KE5205 Text Mining And Data Mining

Text Mining on Hacker News

Team Members:
Tan Ren Jie (A0094567N)

Lim Hui Juin (A0178202L)

Goh Yee Phing (A0178221J)

Cai Yutian (A0178545R)

Shahril Mustafa (A0178552W)

under the supervision of
Fan Zhen Zhen,
Cai Yuhao
&
Rajaraman Kanagasabai

Master of Technology, Institute of System Science, NUS

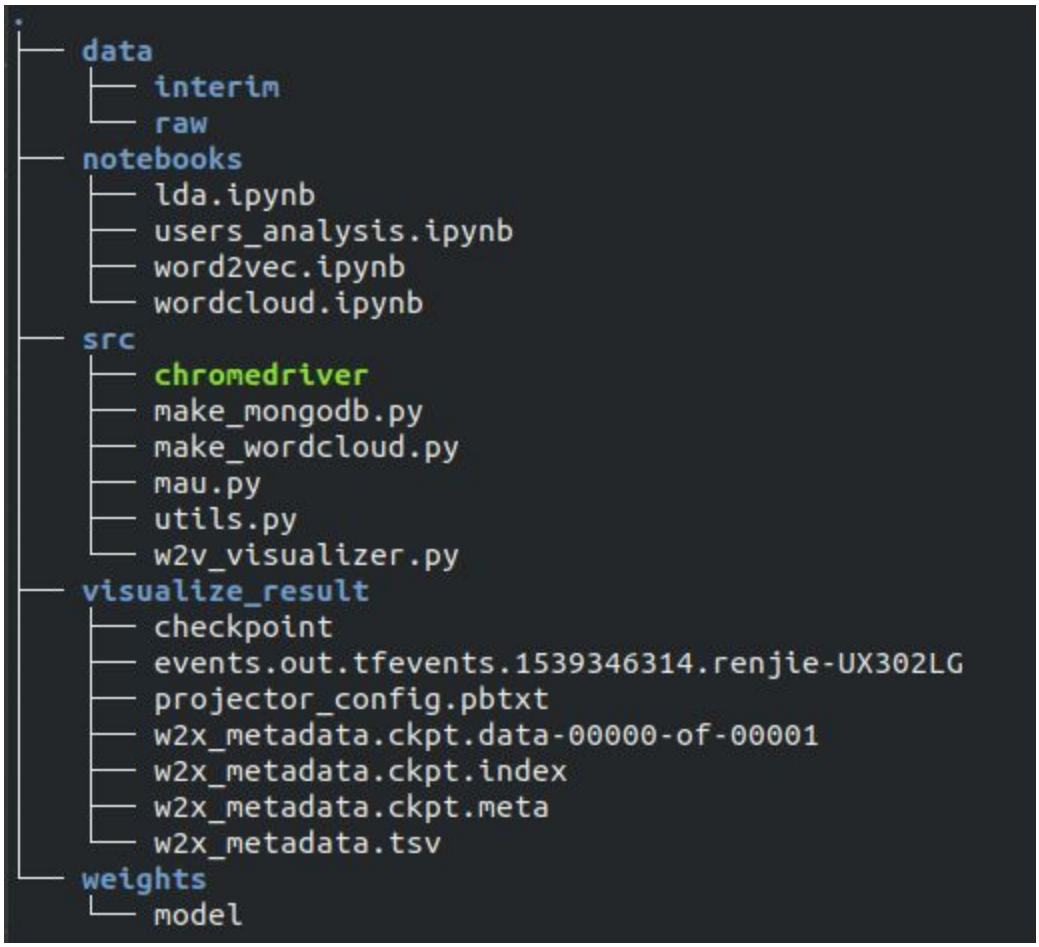
04 Oct 2018

Table of Contents

Links to Additional Code and Dataset

Source code repository: https://github.com/notha99y/TextMiningHackerNews

Project Directory:



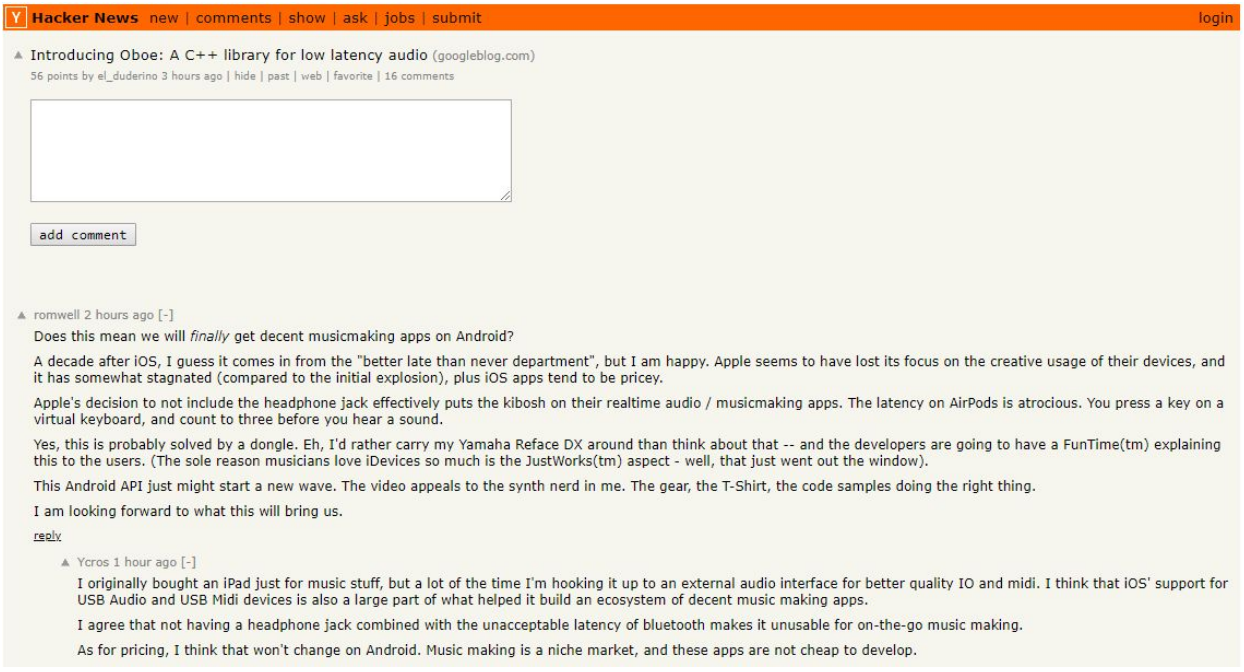
Data Source: https://www.kaggle.com/hacker-news/hacker-news

Introduction

Hacker News is a social news website ran by investment fund and startup incubator, Y Combinator. It focuses on computer science and entrepreneurship and aims to create an active community for constructive and intelligent discussion, using a community-based voting system that ranks posts and comments.[1]

In this report, we shall assume the role of management of Y Combinator to see how our social news website is doing. We will use the various techniques taught in the course to text mine and find answers or solutions to the following business questions.

Screenshot of the Hacker News site, showing the title posts



Screenshot of the comments made in response to a title post, this comments page is shown only after the user clicks on a title post

Business Questions

1. What are the trending topics? Can they be used as part of market research for seed acceleration purposes?
2. What should writers focus on?
3. Recent studies have found that many forums tend to be dominated by a very small fraction of users. Is this true of Hacker News?
4. Hacker News has received complaints that the site is biased towards Y Combinator startups. Do the data support this?

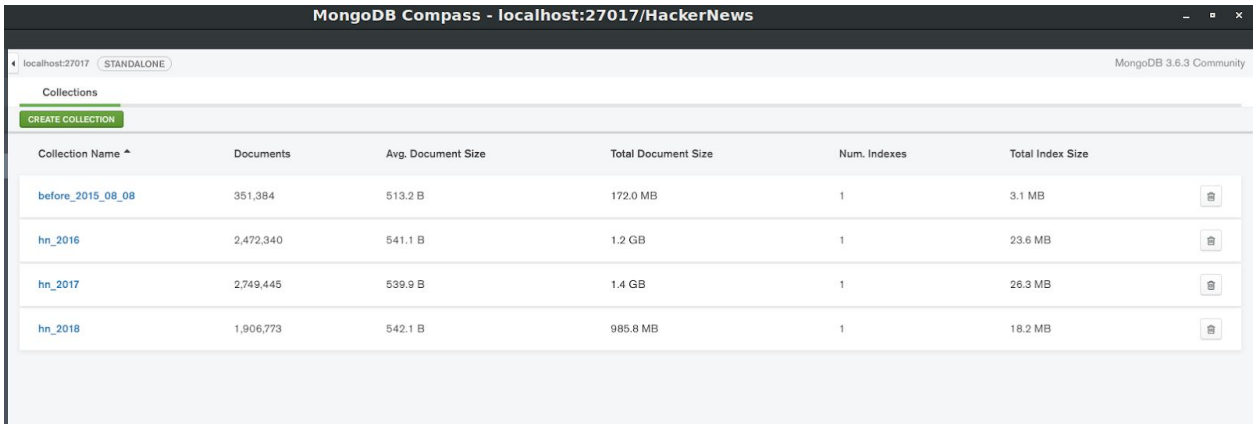
Technologies used

Scripting Language: Python
Relevant libraries: NLTK, wordcloud, gensim, pandas, matplotlib, seaborn
Database: MongoDB
Visualization: Tensorboard
Others: Selenium, Jupyter notebooks, BigQuery, Google Colaboratory

About the Dataset

The dataset we will be using is from an open source dataset containing all the posts from Hacker News from **2006 to September 2018**. The data can be access using Google's BigQuery [2]. For the scope of the our project, we will be focusing on **Year 2016 - 2018** where we will be applying the various text mining techniques learnt in the course to answer the business questions. For 2018 itself, there are approximately **1.9 million rows** of data for analysis, with text data from **113169 unique users**.

The data is exported in full to a JSON file for each year. Each JSON is on average about **1.2 GB** and we imported it to a MongoDB where we could query easily without having to load the entire dataset into memory. The mongo cursor object acts as a generator, reading the data from the disk on-the-fly, which allows us to avoid the lack of memory error. Below is the screenshot of our database viewed from MongoDB Compass[3], a MongoDB Graphic User Interface (GUI).



MongoDB Compass of our HackerNews Collections

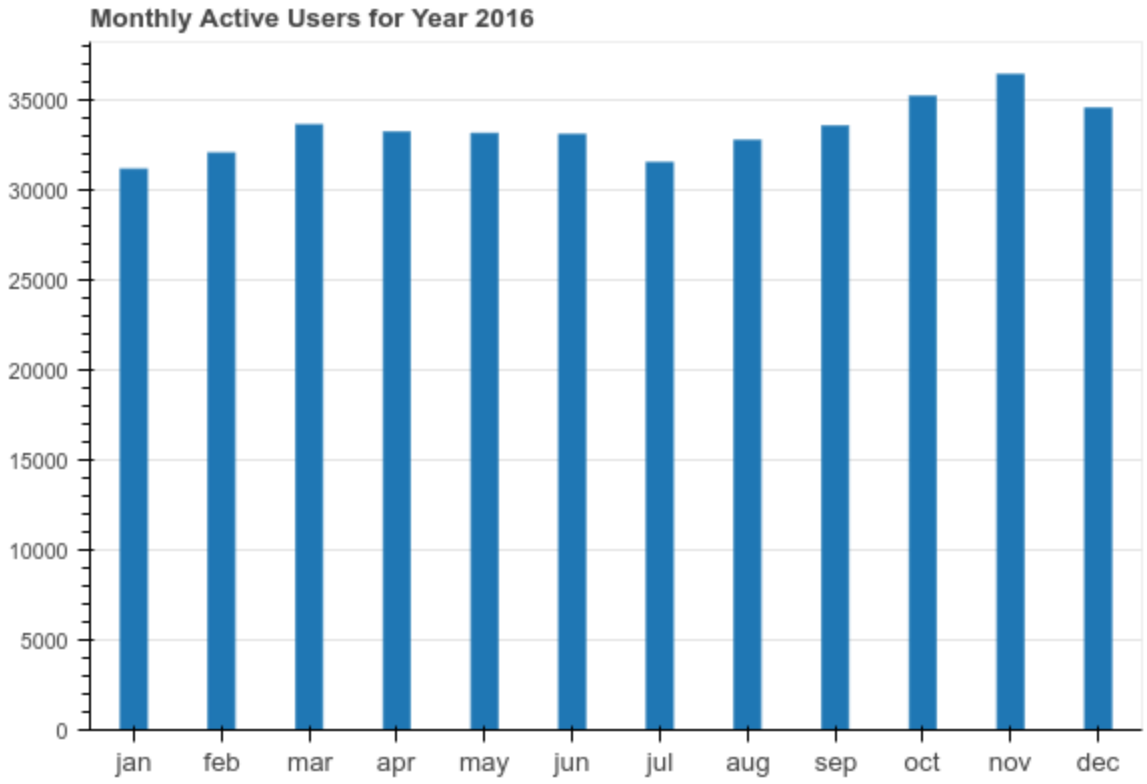
Data Dictionary

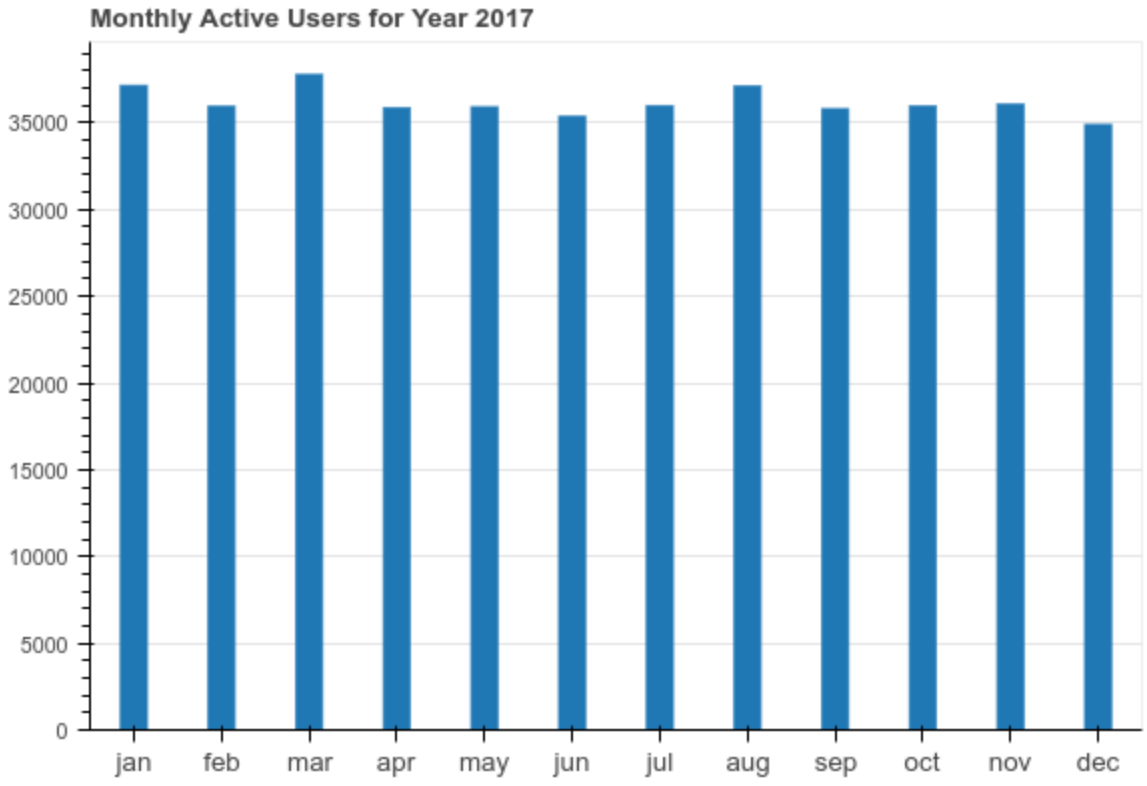
The dataset consist of the following columns [4]:

