

Title of the project:

Predict Password Strength using Machine Learning (ML) & Natural Language Processing (NLP) in Python

Project short description:

Hello everyone!

In this tutorial, we are going to build a model to predict the password strength using the Logistic Regression Machine Learning Algorithm & Natural Language Processing (NLP) in Python.

A user must create a strong password so that their data is secure from unauthorized users. So, using this model, we will predict whether the password strength is strong or average, or weak.

Prerequisites:

- 1) You need to have a dataset file with a .csv extension.
- 2) Install Jupyter Notebook or any similar working environment with the latest version of Python installed.
- 3) You must know the Python language and concepts of Natural Language Processing (NLP) & Machine Learning (ML).

About the dataset:

The dataset contains around 670,000 different passwords of different strengths. The strength of a password is denoted by a number i.e., if

- ➡ password strength=2, it is a strong password
- ➡ password strength=1, it is an average strength password
- ➡ password strength=0, it is a weak password

The password strength is based on rules such as lowercase English letters, digits, uppercase English letters, special characters, etc.

Step by step implementation:

- 1) First of all, import the required libraries

```
In [1]: import numpy as np
import pandas as pd
import seaborn as sns
```

```
In [2]: import warnings
warnings.filterwarnings('ignore')
```

- 2) Reading the dataset

```
In [3]: data=pd.read_csv('E:\DS\Predict Password Strength using Natural Language Processing\data.csv') #Enter your dataset
data.head() #Location on your computer.
```

```
Out[3]:
```

	password	strength
0	lkaj8899	1
1	cgsu5858	1
2	xyfy3y1qw	1
3	dyui159	1
4	qwertyuiop0	1

3) For viewing the unique values in 'strength' column in our dataset,

```
In [4]: data['strength'].unique() # Here 1 refers average strength, 2 refers strong strength and 0 refers weak strength.  
Out[4]: array([1, 2, 0], dtype=int64)
```

4) Code to remove missing(Null/NaN) values in the dataset. The presence of Null values in the dataset will decrease the accuracy of the model.

```
In [5]: data.isna().sum()  
Out[5]: password    1  
        strength    0  
        dtype: int64  
  
In [6]: data[data['password'].isnull()]  
Out[6]:  
        password strength  
367579      NaN      0  
  
In [7]: data.dropna(inplace=True)  
  
In [8]: data.isnull().sum()  
Out[8]: password    0  
        strength    0  
        dtype: int64  
  
In [9]: password_arr=np.array(data)  
  
In [10]: password_arr  
Out[10]: array([[ 'lkaj8899', 1],  
                [ 'cgsu5858', 1],  
                [ 'xyfy3y1qw', 1],  
                ...,  
                [ '184520socram', 1],  
                [ 'marken22a', 1],  
                [ 'fxx4pw4g', 1]], dtype=object)
```

5) Random shuffling will give the model robustness

```
In [11]: import random  
         random.shuffle(password_arr)  
  
In [12]: x=[passwords[0] for passwords in password_arr]  
         y=[strength[1] for strength in password_arr]  
  
In [13]: x  
Out[13]: [ 'lkaj8899',  
          'cgsu5858',  
          'lkaj8899',  
          'xyfy3y1qw',  
          'qwertyuiop0',  
          'xyfy3y1qw',  
          'xyfy3y1qw',  
          'qwertyuiop0',  
          'qwertyuiop0',  
          'qwertyuiop0',  
          'AVYq11DE4MgAZfnt',  
          'asv5o9yu',  
          'AVYq11DE4MgAZfnt',  
          'asv5o9yu',  
          'u6c8vhow',  
          'lkaj8899',  
          'AVYq11DE4MgAZfnt',  
          'xyfy3y1qw',  
          'cgsu5858',  
          'lkaj8899',  
          'AVYq11DE4MgAZfnt']
```

6) Create a function to split the input into characters of list

```
In [14]: def split(inputs):
          character=[]
          for i in inputs:
              character.append(i)
          return character

In [36]: split('CodeSpeedy')
Out[36]: ['C', 'o', 'd', 'e', 's', 'p', 'e', 'e', 'd', 'y']
```

7) Importing TF-IDF vectorizer,

```
In [16]: from sklearn.feature_extraction.text import TfidfVectorizer
```

The goal of using tf-idf is to scale down the impact of tokens that occur very frequently in a given corpus and that are hence empirically less informative than features that occur in a small fraction of the training corpus.

