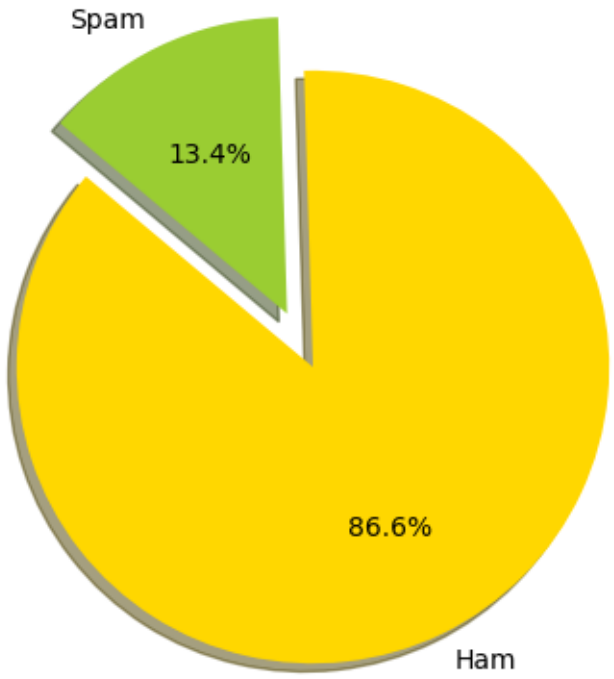
6 Plotting Pie-Chart

[9]:
```
count_Class = pd.value_counts(spam.Label, sort=True)
# Data to plot
labels = "Ham", "Spam"
sizes = [count_Class[0], count_Class[1]]
colors = ["gold", "yellowgreen"]  # 'lightcoral', 'lightskyblue'
explode = (0.1, 0.1)  # explode 1st slice

plt.pie(
    sizes,
    explode=explode,
    labels=labels,
    colors=colors,
    autopct="%1.1f%%",
    shadow=True,
    startangle=140,
)
plt.axis("equal")
```

```
plt.show()
```

```
/tmp/ipykernel_4254/3052618870.py:1: FutureWarning: pandas.value_counts is
deprecated and will be removed in a future version. Use
pd.Series(obj).value_counts() instead.
  count_Class = pd.value_counts(spam.Label, sort=True)
/tmp/ipykernel_4254/3052618870.py:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  sizes = [count_Class[0], count_Class[1]]
```



7 Extract the independent variables to create a dataframe X

[10]:
```
X = spam["EmailText"]
X.head()
```

[10]:
```
0    Go until jurong point, crazy.. Available only …
1                        Ok lar… Joking wif u oni…
2    Free entry in 2 a wkly comp to win FA Cup fina…
3    U dun say so early hor… U c already then say…
4    Nah I don't think he goes to usf, he lives aro…
```

```
Name: EmailText, dtype: object
```

8 Extract the dependent variables to create a dataframe y

[11]:
```
y = spam["Label"]
y.head()
```

[11]:
```
0     ham
1     ham
2    spam
3     ham
4     ham
Name: Label, dtype: object
```

9 Split X and y into train and test dataset with test_size = 0.20, random_state=0

[12]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, random_state=0
)
```

10 Check the shape of X and y of train dataset

[13]:
```
print(X_train.shape)
print(y_train.shape)
```

```
(4457,)
(4457,)
```

11 Check the shape of X and y of test dataset

[14]:
```
print(X_test.shape)
print(y_test.shape)
```

```
(1115,)
(1115,)
```

12 Applying various models of Machine Learning

[15]:
```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer()
trainCV = cv.fit_transform(X_train)
testCV = cv.transform(X_test)
```

[16]:
```
from sklearn.naive_bayes import MultinomialNB

naive_bayes = MultinomialNB()
naive_bayes.fit(trainCV, y_train)
pred_NB = naive_bayes.predict(testCV)
```

[17]:
```
from sklearn.metrics import accuracy_score