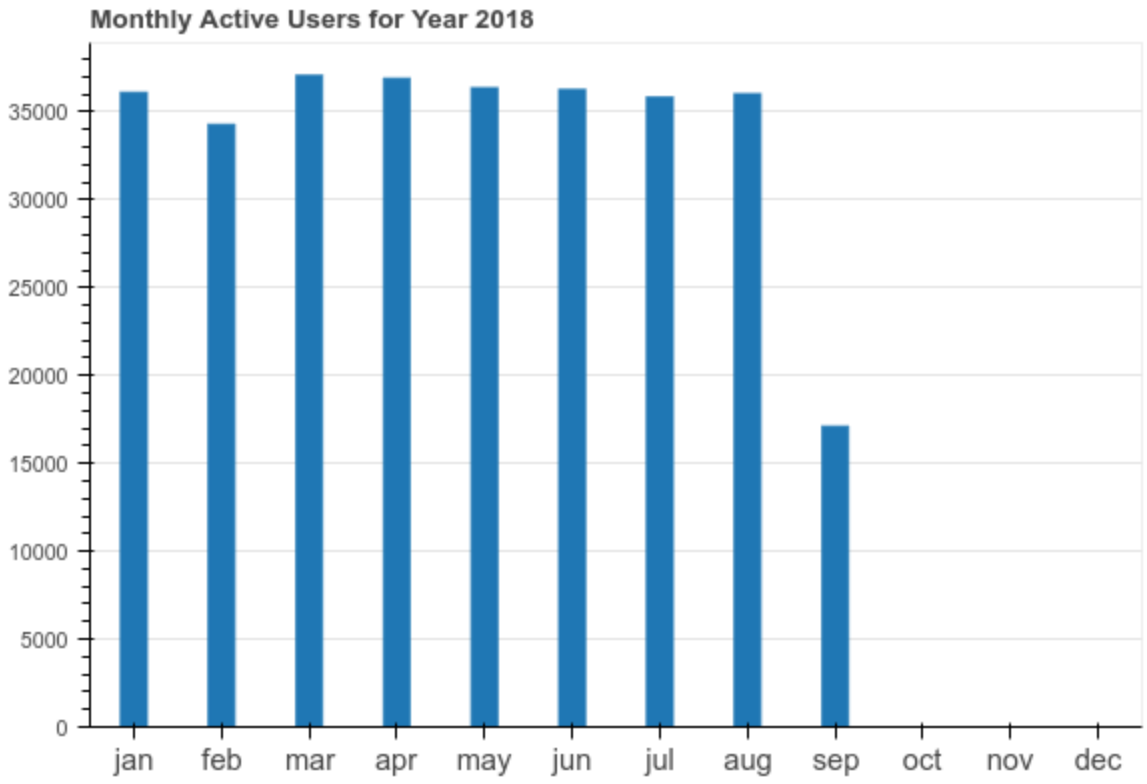
Title	Data Type	Description
by	str	The username of the item's author
score	int	Story score
time	int	Unix time
time stamp	date time	Timestamp for the unix time
title	str	Story title
type	str	Type of details (comment, comment_ranking, poll, story, job, pollopt)
url	str	Story url
text	str	Story or comment text
parent	int	Parent comment ID
deleted	boolean	Is deleted?
dead	boolean	Is dead?
descendants	int	Number of story or poll descendants
id	int	The item's unique id
ranking	int	Comment ranking

Monthly Active Users

Immediately, with the data loaded into our database, we could perform a simply query to get the number of Monthly Active Users (MAU) just to gauge the size and variety of users engaging with the social news website. These might aid in our decisions in choosing certain texting techniques over the other.




Monthly Active Users for Year 2017
35000
30000
25000
20000
15000
10000
5000
0
jan
feb
mar
apr
may
jun
jul
aug
sep
oct
nov
dec


Monthly Active Users for Year 2018
35000
30000
25000
20000
15000
10000
5000
0
jan
feb
mar
apr
may
jun
jul
aug
sep
oct
nov
dec

From the charts above, we noted the increase of MAU from 2016 to 2017. Ever since, the MAU have been consistently slightly about **35,000**. It is important to note that the data for Sep 2018 is incomplete as we extracted the dataset on 11th Sept.

Methodology

Data Exploration

The majority of the text data are comments by the forum users to links posted on this forum (under the 'text' column). The rest of the text data are titles of the posts themselves. Most texts contain links, html encodings, escape characters in them, which need to be cleaned before processing. A sample text is shown below:

```
entries_text[0]
```

```
'BTW: WebVR is now enabled by default in Firefox Nightly<p><a href="https:&#x2F;&#x2F;www.r
eddit.com&#x2F;r&#x2F;WebVR&#x2F;comments&#x2F;3zmclh&#x2F;webvr_enabled_by_default_in_fire
fox_nightly&#x2F;" rel="nofollow">https:&#x2F;&#x2F;www.reddit.com&#x2F;r&#x2F;WebVR&#x2F;c
omments&#x2F;3zmclh&#x2F;webvr_enabled...</a>'
```

Character encoding

Most of the text in the comments contain html encoding which we need to clean, either by dropping them or converting them to their intended display character. We used a python html package method, unescape, to clean these encodings [5].

Text Cleaning

Before working on the data, we performed some text cleaning steps to allow further processing by the NLTK libraries, as well as obtain more accurate data.

Removing Non-ASCII characters

Since the comments on the website may have been entered in any language or use symbols, the first step was to remove all non-ASCII characters. This allowed us to focus on text written using alphanumeric characters only.

```
text
```

```
'â€¢ the language is easy, very attractive for non-CS people<p>â€¢ the language is consistent (everything is an object, or a po
inter to an object rather, you can only pass by pointer, scope and namespace that make sense all the time, etc...)<p>â€¢ you ca
n learn it gradually, you can start using it even if you know only 10% of the language<p>â€¢ Amazing documentation. The officia
l tutorial is easy to read, and by the time you are through with it you are fairly proficient with python<p>â€¢ error message t
hat tells you exactly what the error is and where it is happening. A lot of other languages do that now, but that wasn&#x27;t t
he case 25 years ago.<p>â€¢ inline help &#x2F; magical docstring (`a=4 ; help(a)`)<p>â€¢ The REPL, especially being able to get
into the REPL with your apps (`python -i`)<p>â€¢ batteries, again 25 years ago, no language came with such a huge standard libr
ary<p>â€¢ PEP 8, the grandfather to gofmt and rustfmt. Makes a big difference when working on a team<p>â€¢ the Zen (`import thi
s`)<p>And of course as the language grew in popularity:<p>â€¢ the ecosystem (pypi.org)<p>â€¢ the amount of resources, from book
s to website, the answer to every questions you can think of in SO, very active and helpful mailing list&#x2F;usenet group, act
ive and helpful &#x2F;r&#x2F;Python'
```

```
remove_non_ascii(text)
```

```
' the language is easy, very attractive for non-CS people<p> the language is consistent (everything is an object, or a pointer
to an object rather, you can only pass by pointer, scope and namespace that make sense all the time, etc...)<p> you can learn i
t gradually, you can start using it even if you know only 10% of the language<p> Amazing documentation. The official tutorial i
s easy to read, and by the time you are through with it you are fairly proficient with python<p> error message that tells you e
xactly what the error is and where it is happening. A lot of other languages do that now, but that wasn&#x27;t the case 25 year
s ago.<p> inline help &#x2F; magical docstring (`a=4 ; help(a)`)<p> The REPL, especially being able to get into the REPL with y
our apps (`python -i`)<p> batteries, again 25 years ago, no language came with such a huge standard library<p> PEP 8, the grand
father to gofmt and rustfmt. Makes a big difference when working on a team<p> the Zen (`import this`)<p>And of course as the la
nguage grew in popularity:<p> the ecosystem (pypi.org)<p> the amount of resources, from books to website, the answer to every q
uestions you can think of in SO, very active and helpful mailing list&#x2F;usenet group, active and helpful &#x2F;r&#x2F;Pytho
n'
```

Example of removing â€¢ characters from text

Removing HTML tags

After using html.unescape to convert the Unicode encoding in the raw dataset, it still contained HTML tags to denote “bold”, “italics”, “tables”, or “links” in the text (e.g. , <td>, <a href=>). Since these tags are part of the markup and should not contribute to the processed word bank, we removed these tags using BeautifulSoup.

```
text
```

```
' the language is easy, very attractive for non-CS people<p> the language is consistent (everything is an object, or a pointer
to an object rather, you can only pass by pointer, scope and namespace that make sense all the time, etc...)<p> you can learn i
t gradually, you can start using it even if you know only 10% of the language<p> Amazing documentation. The official tutorial i
s easy to read, and by the time you are through with it you are fairly proficient with python<p> error message that tells you e
xactly what the error is and where it is happening. A lot of other languages do that now, but that wasn&#x27;t the case 25 year
s ago.<p> inline help &#x2F; magical docstring (`a=4 ; help(a)`)<p> The REPL, especially being able to get into the REPL with y
our apps (`python -i`)<p> batteries, again 25 years ago, no language came with such a huge standard library<p> PEP 8, the grand
father to gofmt and rustfmt. Makes a big difference when working on a team<p> the Zen (`import this`)<p>And of course as the la
nguage grew in popularity:<p> the ecosystem (pypi.org)<p> the amount of resources, from books to website, the answer to every q
uestions you can think of in SO, very active and helpful mailing list&#x2F;usenet group, active and helpful &#x2F;r&#x2F;Pytho
n'
```

```
BeautifulSoup(html.unescape(text), "lxml").get_text()
```

```
"the language is easy, very attractive for non-CS people the language is consistent (everything is an object, or a pointer to a
n object rather, you can only pass by pointer, scope and namespace that make sense all the time, etc...) you can learn it gradu
ally, you can start using it even if you know only 10% of the language Amazing documentation. The official tutorial is easy to
read, and by the time you are through with it you are fairly proficient with python error message that tells you exactly what t
he error is and where it is happening. A lot of other languages do that now, but that wasn't the case 25 years ago. inline help
/ magical docstring (`a=4 ; help(a)`) The REPL, especially being able to get into the REPL with your apps (`python -i`) batteri
es, again 25 years ago, no language came with such a huge standard library PEP 8, the grandfather to gofmt and rustfmt. Makes a
big difference when working on a team the Zen (`import this`)And of course as the language grew in popularity: the ecosystem (p
ypi.org) the amount of resources, from books to website, the answer to every questions you can think of in SO, very active and
helpful mailing list/usenet group, active and helpful /r/Python"
```

Removing HTML encoding and tags with unescape and BeautifulSoup

Other techniques such as:

1. Decontracting words (e.g. replacing “won’t” with “will not”)
2. Removing punctuations using regular expressions

Will be elaborated more in the later sections.

If the question requires other processing such as stopwords removal, lemmatization, tokenization for sentences and words, they are done after this cleaning step.

Wordcloud

A wordcloud representation of the data will allow us to see which terms are being used most often in discussions in the HackerNews forums. It produces an intuitive diagram of terms, which gives a rough idea of the discussion topics by looking at the words which are used most regularly in user posts.

Preparations to Process Large Datasets

The “title” and “text” fields for the year 2018 were extracted from MongoDB into a single .csv file. During the processing of this file, the “title” field was able to be processed as a list of size 250,478. However, the “text” field had about 17 million non-empty records and required more than 16GB of memory for processing. Therefore, we’ve divided the processing of the “text” field into batches, organized by month. The results were then added to a master document and saved as a .csv file to reduce repeated processing (filename: interim/texts_clean.csv).

