

Description about Coding Standard

Twitter User Analyser

Submitted By:

IIT2016106 - Aswin VB

IIT2016105 - Aswanth K

IIT2016133 - Utpal Aman

BIM2016501 - Druval CR

IRM2016006 - Sarath Nandimandalam

Introduction

With a significant increase in the number of active social media users, the chances of an account being fake or anomalous have certainly gone higher. Thus, it is very much needed to identify whether a user is anomalous or not. And if the user is found to be anomalous, one should restrict the user from using the social media any further because that user might be a threat to other genuine users. On the other hand, the user can also be a robot which can be used to manipulate the environment of the social media. Here the social media is twitter and the dataset constitutes of the tweets of different users, their timings of the tweet and the content of the tweet. We have certain aspects that we can use to differentiate between a genuine user and an anomalous user. Also, we have given certain weightage to each of the aspects which eventually help us to calculate the FAL rank of the user based on which we categorize user into two categories, anomalous and genuine.

Indentation

Indentation is something one has to care about while writing a code in python. However, you should be careful with it, as it can lead to syntax errors. The recommendation is, therefore, to use 4 spaces for indentation. For example, this statement uses 4 spaces of indentation:

If True:

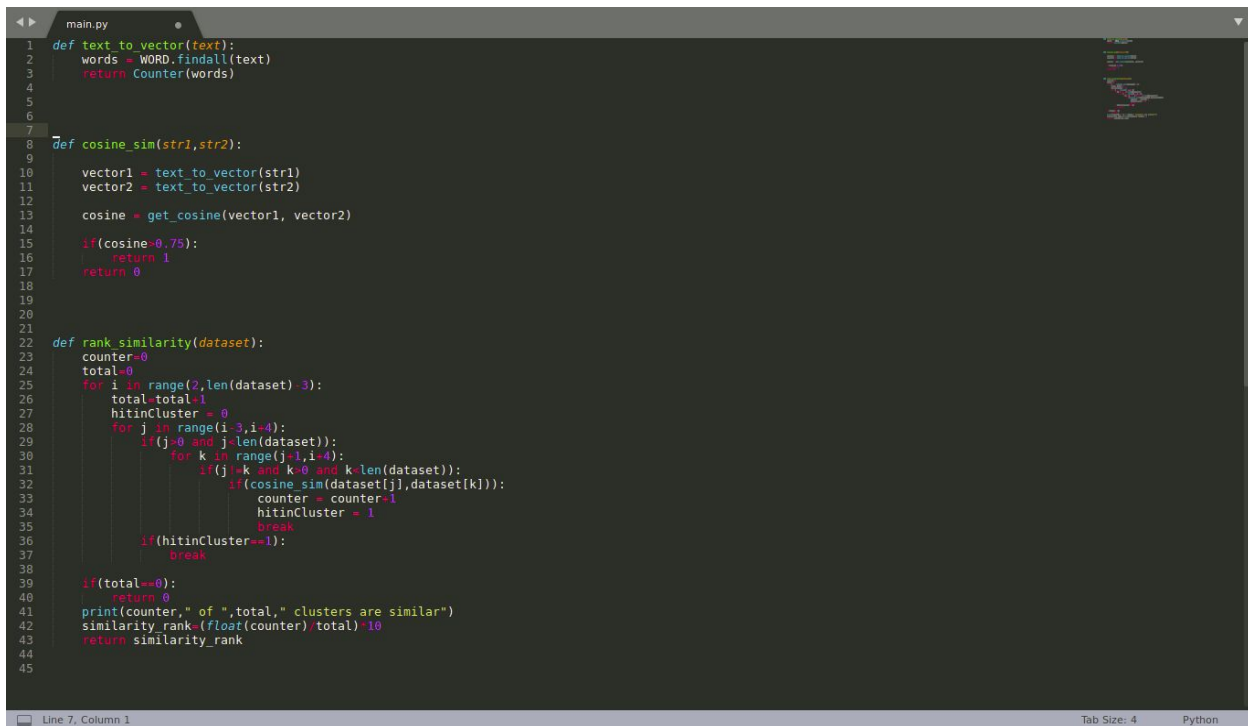
 Print "it works!"

