Cosine Similarity and TFIDF

Taking a look at HackerNews posts

Robert R. · Follow
Published in Web Mining [IS688, Spring 2021] · 7 min read · May 3, 2021

12

Online forums have been a great resource for sharing all kinds of information for a long time. Somewhat recently, different online forums such as Reddit have democratized what is interesting giving the masses the ability to rank or rate a post via the upvote/downvote options or something similar.

Y-Combinator Hacker News https://news.ycombinator.com/

But what makes a post interesting? Is there a formula to creating a “front page” post? What do these posts have in common? How can I find similar posts to those that I frequently read? How can I find a specific post that may be related to a topic of my interest?

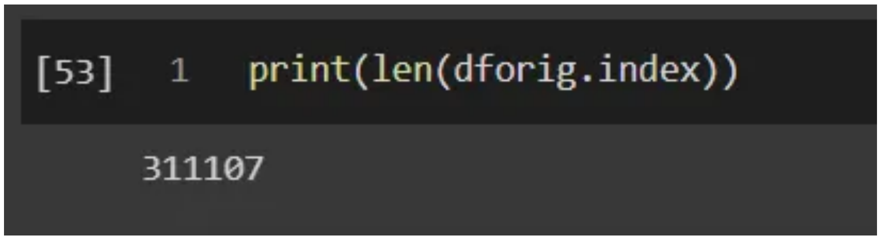
Today, we will focus on the forum HackerNews. This is a forum that is generally geared toward tech, science, and professional discussions based on interesting topics on the internet. This forum is also a place to “Show HN” what kinds of projects they are working on or “Ask HN” a question that a particular user would like some help answering. All of these are susceptible to the same democratization seen on Reddit, with one caveat, not everyone can downvote.

This analysis will be leveraging Pandas, Numpy, Sklearn to assist in our discovery.

```
import pandas as pd
import sklearn as sk
import numpy as np
import re
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
```

The Data

For this post I’ve found an existing dataset that logged every post, it’s poster, Post Type, upvotes, and comments for a period of time. For this post, and for the sake of processing time, I narrowed the analysis to posts in 2012, which gives us about 311,000 posts to work with.


```
[53]   print(len(dforig.index))

    311107
```

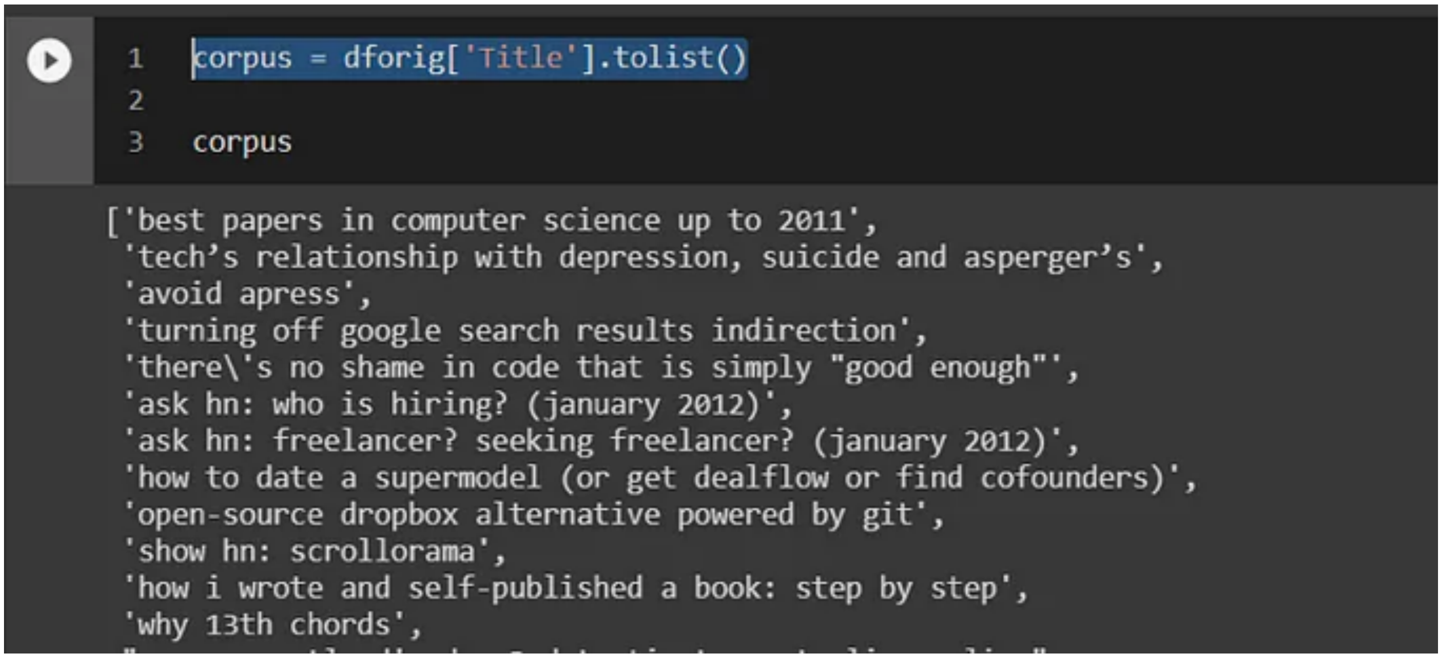
However, since memory is limited, I want to focus on the posts that seem to at least gain some traction, so I will focus on posts that only score above 100 points.