Preprocessing

Tokenization

To perform further analysis of our text data, we first tokenize the raw data using the word_tokenize function from the nltk.corpus library. This gives us a list of comma separated words and punctuation.

```
print(word_tokenize(text))
```

```
['the', 'language', 'is', 'easy', ',', 'very', 'attractive', 'for', 'non-CS', 'people', 'the', 'language', 'is', 'consistent',
'(', 'everything', 'is', 'an', 'object', ',', 'or', 'a', 'pointer', 'to', 'an', 'object', 'rather', ',', 'you', 'can', 'only',
'pass', 'by', 'pointer', ',', 'scope', 'and', 'namespace', 'that', 'make', 'sense', 'all', 'the', 'time', ',', 'etc', '...',
')', 'you', 'can', 'learn', 'it', 'gradually', ',', 'you', 'can', 'start', 'using', 'it', 'even', 'if', 'you', 'know', 'only',
'10', '%', 'of', 'the', 'language', 'Amazing', 'documentation', '.', 'The', 'official', 'tutorial', 'is', 'easy', 'to', 'read',
',', 'and', 'by', 'the', 'time', 'you', 'are', 'through', 'with', 'it', 'you', 'are', 'fairly', 'proficient', 'with', 'python',
'error', 'message', 'that', 'tells', 'you', 'exactly', 'what', 'the', 'error', 'is', 'and', 'where', 'it', 'is', 'happening',
'.', 'A', 'lot', 'of', 'other', 'languages', 'do', 'that', 'now', ',', 'but', 'that', 'was', "n't", 'the', 'case', '25', 'year
s', 'ago', '.', 'inline', 'help', '/', 'magical', 'docstring', '(', '``', 'a=4', ';', 'help', '(', 'a', ')', '`', ')', 'The', 'R
EPL', ',', 'especially', 'being', 'able', 'to', 'get', 'into', 'the', 'REPL', 'with', 'your', 'apps', '(', '`', 'python', '-i',
'`', ')', 'batteries', ',', 'again', '25', 'years', 'ago', ',', 'no', 'language', 'came', 'with', 'such', 'a', 'huge', 'standar
d', 'library', 'PEP', '8', ',', 'the', 'grandfather', 'to', 'gofmt', 'and', 'rustfmt', '.', 'Makes', 'a', 'big', 'difference',
'when', 'working', 'on', 'a', 'team', 'the', 'Zen', '(', '``', 'import', 'this', '`', ')', 'And', 'of', 'course', 'as', 'the',
'language', 'grew', 'in', 'popularity', ':', 'the', 'ecosystem', '(', 'pypi.org', ')', 'the', 'amount', 'of', 'resources', ',',
'from', 'books', 'to', 'website', ',', 'the', 'answer', 'to', 'every', 'questions', 'you', 'can', 'think', 'of', 'in', 'SO',
',', 'very', 'active', 'and', 'helpful', 'mailing', 'list/usenet', 'group', ',', 'active', 'and', 'helpful', '/r/Python']
```

Word_tokenize gives us a list of comma-separated tokens

POS Tagging

To improve the accuracy of the lemmatization process, we used the pos_tag function from the nltk library to tag all of the tokens obtained in the above step. Since we are only interested in the topics being discussed, we removed all tokens that do not have the tags “NOUN”, “VERB”, “ADJ”, or “ADV”.

```
print(pos_tagged)
```

```
[('language', 'NOUN'), ('is', 'VERB'), ('easy', 'ADJ'), ('very', 'ADV'), ('attractive', 'ADJ'), ('non-CS', 'ADJ'), ('people',
'NOUN'), ('language', 'NOUN'), ('is', 'VERB'), ('consistent', 'ADJ'), ('everything', 'NOUN'), ('is', 'VERB'), ('object', 'NOU
N'), ('pointer', 'NOUN'), ('object', 'NOUN'), ('rather', 'ADV'), ('can', 'VERB'), ('only', 'ADV'), ('pass', 'VERB'), ('pointe
r', 'NOUN'), ('scope', 'NOUN'), ('namespace', 'NOUN'), ('make', 'VERB'), ('sense', 'NOUN'), ('time', 'NOUN'), ('can', 'VERB'),
('learn', 'VERB'), ('gradually', 'ADV'), ('can', 'VERB'), ('start', 'VERB'), ('using', 'VERB'), ('even', 'ADV'), ('know', 'VER
B'), ('only', 'ADV'), ('%', 'NOUN'), ('language', 'NOUN'), ('Amazing', 'NOUN'), ('documentation', 'NOUN'), ('official', 'ADJ'),
('tutorial', 'NOUN'), ('is', 'VERB'), ('easy', 'ADJ'), ('read', 'VERB'), ('time', 'NOUN'), ('are', 'VERB'), ('are', 'VERB'),
('fairly', 'ADV'), ('proficient', 'ADJ'), ('python', 'ADJ'), ('error', 'NOUN'), ('message', 'NOUN'), ('tells', 'VERB'), ('exact
ly', 'ADV'), ('error', 'NOUN'), ('is', 'VERB'), ('where', 'ADV'), ('is', 'VERB'), ('happening', 'VERB'), ('lot', 'NOUN'), ('oth
er', 'ADJ'), ('languages', 'NOUN'), ('do', 'VERB'), ('now', 'ADV'), ('was', 'VERB'), ("n't", 'ADV'), ('case', 'NOUN'), ('year
s', 'NOUN'), ('ago', 'ADV'), ('inline', 'VERB'), ('help', 'NOUN'), ('/', 'VERB'), ('magical', 'ADJ'), ('docstring', 'NOUN'),
('a=4', 'ADV'), ('help', 'NOUN'), ('REPL', 'NOUN'), ('especially', 'ADV'), ('being', 'VERB'), ('able', 'ADJ'), ('get', 'VERB'),
('REPL', 'NOUN'), ('apps', 'NOUN'), ('python', 'ADV'), ('-i', 'NOUN'), ('batteries', 'NOUN'), ('again', 'ADV'), ('years', 'NOU
N'), ('ago', 'ADV'), ('language', 'NOUN'), ('came', 'VERB'), ('huge', 'ADJ'), ('standard', 'NOUN'), ('library', 'NOUN'), ('PE
P', 'NOUN'), ('grandfather', 'NOUN'), ('gofmt', 'VERB'), ('rustfmt', 'VERB'), ('Makes', 'NOUN'), ('big', 'ADJ'), ('difference',
'NOUN'), ('when', 'ADV'), ('working', 'VERB'), ('team', 'NOUN'), ('Zen', 'NOUN'), ('import', 'NOUN'), ('course', 'NOUN'), ('lan
guage', 'NOUN'), ('grew', 'VERB'), ('popularity', 'NOUN'), ('ecosystem', 'NOUN'), ('pypi.org', 'NOUN'), ('amount', 'NOUN'), ('r
esources', 'NOUN'), ('books', 'NOUN'), ('website', 'VERB'), ('answer', 'NOUN'), ('questions', 'NOUN'), ('can', 'VERB'), ('thin
k', 'VERB'), ('SO', 'NOUN'), ('very', 'ADV'), ('active', 'ADJ'), ('helpful', 'ADJ'), ('mailing', 'NOUN'), ('list/usenet', 'NOU
N'), ('group', 'NOUN'), ('active', 'ADJ'), ('helpful', 'ADJ'), ('/r/Python', 'NOUN')]
```

Remove all tokens that are not nouns, verbs, adverbs, or adjectives

Punctuation removal

In case some of the tokens were incorrectly tagged by the above step, we also went through the list and removed any tokens that were found in the string.punctuation list.

```
print(nop)
```

```
[('language', 'NOUN'), ('is', 'VERB'), ('easy', 'ADJ'), ('very', 'ADV'), ('attractive', 'ADJ'), ('non-CS', 'ADJ'), ('people',
'NOUN'), ('language', 'NOUN'), ('is', 'VERB'), ('consistent', 'ADJ'), ('everything', 'NOUN'), ('is', 'VERB'), ('object', 'NOU
N'), ('pointer', 'NOUN'), ('object', 'NOUN'), ('rather', 'ADV'), ('can', 'VERB'), ('only', 'ADV'), ('pass', 'VERB'), ('pointe
r', 'NOUN'), ('scope', 'NOUN'), ('namespace', 'NOUN'), ('make', 'VERB'), ('sense', 'NOUN'), ('time', 'NOUN'), ('can', 'VERB'),
('learn', 'VERB'), ('gradually', 'ADV'), ('can', 'VERB'), ('start', 'VERB'), ('using', 'VERB'), ('even', 'ADV'), ('know', 'VER
B'), ('only', 'ADV'), ('language', 'NOUN'), ('Amazing', 'NOUN'), ('documentation', 'NOUN'), ('official', 'ADJ'), ('tutorial',
'NOUN'), ('is', 'VERB'), ('easy', 'ADJ'), ('read', 'VERB'), ('time', 'NOUN'), ('are', 'VERB'), ('are', 'VERB'), ('fairly', 'AD
V'), ('proficient', 'ADJ'), ('python', 'ADJ'), ('error', 'NOUN'), ('message', 'NOUN'), ('tells', 'VERB'), ('exactly', 'ADV'),
('error', 'NOUN'), ('is', 'VERB'), ('where', 'ADV'), ('is', 'VERB'), ('happening', 'VERB'), ('lot', 'NOUN'), ('other', 'ADJ'),
('languages', 'NOUN'), ('do', 'VERB'), ('now', 'ADV'), ('was', 'VERB'), ("n't", 'ADV'), ('case', 'NOUN'), ('years', 'NOUN'),
('ago', 'ADV'), ('inline', 'VERB'), ('help', 'NOUN'), ('magical', 'ADJ'), ('docstring', 'NOUN'), ('a=4', 'ADV'), ('help', 'NOU
N'), ('REPL', 'NOUN'), ('especially', 'ADV'), ('being', 'VERB'), ('able', 'ADJ'), ('get', 'VERB'), ('REPL', 'NOUN'), ('apps',
'NOUN'), ('python', 'ADV'), ('-i', 'NOUN'), ('batteries', 'NOUN'), ('again', 'ADV'), ('years', 'NOUN'), ('ago', 'ADV'), ('langu
age', 'NOUN'), ('came', 'VERB'), ('huge', 'ADJ'), ('standard', 'NOUN'), ('library', 'NOUN'), ('PEP', 'NOUN'), ('grandfather',
'NOUN'), ('gofmt', 'VERB'), ('rustfmt', 'VERB'), ('Makes', 'NOUN'), ('big', 'ADJ'), ('difference', 'NOUN'), ('when', 'ADV'),
('working', 'VERB'), ('team', 'NOUN'), ('Zen', 'NOUN'), ('import', 'NOUN'), ('course', 'NOUN'), ('language', 'NOUN'), ('grew',
'VERB'), ('popularity', 'NOUN'), ('ecosystem', 'NOUN'), ('pypi.org', 'NOUN'), ('amount', 'NOUN'), ('resources', 'NOUN'), ('book
s', 'NOUN'), ('website', 'VERB'), ('answer', 'NOUN'), ('questions', 'NOUN'), ('can', 'VERB'), ('think', 'VERB'), ('SO', 'NOU
N'), ('very', 'ADV'), ('active', 'ADJ'), ('helpful', 'ADJ'), ('mailing', 'NOUN'), ('list/usenet', 'NOUN'), ('group', 'NOUN'),
('active', 'ADJ'), ('helpful', 'ADJ'), ('/r/Python', 'NOUN')]
```

Punctuations that were incorrectly tagged are removed

Lemmatization

We used WordNetLemmatizer from the nltk.corpus library to lemmatize our tokens. Based on the POS tags that we have obtained previously, we are able to pass the correct POS parameter

to the lemmatize function for more accurate lemmatization. The four POS tags that we pass to the lemmatize function are: “n” for “NOUN”, “v” for “VERB”, “a” for “ADJ”, “r” for “ADV”.

```
print(lemmatized)
```

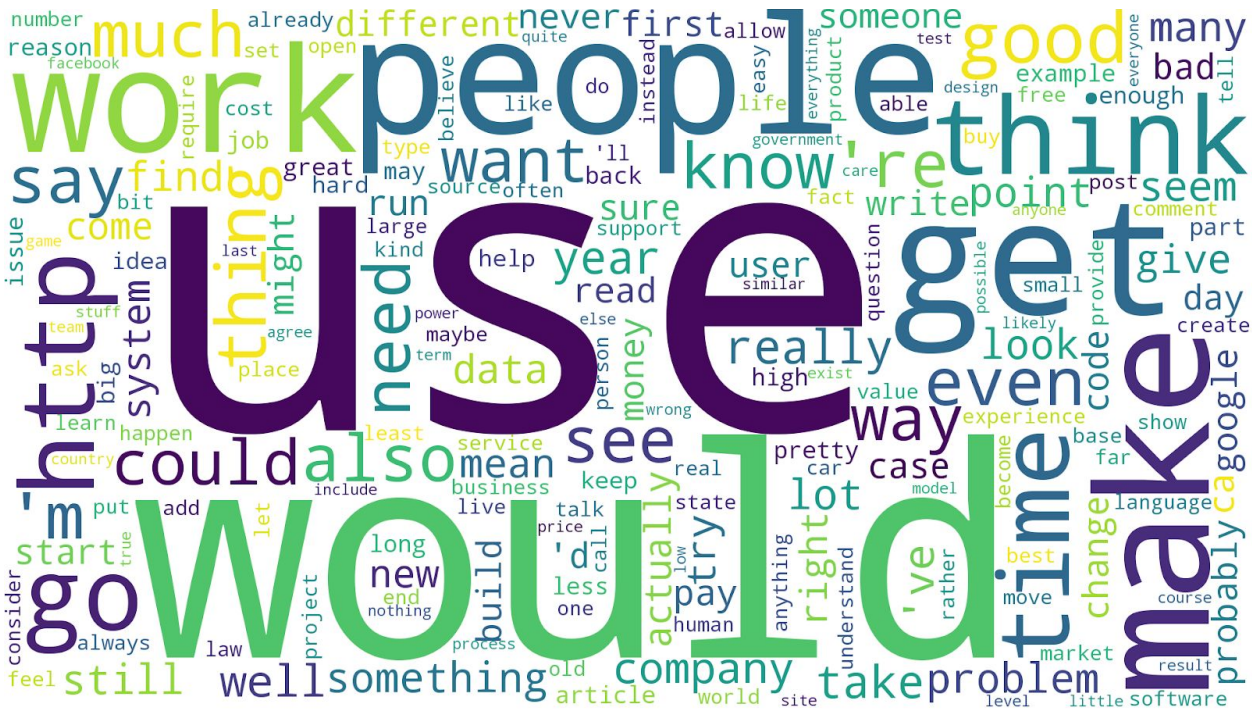
```
['language', 'be', 'easy', 'very', 'attractive', 'non-CS', 'people', 'language', 'be', 'consistent', 'everything', 'be', 'objec
t', 'pointer', 'object', 'rather', 'can', 'only', 'pass', 'pointer', 'scope', 'namespace', 'make', 'sense', 'time', 'can', 'lea
rn', 'gradually', 'can', 'start', 'use', 'even', 'know', 'only', 'language', 'Amazing', 'documentation', 'official', 'tutoria
l', 'be', 'easy', 'read', 'time', 'be', 'be', 'fairly', 'proficient', 'python', 'error', 'message', 'tell', 'exactly', 'error',
'be', 'where', 'be', 'happen', 'lot', 'other', 'language', 'do', 'now', 'be', "n't", 'case', 'year', 'ago', 'inline', 'help',
'magical', 'docstring', 'a=4', 'help', 'REPL', 'especially', 'be', 'able', 'get', 'REPL', 'apps', 'python', '-i', 'battery', 'a
gain', 'year', 'ago', 'language', 'come', 'huge', 'standard', 'library', 'PEP', 'grandfather', 'gofmt', 'rustfmt', 'Makes', 'bi
g', 'difference', 'when', 'work', 'team', 'Zen', 'import', 'course', 'language', 'grow', 'popularity', 'ecosystem', 'pypi.org',
'amount', 'resource', 'book', 'website', 'answer', 'question', 'can', 'think', 'SO', 'very', 'active', 'helpful', 'mailing', 'l
ist/usenet', 'group', 'active', 'helpful', '/r/Python']
```

Use the POS tag to lemmatize the words more accurately

Preliminary Wordcloud diagram

The first set of Wordclouds had words like “ask”, “show”, “ycombinator”, “hn” - which is shorthand for HackerNews. These words did not represent topics and can be removed from our analysis.

