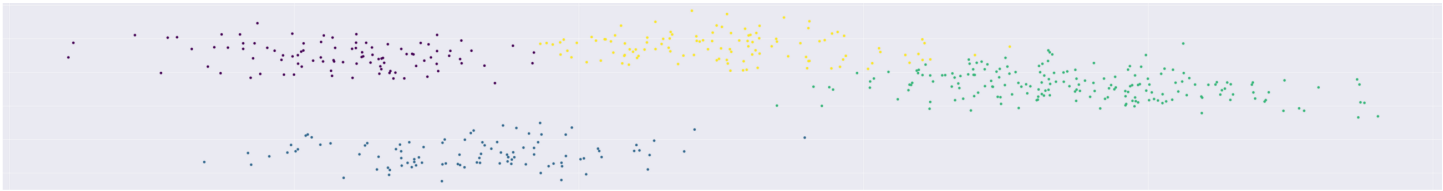

Text clustering with K-means and tf-idf

Mihail Salnikov Follow
Aug 5, 2018 · 4 min read

First of all, I'm not a native English speaker, then I will probably make a lot of mistakes, sorry about that.

In this post, I'll try to describe how to clustering text with knowledge, how important word is to a string. Same words in different strings can be badly affected to clustering this kind of data isn't important for deciding. The first part of this publication is the general information about TF-IDF with examples on Python. In the second part, I'll provide you the example showed how this approach can be applied to real tasks.

TF-IDF is useful for clustering tasks, like a document clustering or in other words, tf-idf can help you understand what kind of document you got now.

TF-IDF

Term Frequency-Inverse Document Frequency is a numerical statistic that demonstrates how important a word is to a corpus.

Term Frequency is just ratio number of current word to the number of all words in document/string/etc.

$$tf(t,d) = \frac{n_t}{\sum_k n_k}$$

Frequency of term t_i, where n_t—the number of t_i in current document/string, the sum of n_k is the number of all terms in current

document/string.

Inverse Document Frequency is a log of the ratio of the number of all documents/string in the corpus to the number of documents with term t_i.

$$idf(t, D) = \log \frac{|D|}{|\{ d_i \in D \mid t \in d_i \}|}$$

tf-idf(t, d, D) is the product tf(t, d) to idf(t, D).

$$tf\text{–}idf(t, d, D) = \mathrm{tf}(t, d) \cdot \mathrm{idf}(t, D)$$

If you want more theoretic information about TF-IDF I want advice you read publication on Wikipedia about it or read NLP Stanford post.

Well, now time for a real example on Python.

TF-IDF example on Python

For all code below you need python 3.5 or newer and scikit-learn *and* pandas *packages.*

Firstly, let's talk about a data set. For this really simple example, I just set a simple corpus with 3 strings. In this example, strings play a role documents.

```
corpus = """
Simple example with Cats and Mouse
Another simple example with dogs and cats
Another simple example with mouse and cheese
```

After that lets make bags of words for our corpus and for every string too. But before we have to clear the data.

```
# clearing and tokenizing
l_A = corpus[0].lower().split()
l_B = corpus[1].lower().split()
l_C = corpus[2].lower().split()

# Calculating bag of words
word_set = set(l_A).union(set(l_B)).union(set(l_C))

word_dict_A = dict.fromkeys(word_set, 0)
word_dict_B = dict.fromkeys(word_set, 0)
word_dict_C = dict.fromkeys(word_set, 0)

for word in l_A:
    word_dict_A[word] += 1
```

	and	another	cats	cheese	dogs	example	mouse	simple	with
0	1	0	1	0	0	1	1	1	1
1	1	1	1	0	1	1	0	1	1
2	1	1	0	1	0	1	1	1	1

that's what we get

In the case of the term frequency, the simplest choice is to use the raw count of a term in a string. For calculating tf for all terms, we must fill a dictionary as follows.

```
def compute_tf(word_dict, l):
    tf = {}
    sum_nk = len(l)
    for word, count in word_dict.items():
        tf[word] = count/sum_nk
    return tf


```

idf is a measure of how much information the token or word in our case, provides. For calculating idf we need fill dict too.

```
def compute_idf(strings_list):
    n = len(strings_list)
    idf = dict.fromkeys(strings_list[0].keys(), 0)
    for l in strings_list:
        for word, count in l.items():
            if count > 0:
                idf[word] += 1

    for word, v in idf.items():
```

Now, I remain you that tf-idf is the product of tf to idf. For our python example, tf-idf it dict with the corresponding products.

```
def compute_tf_idf(tf, idf):
    tf_idf = dict.fromkeys(tf.keys(), 0)
    for word, v in tf.items():
        tf_idf[word] = v * idf[word]
    return tf_idf

tf_idf_A = compute_tf_idf(tf_A, idf)
```

OK, now we have tf-idf weights for each word in our corpus. Below you can clearly see the difference between the original bag of words and the new bag of words with tf-idf weights. For example ‘dogs’, ‘cats’ and ‘mouse’ is important words, but word ‘and’ is not important, because this word is in all the strings and we can’t understand what is a string by the word ‘and’.

	and	another	cats	cheese	dogs	example	mouse	simple	with
0	0.0	0.000000	0.067578	0.000000	0.000000	0.0	0.067578	0.0	0.0
1	0.0	0.057924	0.057924	0.000000	0.156945	0.0	0.000000	0.0	0.0
2	0.0	0.057924	0.000000	0.156945	0.000000	0.0	0.057924	0.0	0.0

TF-IDF bag of words

	and	another	cats	cheese	dogs	example	mouse	simple	with
0	1	0	1	0	0	1	1	1	1
1	1	1	1	0	1	1	0	1	1
2	1	1	0	1	0	1	1	1	1

original bag of words

. . .

KMeans clustering with TF-IDF weights

Now, when we understand how TF-IDF work the time has come for almost real example of clustering with TF-IDF weights. For real life we can use scikit-learn implementation of TF-IDF and KMeans and I suggest you use implementations from scikit-learn or from another popular libraries or frameworks because it's reducing a number of potential errors in your code.

For this example, we must import TF-IDF and KMeans, added corpus of text for clustering and process its corpus.

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


all_textall_text  ==  """""""
 Google and Facebook are strangling the free press to death
Your 60-second guide to security stuff Google touted today
A Guide to Using Android Without Selling Your Soul to Googl
Review: Lenovo's Google Smart Display is pretty and intelli
Google Maps user spots mysterious object submerged off the
Android is better than IOS
In information retrieval, tf-idf or TFIDF, short for term f
is a numerical statistic that is intended to reflect
how important a word is to a document in a collection or co
It is often used as a weighting factor in searches of infor
text mining, and user modeling. The tf-idf value increases
to the number of times a word appears in the document
```

After that let's fit Tfidf and let's fit KMeans, with scikit-learn it's really easy.

```
tfidf_vectorizer = TfidfVectorizer(preprocessor=preprocessin
tfidf = tfidf_vectorizer.fit_transform(all_text)

kmeans = KMeans(n clusters=2).fit(tfidf)
```

Now we have learned KMeans model with k = 2 for clustering strings, it's easy, right?
For predicting, just use predict method as follows.

```
lines_for_predicting = ["tf and idf is awesome!", "some andr
kmeans.predict(tfidf_vectorizer.transform(lines_for_predicti

# array([0, 1], dtype=int32)
```

There we can see, that string '*tf and idf is awesome!*' and '*some androids is there*' from different clusters and it's right.

. . .

In addition, you can read Jupyter notebook with this examples.

Thanks for the reading, please leave a feedback. This can help me improve the quality of my future posts.

And don't forget to follow me on twitter.