For some additional cleanup I removed all capital letters and dropped any null values from the data.

```
dforig[‘Title’] = dforig[‘Title’].str.lower()
dforig = dforig.dropna(subset=[‘Title’])
```

At this point I have my corpus which I will base the rest of my work off of.

```
corpus = dforig[‘Title’].tolist()
```



```
corpus = dforig['Title'].tolist()

corpus
```

```
['best papers in computer science up to 2011',
 'tech’s relationship with depression, suicide and asperger’s',
 'avoid apress',
 'turning off google search results indirection',
 'there\'s no shame in code that is simply "good enough"',
 'ask hn: who is hiring? (january 2012)',
 'ask hn: freelancer? seeking freelancer? (january 2012)',
 'how to date a supermodel (or get dealflow or find cofounders)',
 'open-source dropbox alternative powered by git',
 'show hn: scrollorama',
 'how i wrote and self-published a book: step by step',
 'why 13th chords',
```

Vectorization and Analysis

There are quite a few words in all these thousands of posts. So I want to condense them as much as possible. I initially chose to use a stemmer based upon some suggestions I read that would help reduce the amount of “repeat” words. For example, I could have Computer, Computing, Computation, and ideally a stemmer would take these words and combine them into one “Comput” word. It’s missing an ‘e’ but you get the idea. However, in practice I did not like the results as some of the stemming was too much. However, here is the code I used to create the stemmer.

```
from nltk.stem.snowball import SnowballStemmer
stemmer = SnowballStemmer(“english”)
def stemming_tokenizer(str_input):
words = re.sub(r”[^a-zA-Z]{2,}”, “ “, str_input).lower().split()
words = [stemmer.stem(word) for word in words]
return words
```

The above code block, with some changes, is courtesy of this article by Jonathan Soma.

The regex is in place to leave out numbers. What I noticed is that there were dozens of numerical strings that were nearly meaningless out of their context. On top of that, I am also using CountVectorizer's mindf, the documentation defines mindf as such:

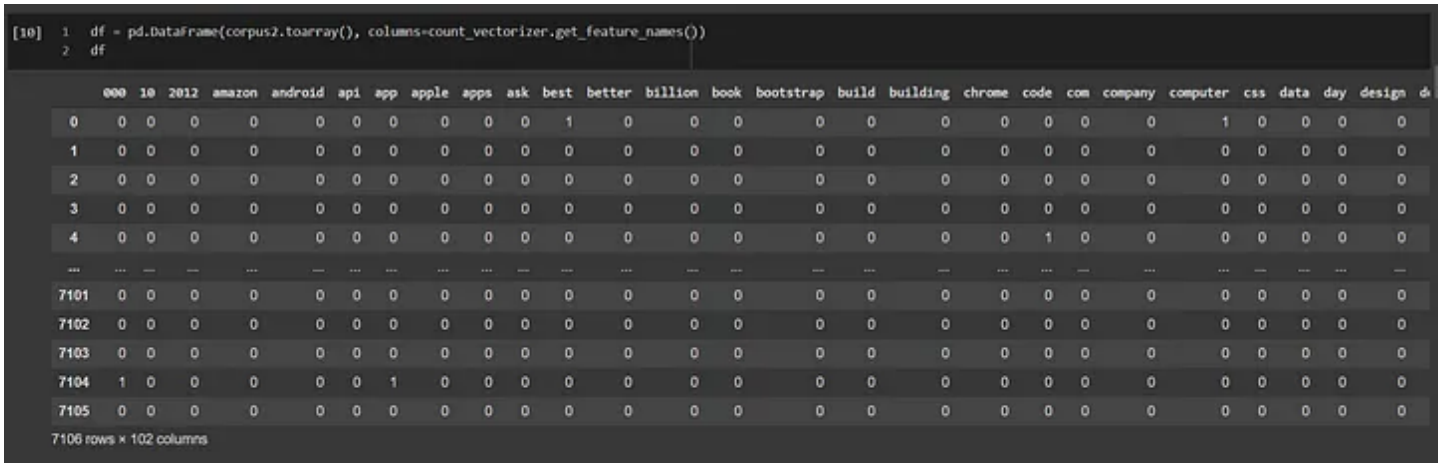
"When building the vocabulary ignore terms that have a document frequency strictly lower than the given threshold. This value is also called cut-off in the literature. If float, the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None."

```
count_vectorizer = CountVectorizer(stop_words='english',
min_df=0.005)

corpus2 = count_vectorizer.fit_transform(corpus)

print(count_vectorizer.get_feature_names())
```

Our result (strangely, with some numbers still in it):