Preliminary titles wordcloud



Preliminary texts wordcloud

Adding Custom Stopwords

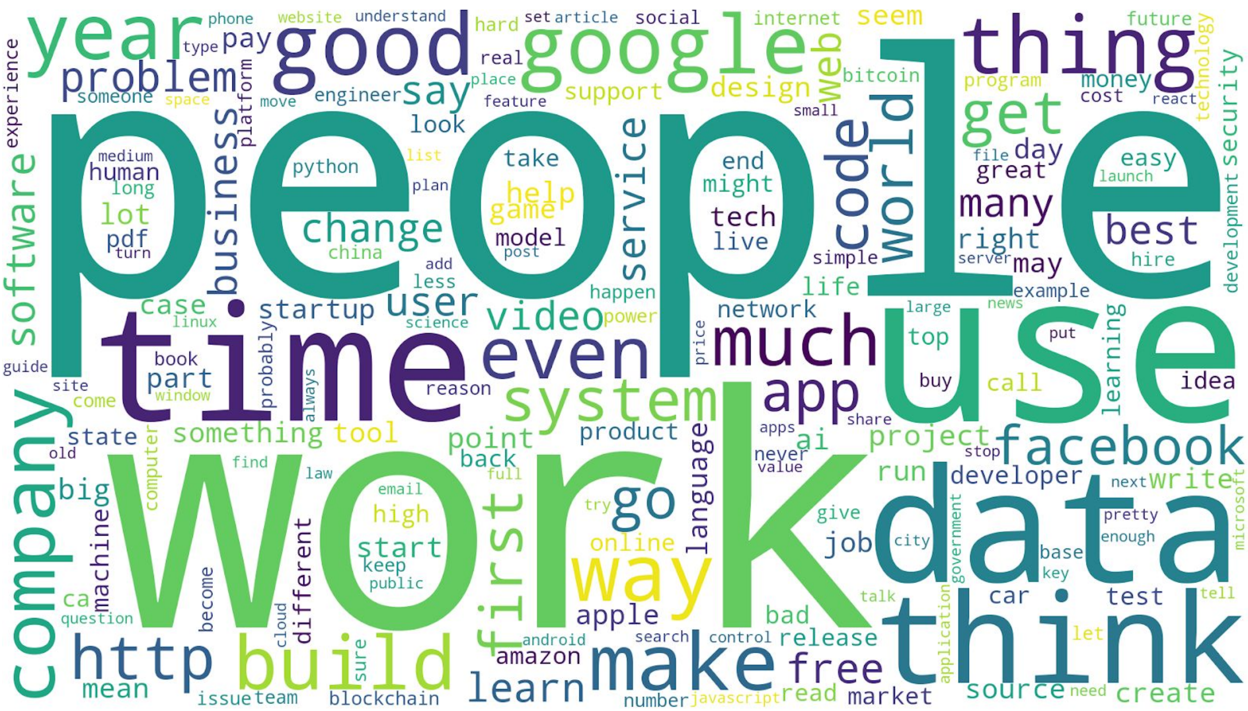
From the frequency distributions of our preliminary attempt, we picked out words which were commonly used in speech, and that are not representative of potential topics of discussion. We added these words into a custom stopwords list and preprocessed the raw data again to remove these custom stopwords

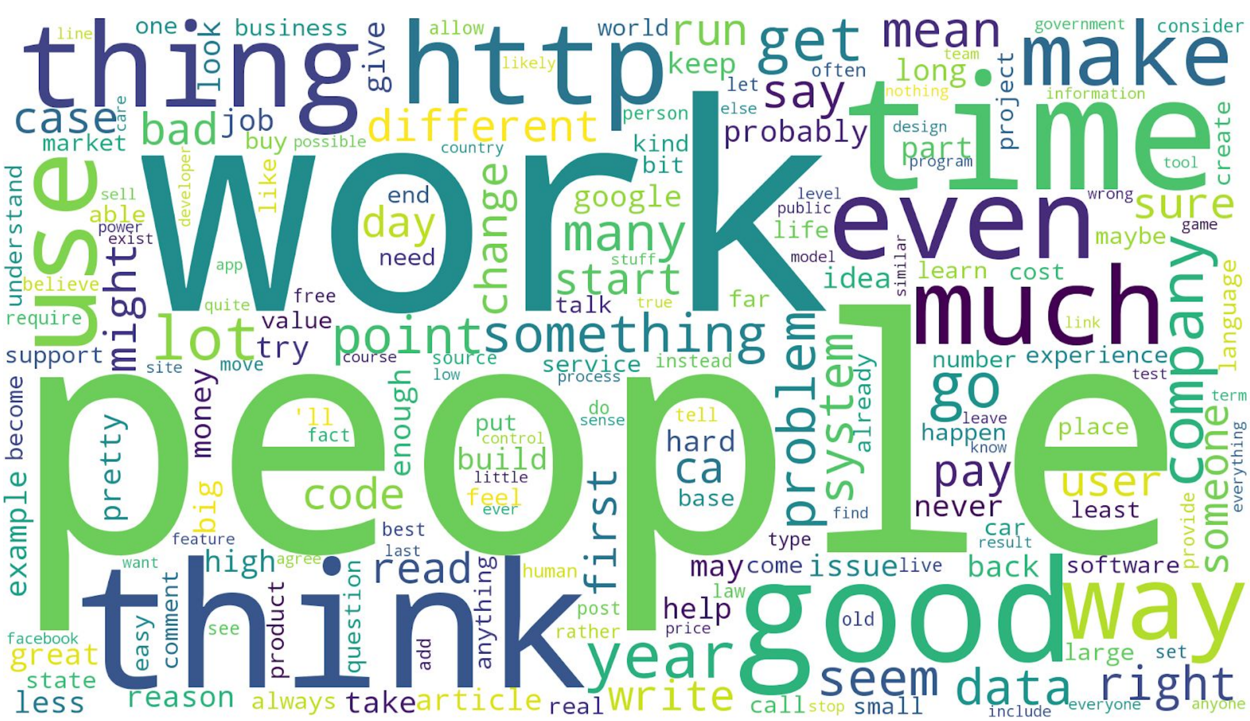
```
# Define custom stopwords
stop = stopwords.words('english')
custom_stop = ['hn', 'ask', 'show', 'new', 'use', 'make', 'get', 'go', 'open',
               'need', 'n\'t', '\'s', '\'m', '\'ve', 'would', 'http', 'go',
               'say', 'also', '\'re', 'need', 'see', '\'m', 'know', 'could',
               'want', 'really', 'take', 'well', '\'ve', 'look', 'still',
               'find', 'try', 'give', 'new', 'come', 'actually', '\'d']
stop.extend(custom_stop)
```

List of custom stopwords gathered from FD

Combining the Wordclouds

To have a clearer idea of all the topics that are being talked about on the forum, we decided to combine the two frequency distributions to see a combined wordcloud. However, because the text data had so many tokens to start with, its frequency distribution numbers overshadowed those from the titles data.





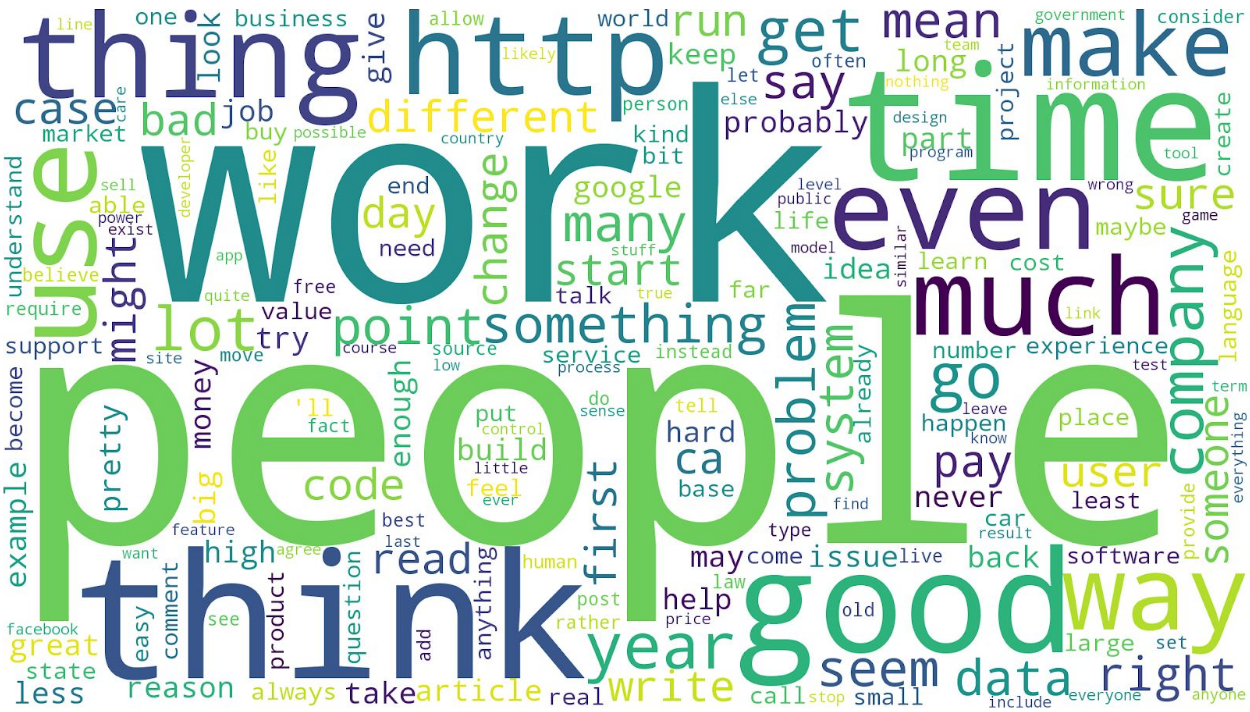
Similarities between preliminary combined wordcloud (top) vs text wordcloud (bottom)

Normalizing the Frequency Distribution

In order to obtain a meaningful result with the combined wordcloud, we had to normalize the frequency distributions before combining them. The normalized frequency distribution would represent the terms as a percentage of the total number of tokens processed.

Wordcloud Diagrams

Final titles wordcloud



Final texts wordcloud

Final combined wordcloud

TF-IDF

Text Cleaning

The preparation steps to generate the TF-IDF were:

1) Takes in raw corpus extracted from MongoDB
2) First round of cleaning
    a) Remove non-ASCII characters
    b) Unescape html encoding
    c) Remove html tags
3) Word Tokenization
4) Second round of cleaning
    a) Convert tokens to lowercase characters
    b) Stopword removal
    c) Lemmatization using WordNetLemmatizer
    d) Remove words with len < 3
5) Join tokens back into string

Generating the TFIDF

Then, we used the TdidfVectorizer library from sklearn to transform the preprocessed data into the TF-IDF. This tf-idf will be used for the generation of the LDA in the next steps.

```
[ ]  1 from sklearn.feature_extraction.text import TfidfVectorizer
     2
     3 vec_tfidf = TfidfVectorizer(min_df=2)
     4 texts_tfidf = vec_tfidf.fit_transform(texts_text)
     5 print(texts_tfidf.toarray()[:10])
```

```
[[ 0.         0.         0.         ...,  0.         0.         0.        ]
 [ 0.         0.         0.         ...,  0.         0.         0.        ]
 [ 0.         0.         0.47368839 ...,  0.         0.         0.        ]
 ...,
 [ 0.         0.         0.         ...,  0.         0.         0.39765614]
 [ 0.         0.         0.         ...,  0.         0.         0.        ]
 [ 0.         0.         0.         ...,  0.         0.         0.        ]]
```

Array representation of tf-idf

Word2Vec

The popular word2vec technique provides another representation of word embedding which is capable of capturing context of a word in a document and also its semantic and syntactic similarity with other words. This method of embedding is much more efficient in terms of space storage and is suitable for a large corpus like ours compared to count vector and TF-IDF embedding which scales linearly to the vocab size of the corpus and often gives very sparse

matrices. This word embedding is an example of a Vector Space Model (VSM) which is based on the distributional hypothesis. The distributional hypothesis assumes that words which appear in the same context shares similar semantic meaning. There are two flavours of word2vec, the Continuous Bag of Words (CBOW) and Skip-gram model, where the former predicts target words from a given source context words and the latter predicts source context words from given target words. The Gensim library [6], implements the word2vc family of algorithms with highly optimized C routines and Pythonic interfaces. The family of algorithms include the mentioned skip-gram and CBOW models, using either hierarchical softmax [7] or negative sampling [8]. In this subsection, we shall show how we use clean the comments corpus and train the word2vec model from Gensim.

