By Nitish Adhikari

Email id :nitishbuzzpro@gmail.com (mailto:nitishbuzzpro@gmail.com), +91-9650740295

Linkedin : https://www.linkedin.com/in/nitish-adhikari-6b2350248 (https://www.linkedin.com/in/nitish-adhikari-6b2350248)

A Project on Natural Language Processing - PASSWORD STRENGTH CLASSIFIER

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
data = pd.read_csv('data.csv',error_bad_lines=False)
```

In [3]:

```
data.head(5)
```

Out[3]:

	password	strength
0	kzde5577	1
1	kino3434	1
2	visi7k1yr	1
3	megzy123	1
4	lamborghin1	1

In [4]:

```
data['strength'].unique()
```

Out[4]:

```
array([1, 2, 0], dtype=int64)
```

In [5]:

```
data.isnull() #check null values
```

Out[5]:

	password	strength
0	False	False
1	False	False
2	False	False
3	False	False
4	False	False
...	...	...
669635	False	False
669636	False	False
669637	False	False
669638	False	False
669639	False	False

669640 rows × 2 columns

In [6]:

```
data.isnull().sum()
```

Out[6]:

```
password    1
strength    0
dtype: int64
```

In [7]:

```
data.dropna(inplace = True) #remove null values
```

In [8]:

```
data.isnull().sum()
```

Out[8]:

```
password    0
strength    0
dtype: int64
```

In [9]:

```
data[data['strength']==0].count()
```

Out[9]:

```
password    89701
strength    89701
dtype: int64
```

In [10]:

```
data[data['strength']==1].count()
```

Out[10]:

```
password    496801
strength    496801
dtype: int64
```

In [11]:

```
data[data['strength']==2].count()
```

Out[11]:

```
password    83137
strength    83137
dtype: int64
```

In [12]:

```
password_tuple=np.array(data) #creating array
password_tuple
```

Out[12]:

```
array([['kzde5577', 1],
       ['kino3434', 1],
       ['visi7k1yr', 1],
       ...,
       ['184520socram', 1],
       ['marken22a', 1],
       ['fxx4pw4g', 1]], dtype=object)
```

In [13]:

```
password_tuple.shape #shape of the array
```

Out[13]:

```
(669639, 2)
```

In [14]:

```
import random
random.shuffle(password_tuple) #shuffle the array
```

In [15]:

```
password_tuple #shuffled array
```

Out[15]:

```
array([['kzde5577', 1],
       ['kino3434', 1],
       ['kzde5577', 1],
       ...,
       ['kobeji659', 1],
       ['kt5tu2o0', 1],
       ['killi48', 0]], dtype=object)
```

In [16]:

```
X = [labels[0] for labels in password_tuple] #list of independent variable
y = [labels[1] for labels in password_tuple] #list of dependent variable
```

In [18]:

```
len(X)
```

Out[18]:

```
669639
```

In [82]:

```
len(y)
```

Out[82]:

```
669639
```

In [21]:

```
def word_divide_char(inputs): #function to split the string to list
    character=[]
    for i in inputs:
        character.append(i)
    return character
```

In [22]:

```
word_divide_char('kzde5577') #check the fuction's working
```

Out[22]:

```
['k', 'z', 'd', 'e', '5', '5', '7', '7']
```

In [23]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [24]:

```
vectorizer=TfidfVectorizer(tokenizer=word_divide_char)
```

In [26]:

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

In [27]:

```
X.shape #shape of sparse matrix
```

Out[27]:

```
(669639, 132)
```

In [28]:

```
print(X) #sparse matrix
```

```
  (0, 34)	0.5917520524694371
  (0, 32)	0.5665331455581984
  (0, 53)	0.2214639539695442
  (0, 52)	0.2855291890678396
  (0, 74)	0.33602096776990453
  (0, 59)	0.2922095342105659
  (1, 31)	0.6175654131802808
  (1, 30)	0.5601711835927342
  (1, 63)	0.2565023277367334
  (1, 62)	0.26785873390846976
  (1, 57)	0.2521638567898762
  (1, 59)	0.3220137409789036
  (2, 34)	0.5917520524694371
  (2, 32)	0.5665331455581984
  (2, 53)	0.2214639539695442
  (2, 52)	0.2855291890678396
  (2, 74)	0.33602096776990453
  (2, 59)	0.2922095342105659
  (3, 34)	0.5917520524694371
  (3, 32)	0.5665331455581984
```

In [29]:

```
vectorizer.get_feature_names()
```

Out[29]:

```
['\x02',
 '\x05',
 '\x06',
 '\x08',
 '\x0f',
 '\x10',
 '\x11',
 '\x16',
 '\x17',
 '\x19',
 '\x1b',
 '\x1c',
 '\x1e',
 ' ',
 '!',
 '"',
 '#',
 '$',
```

In [30]:

```
X.shape
```

Out[30]:

```
(669639, 132)
```

In [31]:

```
first_document_vector= X[0]
first_document_vector
```

Out[31]:

```
<1x132 sparse matrix of type '<class 'numpy.float64'>'
	with 6 stored elements in Compressed Sparse Row format>
```

In [32]:

```
print(first_document_vector) #Sparse matrix of first_document_vector
```

```
  (0, 34)	0.5917520524694371
  (0, 32)	0.5665331455581984
  (0, 53)	0.2214639539695442
  (0, 52)	0.2855291890678396
  (0, 74)	0.33602096776990453
  (0, 59)	0.2922095342105659
```

In [33]:

```
print(first_document_vector.T) #Transpose of first_document_vector
```

```
  (34, 0)	0.5917520524694371
  (32, 0)	0.5665331455581984
  (53, 0)	0.2214639539695442
  (52, 0)	0.2855291890678396
  (74, 0)	0.33602096776990453
  (59, 0)	0.2922095342105659
```

In [34]:

```
first_document_vector.T.todense() #Dense matrix representation of Transpose of first_document_vector
```

```
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.56653315],
        [0.        ],
        [0.59175205],
```

In [35]:

```
pd.DataFrame(first_document_vector.T.todense(),
             index=vectorizer.get_feature_names(),
             columns=['Tf-Idf'],
            ).sort_values(by='Tf-Idf',ascending=False)
```

Out[35]:

	Tf-Idf
7	0.591752
5	0.566533
z	0.336021
k	0.292210
d	0.285529
...	...
=	0.000000
<	0.000000
;	0.000000
9	0.000000
™	0.000000

132 rows × 1 columns

In [36]:

```
from sklearn.model_selection import train_test_split
```

In [37]:

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

In [38]:

```
type(X_train)
```

Out[38]:

```
scipy.sparse._csr.csr_matrix
```

In [39]:

```
X_train.shape
```

Out[39]:

```
(535711, 132)
```

In [40]:

```
type(y_train)
```

Out[40]:

```
list
```

In [41]:

```
from sklearn.linear_model import LogisticRegression
```

In [42]:

```
clf = LogisticRegression(random_state=0,multi_class='multinomial')
```

In [43]:

```
clf.fit(X_train,y_train)
```

Out[43]:

```
▾                     LogisticRegression
LogisticRegression(multi_class='multinomial', random_state=0)
```

In [44]:

```
y_pred=clf.predict(X_test)
y_pred
```

Out[44]:

```
array([1, 1, 1, ..., 1, 1, 2])
```

In [45]:

```
from sklearn.metrics import confusion_matrix, accuracy_score
```

In [46]:

```
confusion_matrix(y_test,y_pred)
```

Out[46]:

```
array([[ 5381, 12513,     8],
       [ 3864, 93046,  2685],
       [   37,  5033, 11361]], dtype=int64)
```

In [47]:

```
accuracy_score(y_test,y_pred)
```

Out[47]:

```
0.8197538976166299
```

In [68]:

```
dt = ['ru76799sdhoh%41']  #Predicting strength of password 'ru76799sdhoh%41'
dt = vectorizer.transform(dt)
clf.predict(dt)
```

Out[68]:

```
array([1])
```

Clasification is 1, means password is average

```
In [70]:
```

```
dt = ['a1']  #Predicting strength of password 'a1'
dt = vectorizer.transform(dt)
clf.predict(dt)
```

```
Out[70]:

array([0])
```

Clasification is 0, means password is weak

```
In [80]:
```

```
dt = ['AsD234Ads&*^%SGSJ7736SK1']  #Predicting strength of password 'AsD234Ads&*^%SGSJ7736SK1'
dt = vectorizer.transform(dt)
clf.predict(dt)
```

```
Out[80]:

array([2])
```

Clasification is 2, means password is Strong!!

Complete!!