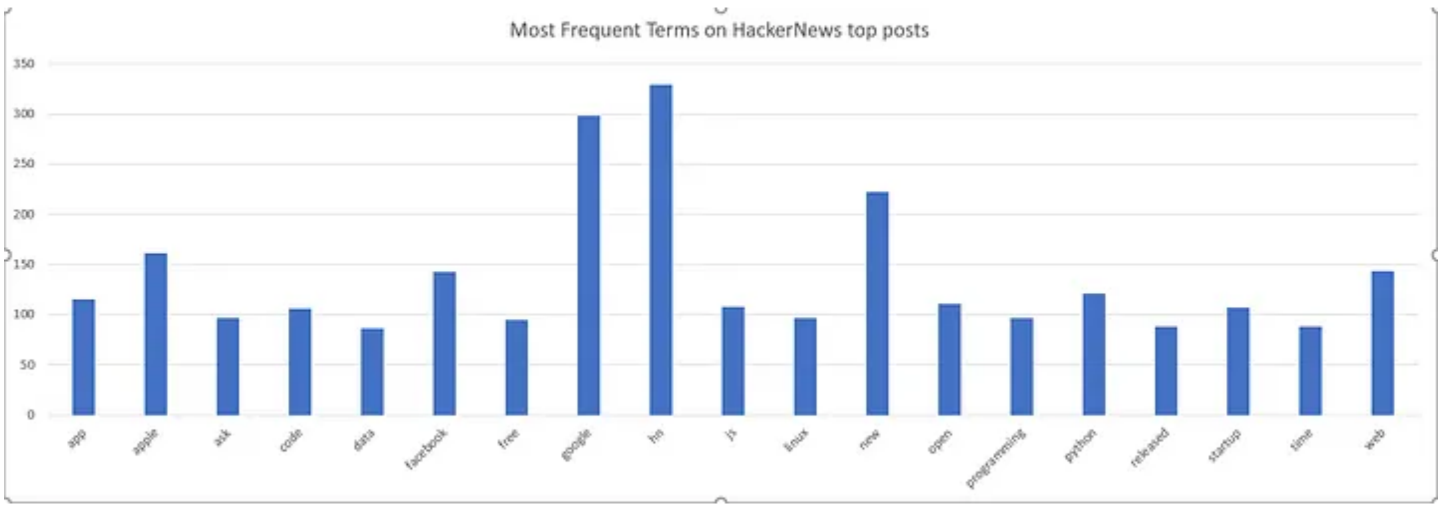
```
[10]  df = pd.DataFrame(corpus2.toarray(), columns=count_vectorizer.get_feature_names())
      df
```

```
        000  10  2012  amazon  android  api  app  apple  apps  ask  best  better  billion  book  bootstrap  build  building  chrome  code  com  company  computer  css  data  day  design
     0    0   0     0       0        0    0    0      0     0    0     1       0        0     0          0      0         0       0     0    0        0         1    0     0    0       0
     1    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
     2    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
     3    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
     4    0   0     0       0        0    0    0      0     0    0     0       0        0     0          0      0         0       0     1    0        0         0    0     0    0       0
   ...  ... ...   ...     ...      ...  ...  ...    ...   ...  ...   ...     ...      ...   ...        ...    ...       ...     ...   ...  ...      ...       ...  ...   ...  ...     ...
  7101    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
  7102    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
  7103    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
  7104    1   0     0       0        0    0    1      0     0    0     0       0        0     0          0      0         0       0     0    0        0         0    0     0    0       0
  7105    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
7106 rows × 102 columns
```

I stopped here a second to see what terms were actually the most frequent among top posts. To do that, I had to set the options so they wouldn't abridge the list:

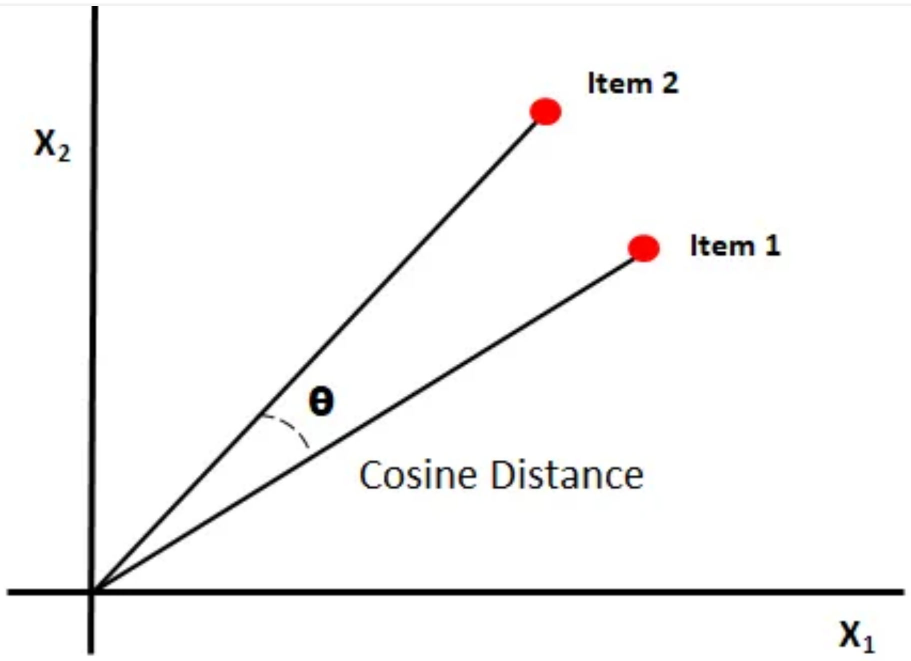
```
pd.set_option('display.max_rows', 134)
df_sum = df.sum(axis=0)
```

Let's see this in a graph (Filtered for words appearing less than 85 times).