Text Cleaning

In preparing the data to train our word2vec model, we adopted the following pipeline:

1) Takes in raw corpus extracted from MongoDB
2) First round of cleaning
    a) Remove comments with length < 5
    b) Unescape html encoding
    c) Remove html tags and unnecessary spaces
    d) Decontraction
3) Sentence Tokenization
4) Second round of cleaning
    a) Remove punctuation
5) Word Tokenization
6) Third round of cleaning
    a) Stopwords removals,
    b) POS tagging
    c) POS filtering
        i) Only allow Noun, Verbs, Adjectives and Adverbs
    d) Lemmatize Noun and Verbs

Here are some code snippets of the pipeline:

```
def clean(text):
    clean_text = html.unescape(text)
    clean_text = re.sub(r'\\n', ' ', clean_text)
    clean_text = re.sub(r'<a.*</a>',' ', clean_text)
    clean_text = re.sub(r'<p.*</p>', ' ', clean_text)
    clean_text = re.sub(r'<.?>', ' ', clean_text)
    clean_text = re.sub(r'</.?>', ' ', clean_text)
    clean_text = re.sub(r'\s+', ' ', clean_text)
    def decontracted(phrase):
        # specific
        phrase = re.sub(r"won't", "will not", phrase)
        phrase = re.sub(r"can\'t", "can not", phrase)

        # general
        phrase = re.sub(r"n\'t", " not", phrase)
        phrase = re.sub(r"\'re", " are", phrase)
        phrase = re.sub(r"\'s", " is", phrase)
        phrase = re.sub(r"\'d", " would", phrase)
        phrase = re.sub(r"\'ll", " will", phrase)
        phrase = re.sub(r"\'t", " not", phrase)
        phrase = re.sub(r"\'ve", " have", phrase)
        phrase = re.sub(r"\'m", " am", phrase)
        return phrase
    clean_text = decontracted(clean_text)
    return clean_text
```

First round of cleaning

```
def clean_2(text):
    # remove punctuations
    regex = re.compile('[%s]' % re.escape(string.punctuation))
    clean_text = regex.sub('', text)
    return clean_text
```

Second round of cleaning

```python
def clean_3(tokens):
    clean_tokens = [token for token in tokens if token not in stopwords.words('english')]
    pos = nltk.pos_tag(clean_tokens, tagset='universal')
    wnl = nltk.WordNetLemmatizer()
    new_tokens = []
    accepted_pos = ['NOUN', 'VERB', 'ADJ','ADV']
    to_lemmatize = ['NOUN', 'VERB']
    change_dict = {'NOUN':'n',
                   'VERB':'v',
                   'ADJ':'a',
                   'ADV':'r'}
    for i in pos:
        if i[-1] in accepted_pos:
            temp = i[0]
            if i[-1] in to_lemmatize:
                temp = wnl.lemmatize(temp, pos = change_dict[i[-1]])
            temp.lower()
            new_tokens.append(temp.lower())
    return new_tokens
```

Third round of cleaning

Putting everything together

```python
clean_text = clean(corpus)
sentences = nltk.sent_tokenize(clean_text)
sentences = [nltk.word_tokenize(clean_2(sentence)) for sentence in sentences]
sentences = [clean_3(tokens) for tokens in sentences]
```

Putting all together

The following are some screenshots of the cleaning of the raw text.

Example 1

```
'BTW: WebVR is now enabled by default in Firefox Nightly<p><a href="https:&#x2F;&#x2F;www.reddit.com&#x2F;r&#x2F;We
bVR&#x2F;comments&#x2F;3zmclh&#x2F;webvr_enabled_by_default_in_firefox_nightly&#x2F;" rel="nofollow">https:&#x2F;&#
x2F;www.reddit.com&#x2F;r&#x2F;WebVR&#x2F;comments&#x2F;3zmclh&#x2F;webvr_enabled...</a>'
```

to

```
['btw', 'webvr', 'enable', 'default', 'firefox', 'nightly']
```

Example 2

```
'I agree and disagree. As an operation person, I think we need to make an argument for why X can&#x27;t be done rig
ht away, but we must have a plan Y for migration. I know for a fact there is still some ancient database server run
ning in our data center serving our critical business but there&#x27;s a plan to migrate that to modern databases.
While re implementing everything is costly, company has to hire specialist from a specific consulting firm to suppo
rt such ancient database, and how long can we retain such talent without spending extreme amount and caution? &quo
t;I am scared if I did this will screw everything up.&quot; There will be a point they have to migrate such databas
e to something modern, and the cost is still several million plus.<p>The same argument that corporate world doesn&#
x27;t allow Python 3 or has a hard time to migrate to Python 3 because machines are running on some ancient RHEL se
rvers or because of some security requirement. I get it and I don&#x27;t get it. First, let&#x27;s ditch RHLE. Fuck
that. I really don&#x27;t see reason to use RHLE; fine, YMMV. But it is people&#x27;s job to do work. Some pieces c
an die, and some pieces will have to evolve, either from scratch, or slowly. No one said software development and o
peration support are easy, see <a href="https:&#x2F;&#x2F;pbs.twimg.com&#x2F;media&#x2F;CWC74tOVAAAsls0.jpg:large"
rel="nofollow">https:&#x2F;&#x2F;pbs.twimg.com&#x2F;media&#x2F;CWC74tOVAAAsls0.jpg:large</a>.'
```

to

```
['agree', 'disagree']
['operation', 'person', 'think', 'need', 'make', 'argument', 'x', 'do', 'right', 'away', 'must', 'plan', 'y', 'migr
ation']
['know', 'fact', 'still', 'ancient', 'database', 'server', 'run', 'data', 'center', 'serve', 'critical', 'busines
s', 'plan', 'migrate', 'modern', 'database']
['implement', 'everything', 'costly', 'company', 'hire', 'specialist', 'specific', 'consult', 'firm', 'support', 'a
ncient', 'database', 'long', 'retain', 'talent', 'spend', 'extreme', 'amount', 'caution']
['scar', 'screw', 'everything']
['point', 'migrate', 'database', 'something', 'modern', 'cost', 'still', 'several']
['argument', 'corporate', 'world', 'allow', 'python', 'hard', 'time', 'migrate', 'python', 'machine', 'run', 'ancie
nt', 'rhel', 'server', 'security', 'requirement']
['get', 'get']
['first', 'let', 'ditch', 'rhle']
['fuck']
['really', 'see', 'reason', 'use', 'rhle', 'fine', 'ymmv']
['people', 'job', 'work']
['piece', 'die', 'piece', 'evolve', 'scratch', 'slowly']
['one', 'say', 'software', 'development', 'operation', 'support', 'easy', 'see']
```

Word2Vec modeling

With the cleaning pipeline, we could now go ahead and start training the word2vec model.

We have chosen the following hyper parameters to train our model:

1) Batch size: One month worth of comments
2) Learning rate = 0.02
3) Vector dimension size = 100
4) Window size = 20
5) Minimum word count = 5
6) Seed = 123 (for reproducibility)
7) Epoch = 10

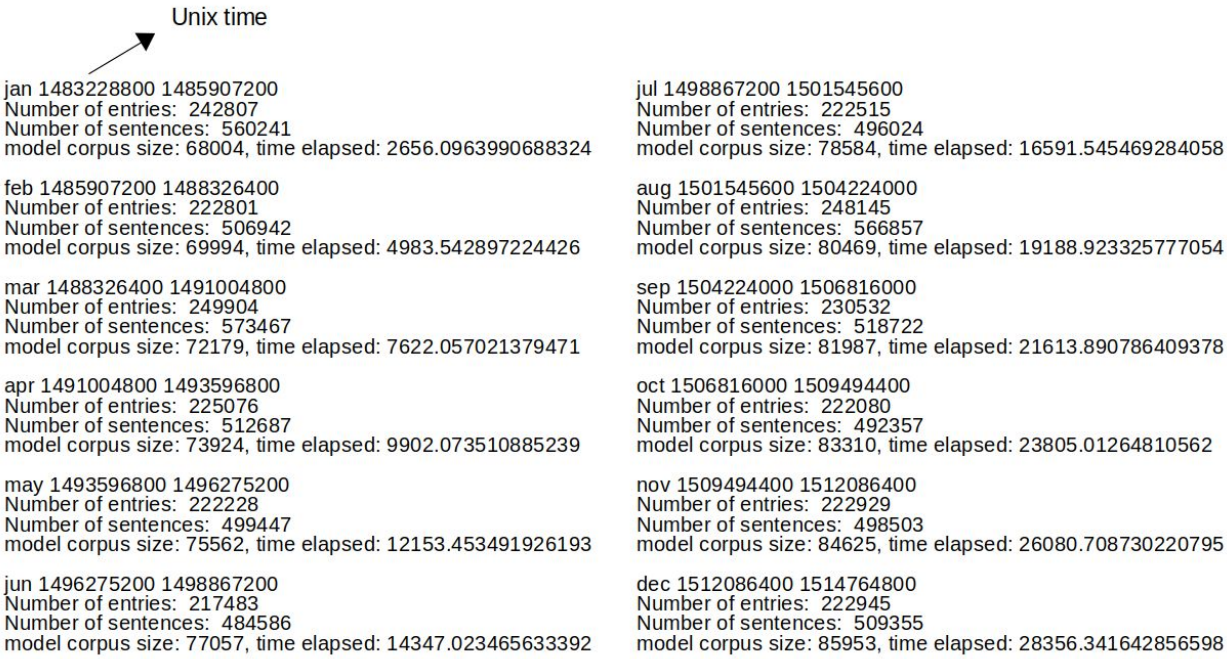
The code snippet to implement this is shown below:

```
from gensim.models import Word2Vec
import multiprocessing
```

```
cpu_count = multiprocessing.cpu_count()
print("Number of cpus: {}".format(cpu_count))
w2v_model = Word2Vec(sentences, size = 100, window = 20, min_count =5, workers = cpu_count, seed = 123)
```

```
Number of cpus: 4
```

Below is an example of training output which took **approximately 6** hours for year 2017