Accuracy_Score_NB = accuracy_score(y_test, pred_NB)
Accuracy_Score_NB
```

[17]:
```
0.9874439461883409
```

[18]:
```
from sklearn.neighbors import KNeighborsClassifier

classifier_knn = KNeighborsClassifier()
classifier_knn.fit(trainCV, y_train)
pred_knn = classifier_knn.predict(testCV)
```

[19]:
```
Accuracy_Score_knn = accuracy_score(y_test, pred_knn)
Accuracy_Score_knn
```

[19]:
```
0.9085201793721973
```

[20]:
```
from sklearn.svm import SVC

classifier_svm_linear = SVC(kernel="linear")
classifier_svm_linear.fit(trainCV, y_train)
pred_svm_linear = classifier_svm_linear.predict(testCV)
```

[21]:
```
Accuracy_Score_SVM_Linear = accuracy_score(y_test, pred_svm_linear)
Accuracy_Score_SVM_Linear
```

[21]:
```
0.9811659192825112
```

[22]:
```
classifier_svm_rbf = SVC(kernel="rbf")
classifier_svm_rbf.fit(trainCV, y_train)
pred_svm_rbf = classifier_svm_rbf.predict(testCV)
```

[23]:
```
Accuracy_Score_SVM_Gaussion = accuracy_score(y_test, pred_svm_rbf)
Accuracy_Score_SVM_Gaussion
```

[23]: 0.9766816143497757

[24]:
```
classifier_svm_poly = SVC(kernel="poly")
classifier_svm_poly.fit(trainCV, y_train)
pred_svm_poly = classifier_svm_poly.predict(testCV)
```

[25]:
```
Accuracy_Score_SVM_Polynomial = accuracy_score(y_test, pred_svm_poly)
Accuracy_Score_SVM_Polynomial
```

[25]: 0.9417040358744395

[26]:
```
classifier_svm_sigmoid = SVC(kernel="sigmoid")
classifier_svm_sigmoid.fit(trainCV, y_train)
pred_svm_sigmoid = classifier_svm_sigmoid.predict(testCV)
```

[27]:
```
Accuracy_Score_svm_Sigmoid = accuracy_score(y_test, pred_svm_sigmoid)
Accuracy_Score_svm_Sigmoid
```

[27]: 0.9300448430493273

[28]:
```
from sklearn.tree import DecisionTreeClassifier

classifier_dt = DecisionTreeClassifier()
classifier_dt.fit(trainCV, y_train)
pred_dt = classifier_dt.predict(testCV)
```

[29]:
```
Accuracy_Score_dt = accuracy_score(y_test, pred_dt)
Accuracy_Score_dt
```

[29]: 0.9623318385650225

[30]:
```
from sklearn.ensemble import RandomForestClassifier

classifier_rf = RandomForestClassifier()
classifier_rf.fit(trainCV, y_train)
pred_rf = classifier_rf.predict(testCV)
```

[31]:
```
Accuracy_Score_rf = accuracy_score(y_test, pred_rf)
Accuracy_Score_rf
```

[31]: 0.9721973094170404

[32]:
```
print("K-Nearest Neighbors =", Accuracy_Score_knn)
print("Naive Bayes =", Accuracy_Score_NB)
```

```
print("Support Vector Machine Linear =", Accuracy_Score_SVM_Linear)
print("Support Vector Machine Gaussion =", Accuracy_Score_SVM_Gaussion)
print("Support Vector Machine Polynomial =", Accuracy_Score_SVM_Polynomial)
print("Support Vector Machine Sigmoid =", Accuracy_Score_svm_Sigmoid)
print("Decision Tree =", Accuracy_Score_dt)
print("Random Forest =", Accuracy_Score_rf)
```

```
K-Nearest Neighbors = 0.9085201793721973
Naive Bayes = 0.9874439461883409
Support Vector Machine Linear = 0.9811659192825112
Support Vector Machine Gaussion = 0.9766816143497757
Support Vector Machine Polynomial = 0.9417040358744395
Support Vector Machine Sigmoid = 0.9300448430493273
Decision Tree = 0.9623318385650225
Random Forest = 0.9721973094170404
```