Cosine Similarity

Cosine similarity at it's most basic definition is measuring the similarity between two documents, regardless of the size of each document.

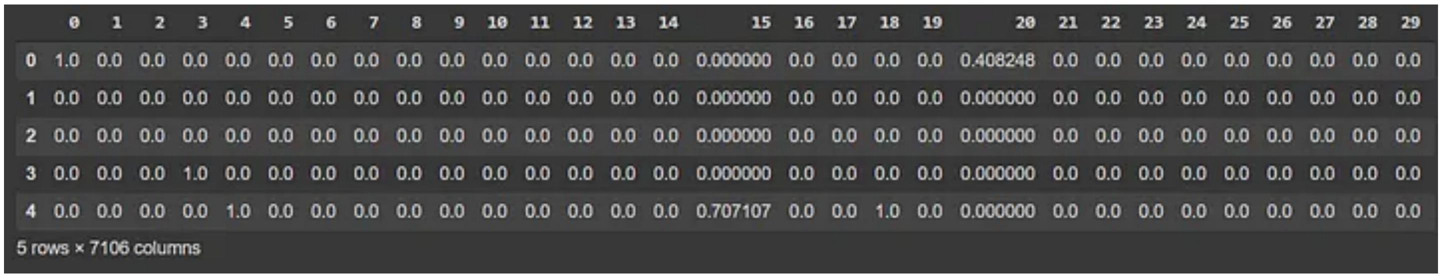


Cosine Similarity

Basically, this could be very useful for taking a particular document, or in our case a post title, and finding those that are similar. In this case, let's try and get a direct referral to another similar post based on each title.

Scikit makes this easy using the following code:

```
from sklearn.metrics.pairwise import cosine_similarity
df2 = pd.DataFrame(cosine_similarity(df, dense_output=True))
df2.head()
```



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	1.0	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	0.000000	0.0	0.0	0.0	0.0	0.408248	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	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	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	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	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.707107	0.0	0.0	1.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 7106 columns

Cosine Similarity dataframe

From here, we needed to do two things.

1. Zero out the 1's for documents that are similar to themselves, this doesn't help us.
2. Find the most similar corresponding document for every document.

WARNING: In my case, this was VERY memory intensive.

