Twitter Sentiment Analysis

- Augustine Jose

Connecting to Twitter by using access tokens fetch data from Twitter

consumer_key= 'Kzzmf7XB9oZCvjOE*********'
consumer_secret= 'Y2YQTkTjhjPQUFzGKsF6tk8gbPlH4ndk******************'
access_token= '126311590073300**************************************'
access_token_secret= '*****************PYVBOLeaeGMZHXkvqmooK'

```
auth = tw.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tw.API(auth, wait_on_rate_limit=True)
```

My experience

- After fetching the tweets from my twitter account and now it's the time to check if my tweets have the same relation or rather different order.
- So I have converted my tweets into vectors so that the machine could understand.
- And I calculated the similarity between them, the cosine similarity.
- That I have converted to an array where it shows less similarity are ignored.
- Then by using the Principle Component Analysis and clearly mentioned it should be 2-D, so 10x10 reduced to 10x2, so that it would make a better coordinate system.
- I have transformed the cosine similarity based on this pca.
- Then I have used the k-means algorithm to form the clusters.
- I faced difficulty in calculating the similarity among texts.
- The final part which is to draw the scatter plot of clusters, I wasn't able to complete it since I have a dilemma on pointing to the co-ordinates.

Getting vector from texts

```
def get_vectors(text):
```

```
    #text = [t for t in str]
    tfidf_vectorizer = TfidfVectorizer(text)
    tfidf_vectorizer.fit(text)
    return tfidf_vectorizer.transform(text).toarray()
#text_1_array = tfidf_vectorizer.transform(tokens_without_sw_text_1).toarray()
#text_1_array

def get_cosine_sim(vectors):
    return cosine_similarity(vectors)


text = [t for t in df.Tweet]


vectors = get_vectors(text) # getting vector
cos_sim = get_cosine_sim(vectors) #cosine similarity
cos_sim


array([[1.        , 0.06741184, 0.        , 0.06103794, 0.        ,
        0.07184351, 0.05110424, 0.08881037, 0.        , 0.0198757 ],
       [0.06741184, 1.        , 0.04773349, 0.        , 0.0772239 ,
        0.03955016, 0.11176628, 0.        , 0.05830578, 0.06130051],
       [0.        , 0.04773349, 1.        , 0.03591799, 0.06193225,
        0.04369208, 0.04003677, 0.04129921, 0.02910858, 0.11289034],
       [0.06103794, 0.        , 0.03591799, 1.        , 0.02905431,
        0.02976028, 0.02727052, 0.05132474, 0.        , 0.12048631],
       [0.        , 0.0772239 , 0.06193225, 0.02905431, 1.        ,
        0.0513147 , 0.04702169, 0.03340722, 0.04709226, 0.02578673],
       [0.07184351, 0.03955016, 0.04369208, 0.02976028, 0.0513147 ,
        1.        , 0.03317295, 0.14280184, 0.02411826, 0.01320665],
       [0.05110424, 0.11176628, 0.04003677, 0.02727052, 0.04702169,
        0.03317295, 1.        , 0.03135618, 0.29095508, 0.03436953],
       [0.08881037, 0.        , 0.04129921, 0.05132474, 0.03340722,
        0.14280184, 0.03135618, 1.        , 0.        , 0.03463438],
       [0.        , 0.05830578, 0.02910858, 0.        , 0.04709226,
        0.02411826, 0.29095508, 0.        , 1.        , 0.01946956],
       [0.0198757 , 0.06130051, 0.11289034, 0.12048631, 0.02578673,
        0.01320665, 0.03436953, 0.03463438, 0.01946956, 1.        ]])
```

Reducing dimensionality to 2-D with PCA and using a clustering algorithm to form clusters.

```
pca=PCA(n_components=2)
```

```
pca.fit(cos_sim)
```

```
   PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
```

```
x_pca=pca.transform(cos_sim)
```

```
cos_sim.shape
```

```
(10, 10)
```

```
x_pca.shape #converted to 2 dimension
```

```
(10, 2)
```

```
x_pca
```

```
array([[-0.24713623, -0.39173644],
       [ 0.20826311,  0.0615023 ],
       [-0.12972199,  0.43672142],
       [-0.34370291,  0.2936762 ],
       [ 0.02501734,  0.11126726],
       [-0.29100971, -0.46829393],
       [ 0.67481576, -0.11617422],
       [-0.40683919, -0.43156864],
       [ 0.72906611, -0.0821803 ],
       [-0.21875228,  0.58678635]])
```

```
kmeans=KMeans(n_clusters=4)

kmeans.fit(x_pca)


KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=4, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)

clusters=kmeans.cluster_centers_ #finding clusters


print(clusters)

[[ 0.70194093 -0.09917726]
 [ 0.11664023  0.08638478]
 [-0.31499504 -0.43053301]
 [-0.23072573  0.43906133]]
```

Does the cluster indicate relevant tweets?

Yes, it should be and in my case since I wasn't able to draw the scatter plot of the clusters, I cannot say the cluster indicates relevant tweets. But looking at my tweets I can say, 3,4,5,6,8 can form a cluster. And these are the relevant tweets and most probably others are less relevant.