➡ Term Frequency (tf) - It tells us, how frequently a term occurs in a document.

$$TF(t) = \frac{\text{(Number of times term } t \text{ appears in a document)}}{\text{(Total number of terms in a document)}}$$

➡ Inverse Data Frequency (idf) – It tells us the weight of rare words. The words that occur rarely in the corpus have a high IDF score.

$$IDF(t) = \log \frac{\text{(Total Number of documents)}}{\text{(Number of documents with term } t \text{ in it)}}$$

➡ TF-IDF example:

Let us take two sentences

sentence 1– earth is the third planet from the sun

sentence 2– Jupiter is the largest planet

We will now calculate the TF-IDF scores:

Word	TF (Sentence 1)	TF (Sentence 2)	IDF	TF*IDF (sentence 1)	TF*IDF (Sentence 2)
earth	1/8	0	$\log(2/1)=0$	0.0375	0
is	1/8	1/5	$\log(2/2)=0$	0	0
the	2/8	1/5	$\log(2/2)=0$	0	0
third	1/8	0	$\log(2/1)=0.3$	0.0375	0
planet	1/8	1/5	$\log(2/2)=0$	0	0
from	0	0	$\log(2/1)=0.3$	0	0
sun	1/8	0	$\log(2/1)=0.3$	0.0375	0
largest	0	1/5	$\log(2/1)=0.3$	0	0.06
Jupiter	0	1/5	$\log(2/1)=0.3$	0	0.06

From the above table, we can say that TF-IDF is zero for common words, which shows that they are not significant. But the TF-IDF is non-zero for important words such as 'earth', 'third', 'sun', 'largest', 'Jupiter'.

Instantiating TfidfVectorizer(),

```
In [17]: vectorizer=TfidfVectorizer(tokenizer=split)
```

8) Applying TfidfVectorizer on the dataset

```
In [18]: Matrix=vectorizer.fit_transform(x) # Transform the data: X
```

```
In [19]: print(vectorizer.vocabulary_)# We can see that all words were made lowercase by default and that the punctuation was ignored if
#it is there.
# Returns a dictionary comprised of the tokens and their respective indices in the array.
```

{ 'l': 59, 'k': 58, 'a': 48, 'j': 70, '8': 34, '9': 35, 'c': 50, 'g': 54, 's': 66, 'u': 68, '5': 31, 'x': 71, 'y': 72, 'f': 53, '3': 29, 'n': 27, 'q': 64, 'w': 70, 'e': 52, 'r': 65, 't': 67, 'i': 56, 'o': 62, 'p': 63, '0': 26, 'v': 69, 'd': 51, '4': 30, 'm': 60, 'z': 73, 'q': 61, '6': 32, 'h': 55, '7': 33, '2': 28, 'b': 49, '1': 63, '@': 41, 'x': 39, '9': 24, '8': 18, '?': 40, '<': 37, 'l': 13, 'i': 46, 'j': 36, '%': 17, '\$': 16, '1': 85, '1': 12, '^': 45, '#': 15, '4': 22, '*': 21, '": 14, '~': 77, '7': 25, '0': 20, '[': 42, ']'': 44, 'j': 74, '1': 76, 'p': 121, '6': 112, '1': 38, '": 47, 'x1c': 10, '0': 113, '': 19, '\\': 43, 'x19': 8, 'u': 117, 'i': 93, 'a': 101, '1': 123, 'a': 100, 'c': 83, 'a': 96, 'x7f': 78, '2': 86, '3': 87, 'x05': '1, 'x1b': 9, 'y': 122, 'd': 109, 'l': 75, 'x16': 6, '0': 111, '1': 90, 'x1e': 11, '9': 84, 'x10': 4, 'x17': 7, '0': 114, '8': 95, 'x08': 3, 'e': 106, 'a': 97, '÷': 116, 'x': 92, '": 88, 'ü': 119, 'ö': 115, 'xao': 80, 'æ': 102, 'è': 104, '": 82, '0': 118, 'x': 94, 'y': 120, 'a': 98, 'e': 105, 'x06': 2, 'j': 81, 'c': 103, 'i': 108, 'x8d': 79, 'µ': 89, 'ñ': 110, 'x12': 5, 'x02': 0, 'i': 107, 'a': 99, 'e': 91}

```
In [20]: Matrix.shape #We have 669639 rows (669639 passwords) and 124 columns (124 unique words).
```

```
Out[20]: (669639, 124)
```

```
In [21]: vectorizer.get_feature_names() # Print all the features(unique words) of the vectorizer.
```

```
Out[21]: ['\x02',
'\x05',
'\x06',
'\x08',
'\x10',
'\x12',
'\x16',
'\x17',
'\x19',
'\x1b',
'\x1c',
'\x1e',
',',
'!',
'",',
'#',
'$',
'%',
'&',
'<']
```

```
In [22]: FirstDocVector=Matrix[0]
FirstDocVector
```

```
Out[22]: <1x124 sparse matrix of type '<class 'numpy.float64''>'
         with 6 stored elements in Compressed Sparse Row format>
```

```
In [23]: FirstDocVector.T.todense() # Print the sparse matrix of the test data.
```

[illegible]

```
In [24]: df=pd.DataFrame(FirstDocVector.T.todense(),index=vectorizer.get_feature_names(),columns=['TF-IDF'])
df.sort_values(by=['TF-IDF'],ascending=False)
```

Out[24]:

	TF-IDF
8	0.596748
9	0.566175
j	0.338298
k	0.302095
l	0.281408
...	...
<	0.000000
;	0.000000
7	0.000000
6	0.000000
,	0.000000

124 rows x 1 columns

9) Splitting the data into training set and test set:

Training set= A subset to train my model

Test set= A subset to test my trained model

```
In [25]: from sklearn.model_selection import train_test_split

In [26]: Matrix_train, Matrix_test, y_train, y_test=train_test_split(Matrix,y,test_size=0.3)
#X->independent data, x->dependent data
#0.3->30% of my data is considered for the testing purpose. Therefore the rest 70% of my data will be considered for the
#training purpose.
```

Here, we are setting test_size as 0.3 and it means that the 30% of the data is considered for the testing purpose and the rest 70% of the data will be considered for the training purpose.

```
In [27]: Matrix_train.shape

Out[27]: (468747, 124)
```

10) Applying Logistic Regression as we are doing classification of passwords

```
In [28]: from sklearn.linear_model import LogisticRegression

In [29]: clf=LogisticRegression(random_state=0,multi_class='multinomial')
```

We are setting multi_class parameter as 'multinomial' because we have more than 2 categories in the data, i.e. 0, 1 and 2. That is why we are considering a case of multinomial Logistic Regression.

```
In [30]: clf.fit(Matrix_train,y_train)

Out[30]: LogisticRegression(multi_class='multinomial', random_state=0)
```

11) Performing predictions on the strength of passwords which are outside of the data

```
In [37]: dt=np.array(['CodeSpeedy123@#'])
prediction=vectorizer.transform(dt)
clf.predict(prediction)

Out[37]: array([2])
```

Here, the password- 'CodeSpeedy123@#'\$ that I have entered is showing the password strength= 2, i.e., it is of strong strength.

12) Performing Prediction on Matrix-Test data

```
In [32]: y_prediction=clf.predict(Matrix_test)
y_prediction

Out[32]: array([1, 1, 1, ..., 2, 1, 1])
```

13) Checking accuracy of the model using confusion_matrix,accuracy_score

```
In [33]: from sklearn.metrics import confusion_matrix,accuracy_score

In [34]: cm=confusion_matrix(y_test,y_prediction)
print(cm)
print(accuracy_score(y_test,y_prediction))

[[ 7959 19124    26]
 [ 5590 139328  3882]
 [    57  7674 17252]]
0.8190420723572865
```

The model has an accuracy of 81.9%

14) Report of the model

```
from sklearn.metrics import classification_report  
print(classification_report(y_test,y_prediction))
```

	precision	recall	f1-score	support
0	0.58	0.29	0.39	27109
1	0.84	0.94	0.88	148800
2	0.82	0.69	0.75	24983
accuracy			0.82	200892
macro avg	0.75	0.64	0.67	200892
weighted avg	0.80	0.82	0.80	200892