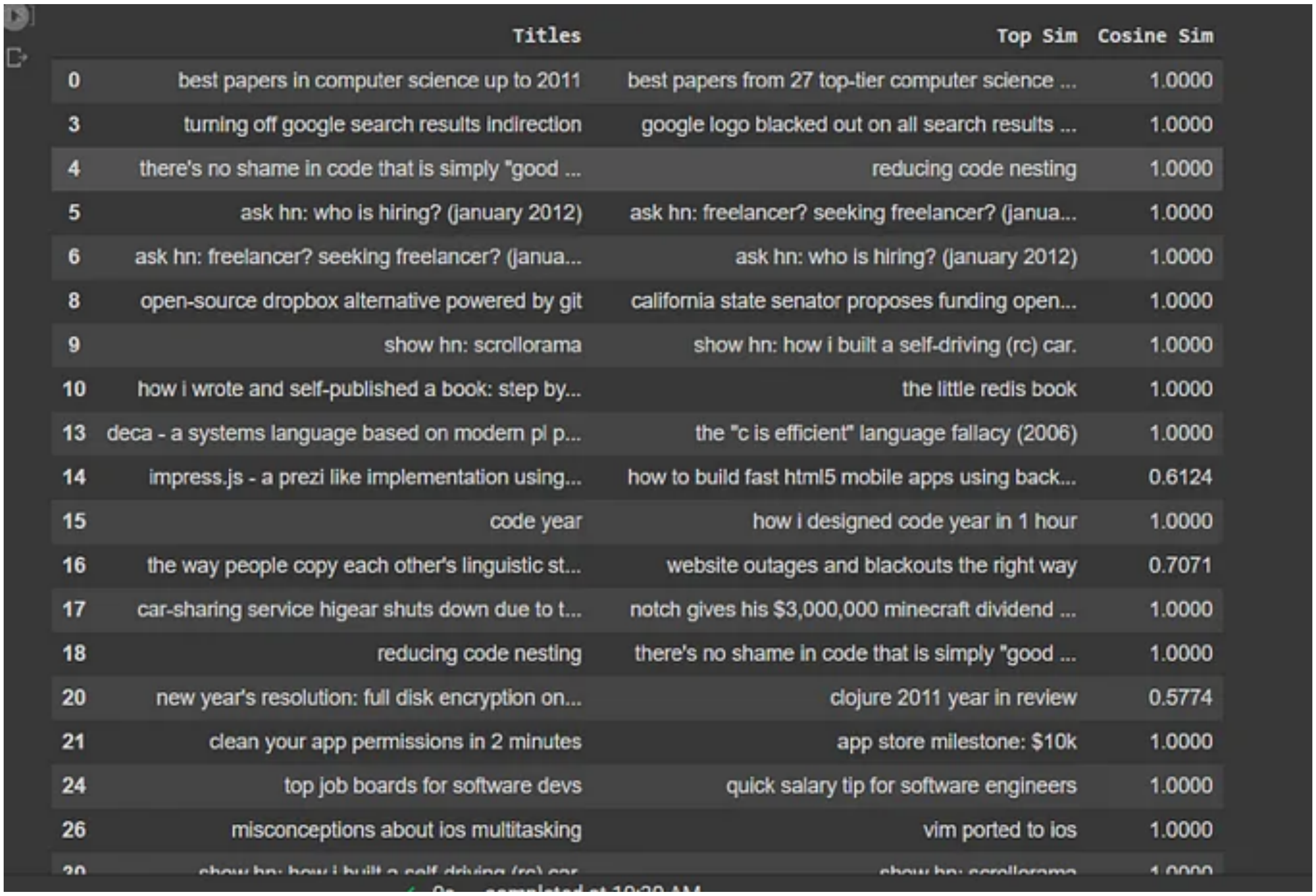
I'll skip a lot of the code for this process, but to get to this point I encourage you to watch Mr. Fugu's video as he breaks down some of the complexity that is involved with getting to this point.

```
titles_df=pd.DataFrame(corpus,columns=['Titles'])

cos_sim_df=pd.concat([titles_df,sim_posts,sim_value],axis=1)

cos_sim_df = cos_sim_df[cos_sim_df['Cosine Sim'] > 0]

cos_sim_df
```



	Titles	Top Sim	Cosine Sim
0	best papers in computer science up to 2011	best papers from 27 top-tier computer science ...	1.0000
3	turning off google search results indirection	google logo blacked out on all search results ...	1.0000
4	there's no shame in code that is simply "good ...	reducing code nesting	1.0000
5	ask hn: who is hiring? (january 2012)	ask hn: freelancer? seeking freelancer? (janua...	1.0000
6	ask hn: freelancer? seeking freelancer? (janua...	ask hn: who is hiring? (january 2012)	1.0000
8	open-source dropbox alternative powered by git	california state senator proposes funding open...	1.0000
9	show hn: scrollorama	show hn: how i built a self-driving (rc) car.	1.0000
10	how i wrote and self-published a book: step by...	the little redis book	1.0000
13	deca - a systems language based on modern pl p...	the "c is efficient" language fallacy (2006)	1.0000
14	impress.js - a prezi like implementation using...	how to build fast html5 mobile apps using back...	0.6124
15	code year	how i designed code year in 1 hour	1.0000
16	the way people copy each other's linguistic st...	website outages and blackouts the right way	0.7071
17	car-sharing service higear shuts down due to t...	notch gives his $3,000,000 minecraft dividend ...	1.0000
18	reducing code nesting	there's no shame in code that is simply "good ...	1.0000
20	new year's resolution: full disk encryption on...	clojure 2011 year in review	0.5774
21	clean your app permissions in 2 minutes	app store milestone: $10k	1.0000
24	top job boards for software devs	quick salary tip for software engineers	1.0000
26	misconceptions about ios multitasking	vim ported to ios	1.0000
30	show hn: how i built a self-driving (rc) car	show hn: scrollorama	1.0000