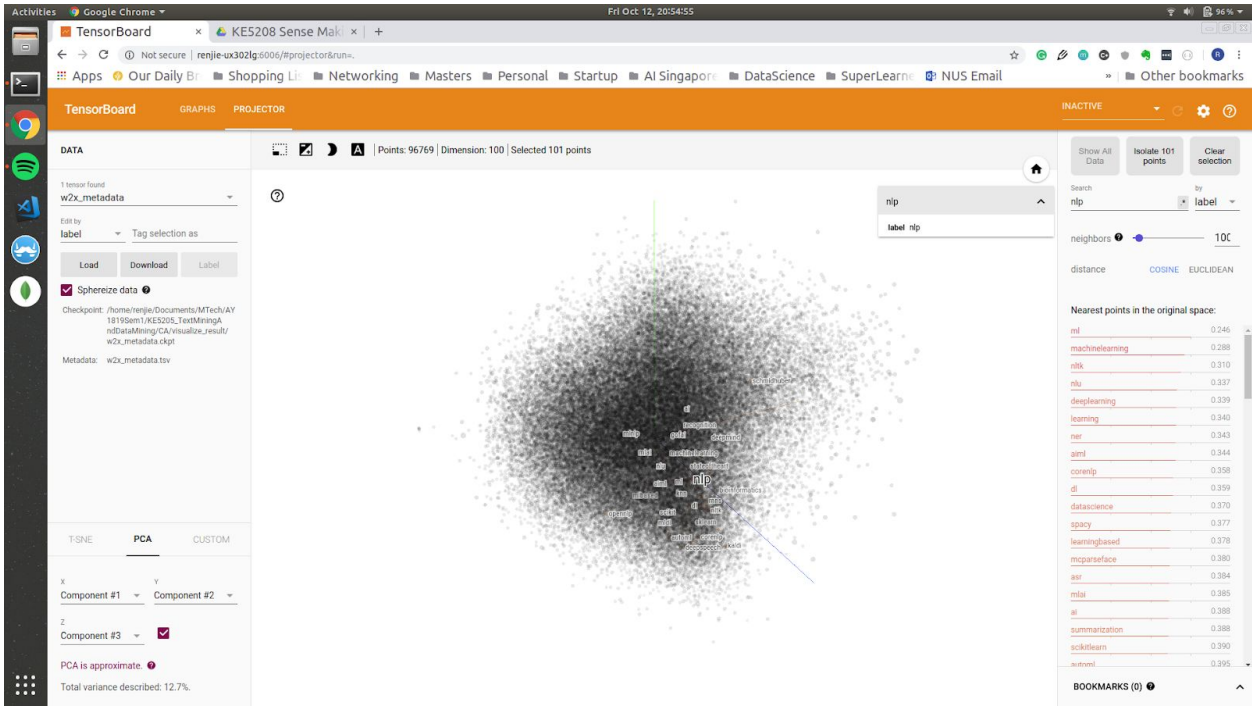


Training output of word2vec for Year 2017 corpus

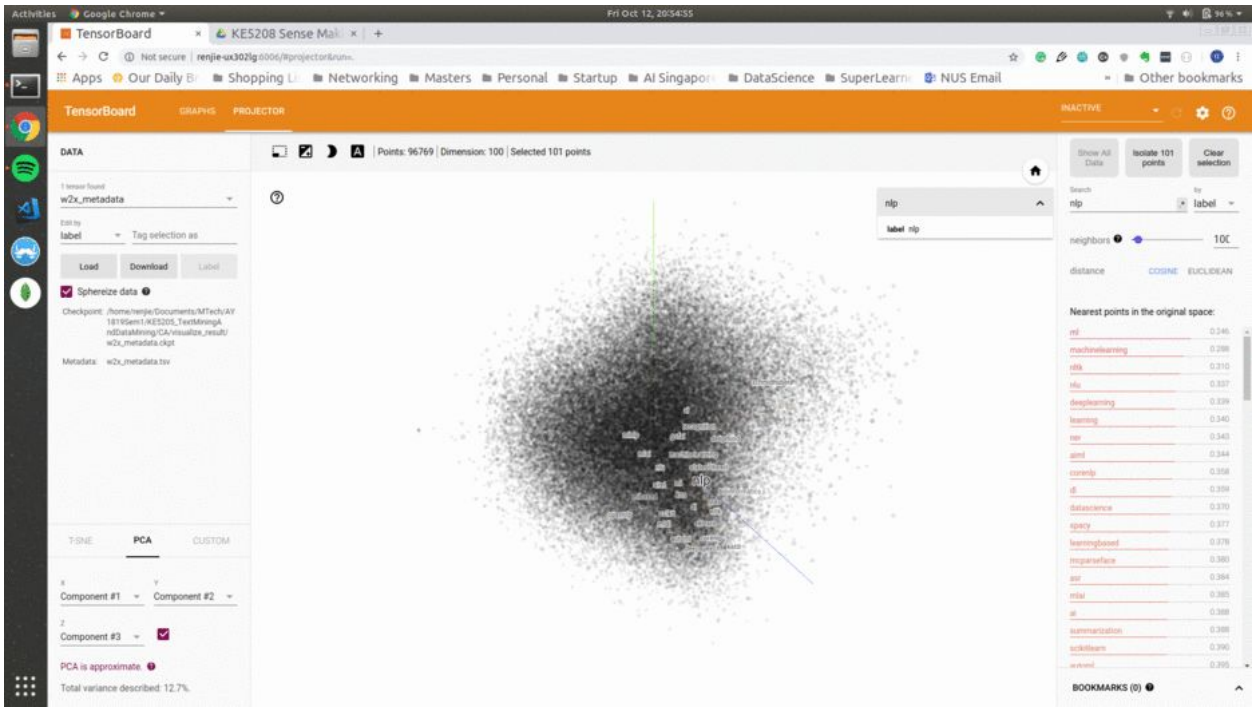
The trained word2vec weights can be found under the **folder directory `weights/`**. The **total vocab size** of the word2vec is **96769 words**.

Word2Vec testing

In the below figure, we have our word2vec visualized by reducing it to three dimension by applying PCA and showing it using Tensorboard Projector.



Screen shot of Tensorboard Project visualizing our three-dimensional word2vec embedding trained on our corpus. Displaying the top 100 closest neighbours to the word "nlp"



GIF Animation of our word2vec

The following visualization can be reproduced by running `tensorboard --logdir visualize_result/` on the root project directory.

With the trained weights, we can now carry out some simple vector operations like arithmetic and cosine similarity testing to get a few on the embedding we have achieved.

Word Similarity

```
similar = w2v_model.wv.most_similar('nlp', topn=5)
similar
```

```
/home/renjie/anaconda3/lib/python3.6/site-packages/
argument of issubdtype from `int` to `np.signedinte
= np.dtype(int).type`.
  if np.issubdtype(vec.dtype, np.int):
```

```
[('ml', 0.7547603845596313),
 ('machinelearning', 0.7133369445800781),
 ('nltk', 0.690643310546875),
 ('nlu', 0.6745010018348694),
 ('deeplearning', 0.6623262166976929)]
```

Words similar to “nlp”

```
similar = w2v_model.wv.most_similar('word2vec', topn=5)
similar
```

```
/home/renjie/anaconda3/lib/python3.6/site-packages/gensim
argument of issubdtype from `int` to `np.signedinteger` i
= np.dtype(int).type`.
  if np.issubdtype(vec.dtype, np.int):
```

```
[('embeddings', 0.880364179611206),
 ('fasttext', 0.8315322995185852),
 ('rnns', 0.7537660002708435),
 ('ngrams', 0.7423354983329773),
 ('rnn', 0.7334786653518677)]
```

Words similar to word2vec

```
similar = w2v_model.wv.most_similar('rnn', topn=5)
similar
```

```
/home/renjie/anaconda3/lib/python3.6/site-packages/
argument of issubdtype from `int` to `np.signedinte
= np.dtype(int).type`.
  if np.issubdtype(vec.dtype, np.int):
```

```
[('lstm', 0.88010573387146),
 ('rnns', 0.8664794564247131),
 ('dnn', 0.8377050161361694),
 ('cnns', 0.7999413013458252),
 ('lstms', 0.7964417934417725)]
```

Words similar to rnn

```
similar = w2v_model.wv.most_similar('windows', topn=5)
similar
```

```
/home/renjie/anaconda3/lib/python3.6/site-packages/gens
argument of issubdtype from `int` to `np.signedinteger`
= np.dtype(int).type`.
  if np.issubdtype(vec.dtype, np.int):
```

```
[('macosx', 0.7819986343383789),
 ('win7', 0.7778578400611877),
 ('win10', 0.7680132389068604),
 ('macos', 0.7463125586509705),
 ('osx', 0.7452881932258606)]
```

Words similar to windows

```
similar = w2v_model.wv.most_similar('docker', topn=5)
similar
```

```
/home/renjie/anaconda3/lib/python3.6/site-packages/gen
argument of issubdtype from `int` to `np.signedinteger
= np.dtype(int).type`.
  if np.issubdtype(vec.dtype, np.int):
```

```
[('kubernetes', 0.6883885860443115),
 ('lxd', 0.672843337059021),
 ('packer', 0.6570402383804321),
 ('dockerized', 0.6553440093994141),
 ('vagrant', 0.6540173888206482)]
```

Words similar to docker

```
similar = w2v_model.wv.most_similar('bitcoin', topn=30)
similar
```

```
/home/renjie/anaconda3/lib/python3.6/site-packages/gensi
argument of issubdtype from `int` to `np.signedinteger`
= np.dtype(int).type`.
  if np.issubdtype(vec.dtype, np.int):
```

```
[('cryptocurrency', 0.8309869170188904),
 ('cryptocurrencies', 0.8131473064422607),
 ('btc', 0.8097014427185059),
 ('cryptos', 0.748173713684082),
 ('egold', 0.7457767724990845),
 ('cryptocoins', 0.739509105682373),
 ('monero', 0.7346054315567017),
 ('xrp', 0.7310683727264404),
 ('bitcoins', 0.726100504398346),
 ('crypto', 0.7208844423294067),
 ('currencyi', 0.7154667377471924),
 ('currency', 0.7151015996932983),
 ('cryptocoin', 0.7134362459182739),
 ('altcoins', 0.7088978290557861),
 ('altcoin', 0.7083050012588501),
 ('fiat', 0.7038096785545349),
 ('dogecoin', 0.7013211250305176),
 ('raiblocks', 0.6984796524047852),
 ('eth', 0.6953101754188538),
 ('stablecoin', 0.6929824352264404),
 ('satoshis', 0.6874501705169678),
 ('ethereum', 0.6871819496154785),
 ('ether', 0.686867892742157),
 ('xmr', 0.6839380264282227),
 ('blockchains', 0.6663721799850464),
 ('litecoin', 0.6648331880569458),
 ('localbitcoins', 0.6614079475402832),
 ('currencythe', 0.6599621772766113),
 ('etherium', 0.6568745374679565),
 ('premined', 0.6552021503448486)]
```

Words similar to bitcoin

```
similar = w2v_model.wv.most_similar('ai', topn=30)
similar
```

```
/home/renjie/anaconda3/lib/python3.6/site-packages/
argument of issubdtype from `int` to `np.signedinte
= np.dtype(int).type`.
  if np.issubdtype(vec.dtype, np.int):
```

```
[('ais', 0.6827752590179443),
 ('aibased', 0.6533987522125244),
 ('deepmind', 0.6412022113800049),
 ('aiml', 0.6361273527145386),
 ('machinelearning', 0.6330133676528931),
 ('humanlevel', 0.6275997757911682),
 ('ml', 0.6252286434173584),
 ('ai1', 0.6187249422073364),
 ('nlp', 0.6128791570663452),
 ('deeplearning', 0.6120526194572449),
 ('agi', 0.6095365285873413),
 ('mlai', 0.6071229577064514),
 ('gameplaying', 0.5975470542907715),
 ('humanlike', 0.5954724550247192),
 ('aii', 0.5762498378753662),
 ('stateoftheart', 0.5759624242782593),
 ('superhuman', 0.5656391382217407),
 ('aidriven', 0.5571626424789429),
 ('superintelligent', 0.5553089380264282),
 ('aipowered', 0.5543289184570312),
 ('skynet', 0.5513216257095337),
 ('rl', 0.548020601272583),
 ('robotic', 0.5466969013214111),
 ('broodwar', 0.5460904836654663),
 ('kasparov', 0.5443347692489624),
 ('dnns', 0.5417208671569824),
 ('chatbots', 0.541343629360199),
 ('tdgammon', 0.5393096208572388),
 ('robotics', 0.5381337404251099),
 ('openai', 0.5312656760215759)]
```

Words similar to “ai”

```
similar = w2v_model.wv.most_similar('ar', topn=30)
similar
```

```
/home/renjie/anaconda3/lib/python3.6/site-packages/
argument of issubdtype from `int` to `np.signedinte
= np.dtype(int).type`.
  if np.issubdtype(vec.dtype, np.int):
```

```
[('arvr', 0.8297337293624878),
 ('vr', 0.7922728061676025),
 ('vrar', 0.7748532891273499),
 ('wearable', 0.7399231195449829),
 ('augmented', 0.7390269637107849),
 ('hololens', 0.7376449704170227),
 ('wearables', 0.719436526298523),
 ('headsets', 0.6951742768287659),
 ('arkit', 0.6901434659957886),
 ('hmd', 0.6819324493408203),
 ('magicleap', 0.6759721040725708),
 ('headset', 0.6691079139709473),
 ('lightfield', 0.6678154468536377),
 ('gearvr', 0.6563864946365356),
 ('stereoscopic', 0.6386868357658386),
 ('smartwatches', 0.6355723738670349),
 ('spectacles', 0.6309350728988647),
 ('hmds', 0.6308977603912354),
 ('augment', 0.6250040531158447),
 ('bigscreen', 0.6243996620178223),
 ('occulus', 0.6172214150428772),
 ('goggles', 0.61564040184021),
 ('goggle', 0.6143735647201538),
 ('tango', 0.612284779548645),
 ('futuristic', 0.6092898845672607),
 ('metaverse', 0.6069212555885315),
 ('glasses', 0.6053491830825806),
 ('immersive', 0.6043901443481445),
 ('kinect', 0.6029477119445801),
 ('oculus', 0.5992230772972107)]
```

Words similar to “ar”

Word Arithmetic

```
w2v_model.most_similar(positive=['browser', 'google'], topn=5)
```

```
/home/renjie/anaconda3/lib/python3.6/site-packages/ipykernel_l
`most_similar` (Method will be removed in 4.0.0, use self.wv.m
  """Entry point for launching an IPython kernel.
/home/renjie/anaconda3/lib/python3.6/site-packages/gensim/matu
argument of issubdtype from `int` to `np.signedinteger` is dep
= np.dtype(int).type`.
  if np.issubdtype(vec.dtype, np.int):
```

```
[('chrome', 0.7445471286773682),
 ('nonchrome', 0.6540366411209106),
 ('mozilla', 0.6472361087799072),
 ('nongoogle', 0.6438800096511841),
 ('adblocking', 0.632420539855957)]
```

“Browser” - “google”

```
w2v_model.most_similar(positive=['docker', 'os'], topn=5)
```

```
/home/renjie/anaconda3/lib/python3.6/site-packages/ipykerr
`most_similar` (Method will be removed in 4.0.0, use self.
  """Entry point for launching an IPython kernel.
/home/renjie/anaconda3/lib/python3.6/site-packages/gensim/
argument of issubdtype from `int` to `np.signedinteger` is
= np.dtype(int).type`.
  if np.issubdtype(vec.dtype, np.int):
```

```
[('vm', 0.7280181646347046),
 ('linux', 0.7059256434440613),
 ('chroots', 0.6838865280151367),
 ('lxd', 0.6769089102745056),
 ('rancheros', 0.6695564985275269)]
```

“Docker” - “os”

```
w2v_model.most_similar(positive=['doctor', 'female'], negative = ['male'], topn=5)
```

```
/home/renjie/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: Deprecat
`most_similar` (Method will be removed in 4.0.0, use self.wv.most_similar() instead
  """Entry point for launching an IPython kernel.
/home/renjie/anaconda3/lib/python3.6/site-packages/gensim/matutils.py:737: FutureWa
argument of issubdtype from `int` to `np.signedinteger` is deprecated. In future, i
= np.dtype(int).type`.
  if np.issubdtype(vec.dtype, np.int):
```

```
[('physician', 0.7627346515655518),
 ('nurse', 0.6784584522247314),
 ('anesthesiologist', 0.6604536771774292),
 ('doctors', 0.6466319561004639),
 ('pediatrician', 0.6431610584259033)]
```

“Docter” - “male” + “female”

SVD

LDA

As each document consisted of a mixture of various topics, we used the Latent Dirichlet Allocation (LDA) method to obtain the topic distribution in the documents in the first 8 months of 2018. After preprocessing and tokenizing the data in the steps above, we used the Gensim package to get the dictionary of the cleansed corpus.

```
import logging
import gensim
from gensim import corpora

logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)
```

```
C:\Users\joell\Downloads\newpython\lib\site-packages\gensim\utils.py:1197: UserWarning: detected Windows; aliasing chunkize to
chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

```
# Create the corpora.Dictionary object with the retained tokens.
dictionary = corpora.Dictionary(texts_clean)
dictionary
```

```
2018-10-14 13:50:14,622 : INFO : adding document #0 to Dictionary(0 unique tokens: [])
2018-10-14 13:50:14,629 : INFO : built Dictionary(1839 unique tokens: ['chain', 'cheaper', 'fewer', 'higher', 'trust/censorabil
ity']...) from 144 documents (total 2974 corpus positions)
```

The words were then converted into bag-of-words representation. This representation provides the term frequency of each words and does not indicate where these words occur in the document. The assumption behind this is that documents are similar if they have similar content. We are also able to learn something about the meaning of the document from the its content alone.