When you write a big expression, it is best to keep the expression vertically aligned. When you do this, you'll create a "hanging indent".

```
value = square_of_numbers(num1, num2, num3, num4)
```

Blank Lines


To increase the readability of the code, we have left between 3-5 blank lines after every module. For example, after each function, a few blank lines separate the next module



```
1 def text_to_vector(text):
2     words = WORDS.findall(text)
3     return Counter(words)
4
5
6
7
8 def cosine_sim(str1, str2):
9
10    vector1 = text_to_vector(str1)
11    vector2 = text_to_vector(str2)
12
13    cosine = get_cosine(vector1, vector2)
14
15    if(cosine>0.75):
16        return 1
17    return 0
18
19
20
21
22 def rank_similarity(dataset):
23     counter=0
24     total=0
25     for i in range(2, len(dataset)-3):
26         total+=1
27         hitinCluster = 0
28         for j in range(i+3, i+4):
29             if(j<0 and j>len(dataset)):
30                 for k in range(j+1, i+4):
31                     if(j!=k and k>0 and k<len(dataset)):
32                         if(cosine_sim(dataset[j], dataset[k])):
33                             counter += counter+1
34                             hitinCluster = 1
35                             break
36             if(hitinCluster==1):
37                 break
38
39     if(total==0):
40         return 0
41     print(counter, " of ", total, " clusters are similar")
42     similarity_rank=(float(counter)/total)*10
43     return similarity_rank
44
45
```

Source File Encoding

A computer cannot store "letters", "numbers", "pictures" or anything else; It can only store and work with bits, which can only have binary values: yes or no, true or false, 1 or 0, etc. As you already know, a computer works with electricity; This means that an "actual" bit is the presence or absence of a blip of electricity. You would usually represent this (lack of) presence with 1 and 0.



To use bits to represent anything at all besides bits, you need a set of rules. You need to convert a sequence of bits into something like letters, numbers and pictures using an encoding scheme or encoding. Examples of encoding schemes are ASCII, UTF-8, etc:

Unicode Worldwide Character Standard, or Unicode in short, is a system for "the interchange, processing, and display of the written texts of the diverse languages of the modern world". In short, Unicode is designed to accommodate all of the world's known writing systems. Unicode currently employs three different encodings to represent Unicode character sets: UTF-8, UTF-16, and UTF-32.

- UTF-8 is another type of Unicode variable-length encoding, using one to four 8-bit bytes.

For this project, the python file uses UTF-8 encoding.

Imports

As is the general convention, we have imported all the required modules right at the start of the script.

Additionally, we took into account the fact that there is an order that we need to respect when we are importing libraries. In general, we follow this order as shown in the following screenshot:

1. Standard library imports.
2. Related third-party imports.
3. Local application/library specific imports.

```
import tweepy
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import re, math
from collections import Counter
import urllib, sys, bs4
from collections import Counter
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import datetime
import os
import requests
```

Comments

We have used comments and when we thought there was a need to explain the use of a particular line of code i.e. to simplify the working and explain the use of a particular line or a block of lines of code.

Naming Conventions

The following table shows you some general guidelines on how to name your identifiers:

Identifier	Convention
Module	lowercase

Class	CapWords
Functions	lowercase
Methods	lowercase
Type variables	CapWords
Constants	UPPERCASE
Package	lowercase

As seen from the screenshot posted below, we can see that all the module have been named in lowercase, the class has been named using the cap words notation and functions using lowercase.

```
from django.db import models

class Document(models.Model):
    username=models.CharField(max_length=200,default="")
```

```

def get_cosine(vec1, vec2):
    intersection = set(vec1.keys()) & set(vec2.keys())
    numerator = sum([vec1[x] * vec2[x] for x in intersection])

    sum1 = sum([vec1[x]**2 for x in vec1.keys()])
    sum2 = sum([vec2[x]**2 for x in vec2.keys()])
    denominator = math.sqrt(sum1) * math.sqrt(sum2)

    if not denominator:
        return 0.0
    else:
        return float(numerator) / denominator

def text_to_vector(text):
    words = WORD.findall(text)
    return Counter(words)

def cosine_sim(str1, str2):
    vector1 = text_to_vector(str1)
    vector2 = text_to_vector(str2)

    cosine = get_cosine(vector1, vector2)

    if(cosine>0.75):
        return 1
    return 0

```

Thus, we developed our project using the above-mentioned coding standards. The following image depicts the advantages of following such a coding standard.