Result of matching one post with another

While we see that there are a number of posts with greater than zero cosine similarity, so it appears that there might not be a magic formula for a top HN post outside of maybe topic selection. Also, many of the “similar” posts are fundamentally different, despite them having similar subject matter. Using this code, we can certainly leverage the data to determine subject matter that may be of interest to the reader. For example, if the reader is looking for something like the post regarding “Best Papers in Computer science up to 2011” we can reasonably assume they will also like the post regarding the “Best papers from 27 top-tier computer science conferences” or someone interested in “Top Job boards for software devs” would also be interested in “salary tips for software engineers.

But let’s take it one step further. Because while finding similar posts in some use cases can be great, maybe I don’t have a post I like and just a topic. Let’s consider maybe I need some help writing this post so I want to find good articles on “Python web mining”. I’ll start with working through TFIDF. The TF stands for Term Frequency, this is exactly as it sounds, we’re looking at how often a term shows up. IDF stands for inverse document frequency, this process gives for weight to words that appear in less documents. So for example, if I didn’t have stop words removed already, it would make certain words like “The” and “and” (which probably are frequently represented in each document) weigh less. Sckit makes this easy for us, see below.

```
tfidf_vectorizer = TfidfVectorizer(stop_words=’english’,
use_idf=True)

corpus2 = tfidf_vectorizer.fit_transform(corpus)

df_tfidf = pd.DataFrame(corpus2.toarray(),
columns=tfidf_vectorizer.get_feature_names())

df_tfidf
```

Next I’ll code in the search terms we want to use by basically taking the terms and parsing out their respective columns from the above dataframe. Remember, we’re curious about “Python Web Mining”. But in this result I also only want to keep what’s useful to me, so I go ahead and filter out all the results that will return a 0, ensuring I only get relevant results. To make life easy, I sort by descending value, showing me the best result at the top.

```
search_df = pd.DataFrame([df_tfidf[‘python’], df_tfidf[‘web’],
df_tfidf[‘mining’], df_tfidf[‘python’] + df_tfidf[‘web’] +
df_tfidf[‘mining’]], index=[“Python”, “Web”, “Mining”, “Python + Web
+ Mining”]).T

search_df = search_df[search_df[‘Python + Web + Mining’] > 0]

search_df = search_df[search_df[‘Mining’] > 0]

search_df

test = search_df.sort_values([‘Python + Web + Mining’], ascending=
[False])

test
```

Document 5726 is my best result

As you can see, this returned document 5726. What is that document exactly? Returning to our original corpus we can see it is…

Bingo! This looks like it could absolutely be of some use.

Limitations and Issues

- I was not able to determine some of my original questions. For example, what makes a top post? From what I could tell there are some popular topics, but it may not be as formulaic as I had though.
- I was having some trouble with the stemmer. It would often stem words down to a single letter. This was not helpful.
- It seems with many of the cosine similarities, we had a '1' returned which would indicate they are VERY similar. But looking at the titles, this was not always the case.

References

scikit-learn

"We use scikit-learn to support leading-edge basic research [...]" "I think it's the most well-designed ML package I've...

scikit-learn.org

TF-IDF

Text analysis has a few parts. We are going to use bag of words analysis, which just treats a sentence like a bag of...

jonathansoma.com

Stemmers

Stemmers remove morphological affixes from words, leaving only the word stem.

www.nltk.org

Hacker News

All posts from Y Combinator's social news website from 2006 to late 2017

www.kaggle.com

Is688spr21a03