```
# Create corpus matrix
# Use the dictionary.doc2bow() function to convert the texts to bag-of-word representations.
corpus = [dictionary.doc2bow(text) for text in texts_clean]
corpus
```

```
[[],
 [(0, 1), (1, 1), (2, 1), (3, 1), (4, 1)],
 [(5, 2),
  (6, 1),
  (7, 1),
  (8, 2),
  (9, 1),
  (10, 1),
  (11, 1),
  (12, 1),
  (13, 1),
  (14, 1),
  (15, 1),
  (16, 1),
  (17, 1),
  (18, 1),
  (19, 1),
```

Next, we created the LDA model using the gensim package with the following assumptions:

- Number of latent topics : 5
- Number of documents to be used in each training chunk : 30
- Number of passes through the corpus: 20
- Random state fixed at 432 in order to validate the processing over multiple runs of the code

```
# Build an LDA model for 5 topics
%time lda = gensim.models.ldamodel.LdaModel(corpus, num_topics = 5,  alpha='auto',chunksize=30,id2word = dictionary, passes = 20,
#chunksize (int, optional) – Number of documents to be used in each training chunk.
#passes (int, optional) – Number of passes through the corpus during training.
```

```
2018-10-15 20:00:55,399 : INFO : using autotuned alpha, starting with [0.2, 0.2, 0.2, 0.2, 0.2]
2018-10-15 20:00:55,401 : INFO : using symmetric eta at 0.2
2018-10-15 20:00:55,403 : INFO : using serial LDA version on this node
2018-10-15 20:00:55,406 : INFO : running online (multi-pass) LDA training, 5 topics, 20 passes over the supplied corpus of 14
4 documents, updating model once every 30 documents, evaluating perplexity every 144 documents, iterating 50x with a converge
nce threshold of 0.001000
2018-10-15 20:00:55,407 : INFO : PROGRESS: pass 0, at document #30/144
2018-10-15 20:00:55,427 : INFO : optimized alpha [0.15151134, 0.16800782, 0.13767171, 0.18346992, 0.11167908]
2018-10-15 20:00:55,428 : INFO : merging changes from 30 documents into a model of 144 documents
```

The top 20 words of each topic were then displayed.

```
lda.show_topics(num_words=20)
```

```
[(0,
  '0.008*"market" + 0.007*"price" + 0.007*"playing" + 0.006*"intel" + 0.006*"network" + 0.006*"interview" + 0.005*"system" + 0.
005*"exist" + 0.005*"extra" + 0.004*"understand" + 0.004*"going" + 0.004*"technical" + 0.004*"except" + 0.004*"favorite" + 0.00
4*"solid" + 0.004*"hardware" + 0.004*"present" + 0.004*"wonder" + 0.003*"affect" + 0.003*"random"'),
 (1,
  '0.011*"android" + 0.009*"language" + 0.006*"development" + 0.006*"value" + 0.005*"built" + 0.005*"thing" + 0.005*"javascrip
t" + 0.005*"current" + 0.005*"currently" + 0.005*"project" + 0.005*"developer" + 0.004*"sharing" + 0.004*"going" + 0.004*"preci
sion" + 0.004*"forest" + 0.004*"ideal" + 0.004*"ensure" + 0.004*"france" + 0.004*"world-class" + 0.004*"skill"'),
 (2,
  '0.009*"feature" + 0.008*"important" + 0.007*"government" + 0.006*"model" + 0.006*"right" + 0.006*"point" + 0.005*"option" +
0.005*"provide" + 0.005*"without" + 0.005*"community" + 0.005*"least" + 0.005*"system" + 0.004*"first" + 0.004*"server" + 0.004
*"version" + 0.004*"modern" + 0.004*"account" + 0.004*"software" + 0.004*"popular" + 0.004*"query"'),
 (3,
  '0.011*"experience" + 0.009*"every" + 0.008*"still" + 0.008*"probably" + 0.007*"world" + 0.006*"everyone" + 0.006*"coming" +
0.006*"agree" + 0.005*"failing" + 0.005*"better" + 0.004*"reason" + 0.004*"using" + 0.004*"partly" + 0.004*"microsoft" + 0.004
*"patient" + 0.004*"white" + 0.004*"front" + 0.004*"situation" + 0.003*"research" + 0.003*"keeping"'),
 (4,
  '0.012*"education" + 0.008*"google" + 0.007*"given" + 0.007*"something" + 0.006*"level" + 0.006*"source" + 0.006*"thought" +
0.005*"money" + 0.005*"woman" + 0.005*"grade" + 0.005*"negligible" + 0.005*"sense" + 0.005*"arguing" + 0.005*"minimum" + 0.004
*"considered" + 0.004*"wrong" + 0.004*"talking" + 0.004*"needed" + 0.004*"markdown" + 0.004*"saying"')]
```

The top topics for each document were displayed.
After which, the topic labels were assigned to the topics based on the list of words with the highest probabilities.

```
# Pick the topic with the highest probability for each document, map it to the label

from operator import itemgetter
top_train = [ max(t, key=itemgetter(1))[0] for t in dtopics_train ]
print(top_train)
dict = {0: 'Technology Market', 1: 'Android',2: 'Government', 3:'User experience', 4:'Education'}
topic_train = [ dict[t] for t in top_train ]
print(topic_train)
```

```
[4, 2, 1, 0, 4, 1, 1, 0, 0, 3, 1, 3, 2, 3, 0, 3, 0, 0, 1, 2, 3, 3, 4, 1, 3, 1, 3, 2, 2, 3, 4, 4, 4, 4, 4, 0, 4, 0, 4, 4, 4, 4,
4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 2, 2, 0, 2, 0, 2, 1, 1, 1, 1, 2, 1, 0, 1, 2, 2, 0, 2, 0, 2, 1, 2, 0, 2, 2,
1, 0, 2, 2, 0, 3, 3, 3, 3, 1, 3, 4, 3, 3, 4, 3, 1, 0, 1, 2, 0, 3, 4, 3, 1, 3, 3, 3, 0, 3, 1, 3, 4, 4, 0, 1, 1, 1, 4, 2, 1, 1,
0, 3, 2, 4, 1, 0, 1, 4, 4, 1, 4, 4, 2, 1, 2, 2, 2, 3]
['Education', 'Government', 'Android', 'Technology Market', 'Education', 'Android', 'Android', 'Technology Market', 'Technology
Market', 'User experience', 'Android', 'User experience', 'Government', 'User experience', 'Technology Market', 'User experienc
e', 'Technology Market', 'Technology Market', 'Android', 'Government', 'User experience', 'User experience', 'Education', 'Andr
oid', 'User experience', 'Android', 'User experience', 'Government', 'Government', 'User experience', 'Education', 'Education',
'Education', 'Education', 'Education', 'Technology Market', 'Education', 'Technology Market', 'Education', 'Education', 'Educat
ion', 'Education', 'Education', 'Education', 'Education', 'Education', 'Education', 'Education', 'Education', 'Education', 'Edu
cation', 'Education', 'Education', 'Education', 'Education', 'Education', 'Education', 'Education', 'Education', 'Education',
'Government', 'Government', 'Technology Market', 'Government', 'Technology Market', 'Government', 'Android', 'Android', 'Androi
d', 'Government', 'Android', 'Technology Market', 'Android', 'Government', 'Government', 'Technology Market', 'Government', 'Te
chnology Market', 'Government', 'Android', 'Government', 'Technology Market', 'Government', 'Government', 'Android', 'Technolog
y Market', 'Government', 'Government', 'Technology Market', 'User experience', 'User experience', 'User experience', 'User expe
rience', 'Android', 'User experience', 'Education', 'User experience', 'User experience', 'Education', 'User experience', 'Andr
```

We proceeded to display our findings in a pivot table format using pandas. This allows clearer visibility to the topic contribution for each document.

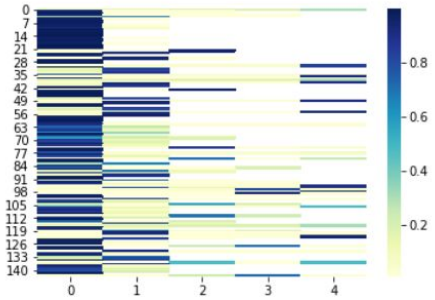
```
import pandas as pd

doc_topics = [lda.get_document_topics(doc) for doc in corpus]

topic_data = []

for document_id, topics in enumerate(doc_topics):

    document_topics = []

    for topic, probability in topics:

        topic_data.append({
            'document_id':  document_id,
            'topic_id':     topic,
            'topic':        topics_labels[topic],
            'probability':  probability
        })

topics_df = pd.DataFrame(topic_data)
topics_df.pivot_table(values="probability", index=["document_id", "topic"]).T
```

document_id	0					1			2	3	...	140	
topic	Android	Education	Government	Technology Market	User Experience	Education	Government	User Experience	Android	Education	...	Education	Government
probability	0.164561	0.310041	0.188123	0.149685	0.18759	0.018471	0.951632	0.011176	0.996289	0.018471	...	0.174881	0.81594

To have a summarized view on which are the top 5 topics, we displayed the findings in a heatmap using the seaborn package.

```
import seaborn as sns

doc_topics = [lda.get_document_topics(doc) for doc in corpus]

doc_topic_probabilities = []

for document in doc_topics:

    single_document = []

    for topic, probablity in document:

        single_document.append(probablity)

    doc_topic_probabilities.append(single_document)

docs_topics = pd.DataFrame(doc_topic_probabilities)
sns.heatmap(docs_topics,cmap='YlGnBu')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fbab4c8e48>
```



Results

Question 1: Trending Topics

What are the trending topics? Can they be used as part of market research for seed acceleration purposes?
We can use the generated wordclouds to pick out related words and determine the trending topics that most users are commenting about in 2018. The company can then focus on the specific keywords in order to prepare for any seed accelerator events.

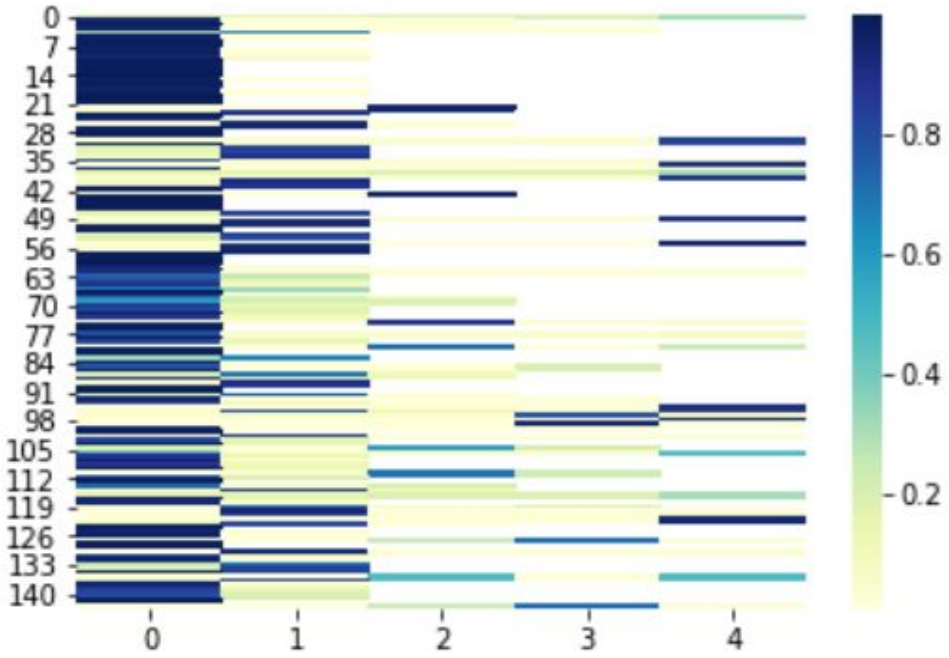
Keywords	Topic	Analysis
Job, work, people, time, problem, crypto	Jobs and Careers	A large portion of the user comments in the forum relate to their current work and issues faced when carrying out their jobs
Data, system, build, code, software, developer, program, AI, blockchain	Technology	Majority of the posts contain technological terms, which is to be expected of a tech forum. These keywords (e.g. AI and blockchain) can be used to prepare for seed accelerator programs as they represent the topics that users are interested in.
Google, apple, facebook, company, app	Mobile OS	Another common topic is mobile OS. This was derived from the keywords Google and Apple. Together with the technology keywords, mobile OS can be another topic to focus on for accelerator programs
Post, article, language, information, experience	Learning	The forum serves as a place to share ideas and articles.
Startup, business, money, market, government, idea	Starting a company	Since YCombinator is a seed accelerator company, it is no surprise that most of the topics on the forum are related to startup and businesses
Cryptocurrency, market, tesla, trump, government	Finance and Current Affairs	Fintech is another trending topic in the forums, which can be used to prepare for accelerator programs.

Summary of topics

The major trending topics that we have derived from the wordclouds are: 1. Technology, 2. Mobile OS, 3. Finance and Current Affairs.
It is possible to use the results in future seed acceleration programs by focusing on keywords such as: Blockchain, Cryptocurrency, AI, and Data.

Question 2: Topics to focus on

What should writers focus on?
Based on the LDA analysis, we obtained the following heatmap.



Topic 0: Technology Market
Topic 1: Android
Topic 2: Government
Topic 3: User Experience
Topic 4: Education

From the heatmap, it is observed that the topic on technology market was the most prevalent amongst the documents. This topic contained reviews on technology products, prices, hardware etc. This is in line with our own understanding as HackerNews is one of the main websites where many users visit to read reviews prior to buying a technological product such as a laptop.

The next most popular topic was Android, or in general, app development. Keywords mentioned under this topic included Javascript, Developer, Language, value etc.
Thereafter, popularity was split between government and education where many writers and comments were centered around the themes of community, money and grades.

Interestingly, these top few topics generated by LDA were consistent with the trending topics in Question 1. This further emphasizes that writer should focus on the following 3 topics in 2018: Technology market, App development and Government/Education.

Question 3: User Post Distribution

Recent studies have found that many forums tend to be dominated by a very small fraction of users [9]**. Is this true of Hacker News?**

This question was based on a study in 2014 [9], which found out that in a financial discussion forum, 0.4% of the users posted 47% of the 25 million posts.

To answer this question, we implemented the following steps:

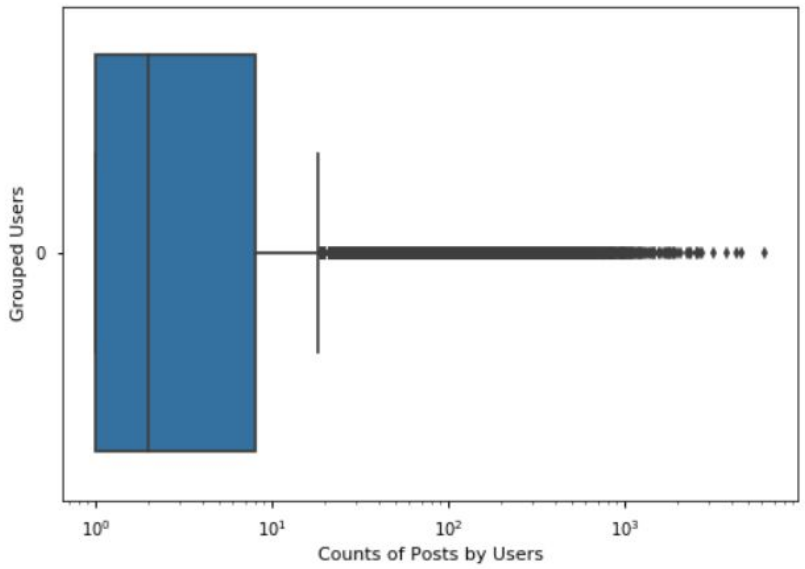
1. Break the question into 2 smaller questions: (a) Does a small fraction of users submit the majority of the posts or comments?, and (b) Does the contribution of these top users contain more words than the average user?
2. Clean the text data as described in the cleaning section above.
3. Tokenize the text data for each row, then create a column that states how many words for each text.
4. Remove rows that contain word counts of less than 3. These posts are unlikely to contribute to constructive forum discussion in Hacker News.
5. Find out the number of text contributions (or “comments” in the forum) that each user made in 2018 and sort them with the largest first. We disregard the number of post (or ‘titles’ in the data) by the user for 2 reasons.
    i. From our data, 80.2% of the forum activity are through comment posts
    ii. Bots can be written to automatically post content to the forum, while adding little value to the discussion.
6. Analyse the statistics for users and their comments and word count per comment.

At the data exploration stage, we found that for the 113169 unique users in 2018, the median number of posts or comments that each user submitted is only 2. The majority (75%) of the users submitted 8 or less comments in 2018. However, there exists “super contributors” who made hundreds or even thousands of comments. Please refer to the following charts in the next page.

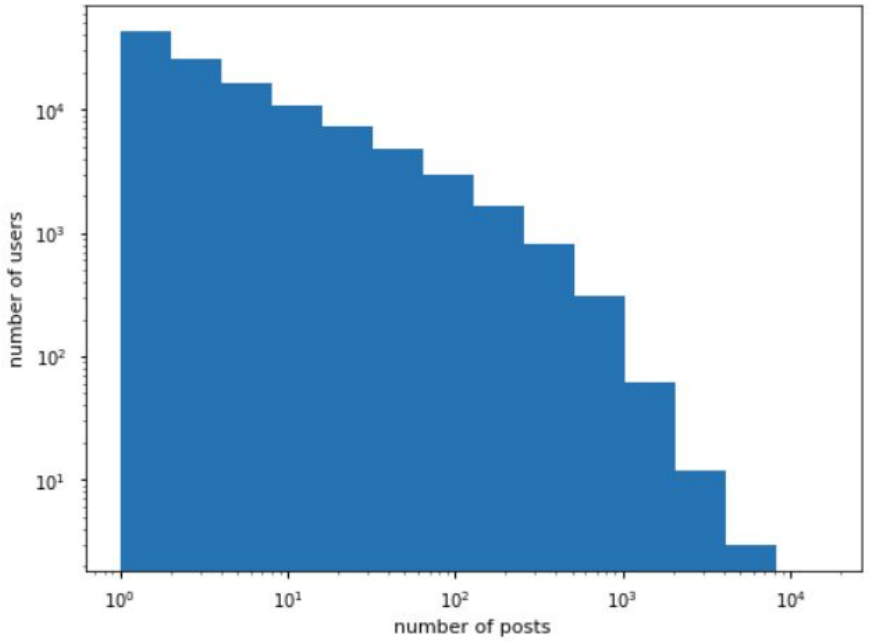
In order to account for comments or posts which do not contribute to the discussion, we remove comments with less than 3 words in them. Some examples of these comments are “Yes”, “Is it?” and “This”. The result is that a small proportion of users do submit the majority of the posts or

comments. The top 1% active users submit 31.1% of all posts, while the top 10% submit 75.8% of the posts.

There is no significant difference in the number of words between the top 10% commenters (63 words per comment) vs the rest of the 90% of the users (59 words per post).



Boxplot showing number of users against their comments or post counts



Histogram, with log scales, showing number of users against their comments or post counts

To summarize, Hacker News forum is indeed dominated by a small group of users. However, these users do not necessarily make longer comments than other users.

Before Cleaning Users - Post Counts		After Cleaning Users - Post Counts	
rbanffy	6218	dang	4367
dang	4572	dragonwriter	4278
dragonwriter	4286	pjmlp	3647
pjmlp	3806	jacquesm	3067
jacquesm	3198	scarface74	2710
scarface74	2728	icebraining	2631
icebraining	2667	TeMPOraL	2565
TeMPOraL	2579	coldtea	2540
JumpCrisscross	2574	s73v3r_	2310
coldtea	2563	tptacek	2305

The above table shows the top users before and after our data cleaning and transformation step. Note that the user, “rbanffy”, as highlighted in red above, was the top user at 6218 posts. However, he dropped out of the top 10 (even out of the top 30 users) after the data cleaning. This is strongly suggestive of bot behaviour, automatically posting links to the forum while not participating in discussion. Note also that “rbanffy” posted an inhumanly average of 22.3 times per day, as he had 6218 title posts from Jan to Sept 2018.

Question 4: Hacker News Bias

Hacker News has received complaints that the site is biased towards Y Combinator startups[10]**. Do the data support this?**

To answer this, we would first define bias-ness as the following:

- The site is biased if the topics talked about contain mainly pro Y Combinator startups contents
- The site is biased if the topics on the Y Combinator startups are over selling their capability

Till date, Y Combinator has accelerated 1,281 companies, with 1,029 active, 94 exited, and 158 dead [11]

From our wordcloud and LDA topic modeling, we could see that main topics were discussed on 1. Technology, 2. Mobile OS, 3. Finance and Current Affairs, 4. Blockchain, 5. Cryptocurrency, 6. AI, and 7. Data. No consistent mention of Y Combinator startups were observed.

If we recall the word2vec embedding we have trained upon the user comments in the previous, we could apply a cosine similarity on the Y Combinator startups to the entire corpus of word vectors to see if they are closely similar to any positive words.

We have chosen the following startups and the similar words are shown below. You could also run the following script to perform a cosine similarity on the startup of your interest. (insert script name)

Watsi

The first startup we did a cosine similarity on is Watsi [12], which is the first nonprofit funded by Y Combinator. Watsi is a non profit healthcare crowdsourcing platform that enables individual donors to directly fund medical care for individuals in developing countries without access to affordable medical care. Founded on May 2011 by Chase Adam and Jesse Cooke, Watsi has reported a 3 million USD revenue on 2014 [13].

Being the first nonprofit startup funded by Y Combinator, Watsi would be an ideal startup to see if there is any bias in Hacker News. Our word2vec cosine similarity shows there is little to no bias, with neutral words that purely describe Watsi showing up to be most similar, as shown below:

```
: similar = w2v_model.wv.most_similar('watsi', topn=10)
  similar

  /home/renjie/anaconda3/lib/python3.6/site-packages/gen
  argument of issubdtype from `int` to `np.signedinteger
  = np.dtype(int).type`.
    if np.issubdtype(vec.dtype, np.int):

: [('donation', 0.5762361884117126),
   ('donor', 0.5735940337181091),
   ('charity', 0.5446241497993469),
   ('nonprofit', 0.5402898788452148),
   ('givedirectly', 0.5401898622512817),
   ('notforprofit', 0.5381596088409424),
   ('gofundme', 0.5221085548400879),
   ('givewell', 0.5214895009994507),
   ('taxdeductible', 0.5214782953262329),
   ('crowdfunding', 0.5160553455352783)]
```

Airbnb

The second startup we are interested in needs no further introduction; it is the privately held global online marketplace broker for hospitality services which includes mainly homestay and tourism experiences. Founded in 2008, Airbnb is one the many success stories of Paul Graham's programme. It was evaluated to be at least worth 38 billion USD at May 2018 [14].

Applying our word2vec cosine similarity, it also shows that there is little to no bias towards Airbnb in the comments sections as shown below:

```
similar = w2v_model.wv.most_similar('airbnb', topn=10)
similar
```

```
/home/renjie/anaconda3/lib/python3.6/site-packages/gens
argument of issubdtype from `int` to `np.signedinteger`
= np.dtype(int).type`.
  if np.issubdtype(vec.dtype, np.int):
```

```
[('vrbo', 0.6933197975158691),
 ('airbnbs', 0.6799745559692383),
 ('homeaway', 0.6460170745849609),
 ('couchsurfing', 0.6147311925888062),
 ('expedia', 0.6009129881858826),
 ('sublet', 0.5885988473892212),
 ('wework', 0.5836077332496643),
 ('rentals', 0.5817515254020691),
 ('hotelscom', 0.580120325088501),
 ('bookingcom', 0.5730702877044678)]
```

Callisto

The last startup we would like to investigate is a recent graduate of the W18 class. Callisto [15] is a technology which aims to combat sexual assault and harassment. Being new to the startup game, we find that Callisto would be a suitable candidate to see if there is any bias towards Y Combinator recent graduates.

Sadly, Callisto did not appear more than 5 times among the comments sections and hence cannot be found in our word2vec corpus. This reinforces the fact that Hacker News is not bias towards its own startups.

To conclude, our analysis seems to disprove the claims that Hacker News is biased towards Y Combinator startups.

Conclusion

Hacker News is an immensely active and large online community, with an estimated 437 posts and comments per day by users. Monthly unique user count is at around 35,000. From our analysis, we have found that:

- Trending topics are related to the latest movements and technologies in the IT industry. Users are interested in how trends in technology, mobile devices and software, finance and current affairs affect their lives and career.

- Writers can focus on the similar topics above (Technology market, App development and Government/Education) to garner more views.
- The discussion in Hacker News is dominated by a small minority of users. The top 1% active users submit 31.1% of all posts, while the top 10% submit 75.8% of the posts.
- From our Word2Vec analysis of presence of bias towards Y Combinator startups (Watsi, AirBnB and Callisto), there are no bias detected in Hacker News forum discussions.

Bibliography

[1] "Hacker News." [Online]. Available: https://news.ycombinator.com/. [Accessed: 02-Oct-2018].

[2] "bigquery-public-data:hacker_news." [Online]. Available: https://bigquery.cloud.google.com/dataset/bigquery-public-data:hacker_news. [Accessed: 11-Sep-2018].

[3] "MongoDB Compass | MongoDB," *MongoDB*. [Online]. Available: https://www.mongodb.com/products/compass. [Accessed: 02-Oct-2018].

[4] "Hacker News." [Online]. Available: https://www.kaggle.com/hacker-news/hacker-news. [Accessed: 02-Oct-2018].

[5] "html — HyperText Markup Language support — Python 3.7.1rc1 documentation." [Online]. Available: https://docs.python.org/3/library/html.html. [Accessed: 02-Oct-2018].

[6] "gensim: topic modelling for humans." [Online]. Available: https://radimrehurek.com/gensim/index.html. [Accessed: 03-Oct-2018].

[7] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," 16-Jan-2013.

[8] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," 16-Oct-2013.

[9] T. Graham and S. Wright, "Discursive Equality and Everyday Talk Online: The Impact of 'Superparticipants,'" *J. Comput. Mediat. Commun.*, vol. 19, no. 3, pp. 625–642, 2013.

[10] L. Rao, "The Evolution Of Hacker News," *TechCrunch*, 18-May-2013. [Online]. Available: http://social.techcrunch.com/2013/05/18/the-evolution-of-hacker-news/. [Accessed: 05-Oct-2018].

[11] "Y Combinator Company List." [Online]. Available: https://yclist.com/. [Accessed: 04-Oct-2018].

[12] "Watsi | Building technology to finance universal healthcare," *Watsi*. [Online]. Available: https://watsi.org/. [Accessed: 06-Oct-2018].

[13] "[No title]." [Online]. Available: https://www.guidestar.org/FinDocuments/2014/453/236/2014-453236734-0be266f5-9.pdf. [Accessed: 06-Oct-2018].

[14] Trefis Team, "As A Rare Profitable Unicorn, Airbnb Appears To Be Worth At Least $38 Billion," *Forbes*, 11-May-2018. [Online]. Available: https://www.forbes.com/sites/greatspeculations/2018/05/11/as-a-rare-profitable-unicorn-airbnb-appears-to-be-worth-at-least-38-billion/. [Accessed: 06-Oct-2018].

[15] Callisto, "Home | Callisto: Tech to combat sexual assault." [Online]. Available: https://www.projectcallisto.org/. [Accessed: 06-Oct-2018].