Password_Strength_Classifier

August 3, 2023

1 Password Strength Classifier

[1]:
```
#libraries
import pandas as pd
import numpy as np
```

[2]:
```
# read data
data = pd.read_csv('data.csv', on_bad_lines='skip')
data
```

[2]:

	password	strength
0	kzde5577	1
1	kino3434	1
2	visi7k1yr	1
3	megzy123	1
4	lamborghin1	1
…	…	…
669635	10redtux10	1
669636	infrared1	1
669637	184520socram	1
669638	marken22a	1
669639	fxx4pw4g	1

[669640 rows x 2 columns]

[3]:
```
# extra info
print(data.columns)
print(data.size)
print(data.dtypes)
```

```
Index(['password', 'strength'], dtype='object')
1339280
password    object
strength     int64
dtype: object
```

[4]:
```
# unique data
data['strength'].unique()
```

[4]:
```
array([1, 2, 0], dtype=int64)
```

[5]:
```
# check total missing data
data.isna().sum()
```

[5]:
```
password    1
strength    0
dtype: int64
```

[6]:
```
# find missing data
data[data['password'].isnull()]
```

[6]:
```
        password  strength
367579      NaN         0
```

[7]:
```
# drop missing data
data.dropna(inplace = True)
```

[8]:
```
# again check missing data
data.isnull().sum()
```

[8]:
```
password    0
strength    0
dtype: int64
```

[9]:
```
# best visualization plot
import plotly.express as px
# plot
fig = px.histogram(data, x='strength', color='strength', title='Countplot of␣
 ↪Strength')
fig.show()
```

The data has high count of 1, so the data is imbalanced

[10]:
```
# converting to array data so we can perform on it ,can be imported to model by␣
 ↪dataframe also
password_tuple = np.array(data)
password_tuple
```

[10]:
```
array([['kzde5577', 1],
       ['kino3434', 1],
       ['visi7k1yr', 1],
       …,
       ['184520socram', 1],
       ['marken22a', 1],
       ['fxx4pw4g', 1]], dtype=object)
```

[11]:
```
# shuffle data
import random
random.shuffle(password_tuple)
```

[12]:
```
# split data
X = [labels[0] for labels in password_tuple]
y = [labels[1] for labels in password_tuple]
```

[13]:
```
#data passed to tfidf must be in the form of character but not in word, as we␣
 ↪determine the strength of password based on character
#create a custom function to split input into characters of list
def word_divide(inputs):
    character = []
    for i in inputs:
        character.append(i)
    return character
word_divide('kzde5577')
```

[13]:
```
['k', 'z', 'd', 'e', '5', '5', '7', '7']
```

[14]:
```
# vectorize(give numerical value acc to the character) the characters from␣
 ↪string to numerical data
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(tokenizer=word_divide)
X_tf = vectorizer.fit_transform(X)
X_tf.shape
```

```
C:\Users\Abdul Mateen\anaconda3\lib\site-
packages\sklearn\feature_extraction\text.py:528: UserWarning:

The parameter 'token_pattern' will not be used since 'tokenizer' is not None'
```

[14]:
```
(669639, 129)
```

[15]:
```
# 126 features/vector, this method gives all features name
vectorizer.get_feature_names_out()
```

[15]:
```
array(['\x05', '\x06', '\x08', '\x0f', '\x10', '\x11', '\x12', '\x13',
       '\x16', '\x17', '\x19', '\x1b', '\x1c', '\x1d', '\x1e', ' ', '!',
       '"', '#', '$', '%', '&', '(', ')', '*', '+', '-', '.', '/', '0',
       '1', '2', '3', '4', '5', '6', '7', '8', '9', ';', '<', '=', '>',
       '?', '@', '[', '\\', ']', '^', '_', '`', 'a', 'b', 'c', 'd', 'e',
       'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r',
       's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '{', '|', '}', '~', '\x7f',
       '\xa0', '¡', '¦', '¨', '°', '±', '²', '³', '´', 'µ', '¶', '·', '¹',
       'º', '¼', '¿', 'ß', 'à', 'á', 'ä', 'å', 'æ', 'ç', 'è', 'é', 'ê',
       'í', 'î', 'ï', 'ð', 'ñ', 'ò', 'ó', 'ô', 'õ', 'ö', '÷', 'ù', 'ú',
```

```
       'û', 'ü', 'ý', 'þ', 'ÿ', '‚', '…', ' '], dtype=object)
```

[16]:
```
# first document (124,)
first_document_vector = X_tf[0]
first_document_vector
```

[16]:
```
<1x129 sparse matrix of type '<class 'numpy.float64'>'
        with 6 stored elements in Compressed Sparse Row format>
```

[17]:
```
#(change it to (124,1) the .todense transposes the data so add T to neutralize
 ↪the effect)
first_document_vector.T.todense()
```

[17]:
```
matrix([[0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
```

```
[0.56714571],
[0.        ],
[0.59113522],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.28560851],
[0.22129726],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.29207797],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.33623006],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
```

```
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
[0.        ],
```

```
       [0.        ]])
```

[18]:
```
# build a dataframe in the descending order to use for train test split
data1 = pd.DataFrame(first_document_vector.T.todense(),index = vectorizer.
 ↪get_feature_names_out(), columns = ['TF-IDF'])
data1.sort_values(by = ['TF-IDF'], ascending = False)
```

[18]:
```
      TF-IDF
7   0.591135
5   0.567146
z   0.336230
k   0.292078
d   0.285609
..       …
9   0.000000
8   0.000000
6   0.000000
4   0.000000
   0.000000

[129 rows x 1 columns]
```

[19]:
```
# split the data in train and test dataset
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_tf, y)
```

[20]:
```
# balancing imbalance data
from imblearn.over_sampling import RandomOverSampler
os = RandomOverSampler()
X_train_res, y_train_res = os.fit_resample(X_train,y_train)
from collections import Counter
print('Original Dataset shape {}'.format(Counter(y_train)))
print('Resampled Dataset shape {}'.format(Counter(y_train_res)))
```

```
Original Dataset shape Counter({1: 372495, 0: 67204, 2: 62530})
Resampled Dataset shape Counter({1: 372495, 0: 372495, 2: 372495})
```

[21]:
```
# build logistic model with multiple classes(multinomial)
from sklearn.linear_model import LogisticRegression
log = LogisticRegression(random_state=0, multi_class='multinomial',␣
 ↪max_iter=1000)
log.fit(X_train_res, y_train_res)
```

[21]:
```
LogisticRegression(max_iter=1000, multi_class='multinomial', random_state=0)
```

[22]:
```
# sample test data provied to check if the function works
# convert pass to array then vectorize it
dt = np.array(['%@123abcd'])
```

```
pred = vectorizer.transform(dt)
log.predict(pred)
```

[22]:
```
array([2])
```

[23]:
```
# predict the value for test data
y_pred = log.predict(X_test)
y_pred
```

[23]:
```
array([2, 0, 1, …, 1, 1, 1])
```

[24]:
```
# check confusion_matrix, accuracy_model, classification_report
from sklearn.metrics import confusion_matrix , accuracy_score,␣
 ↪classification_report
print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[18773  3640    37]
 [25387 84297 14352]
 [  287  1950 18687]]
0.7272982498058659
              precision    recall  f1-score   support

           0       0.42      0.84      0.56     22450
           1       0.94      0.68      0.79    124036
           2       0.56      0.89      0.69     20924

    accuracy                           0.73    167410
   macro avg       0.64      0.80      0.68    167410
weighted avg       0.82      0.73      0.75    167410
```

[ ]: