Holden Bruce (hab6xf)
DS5001 - Exploratory Text Analytics
May 10, 2021

Final Project Summary

Introduction

In Module 6, we read a portion of the textbook that introduced various applications of the tf-idf vector model, explaining how we can decide if two documents are similar by representing the document as vectors of all the words and "computing the centroid of all those vectors. Given two documents, we can then compute their document vectors d1 and d2 and estimate the similarity between the two documents by cos(d1,d2)" (Jurafsky and Martin, page 115, chapter 6.6).

My first attempt at tackling this project was rather naive. I was given a training dataset of 1,000 companies that had been manually classified into 87 unique industries and a training dataset of 50,000 companies that were unlabeled. Without doing any data exploration, I started throwing models at the data to see what worked. My goal was to classify companies into predefined industry classifications based on their 2-10 sentence descriptions. Unsurprisingly, this approach was unsuccessful.

In this project, I attempt to use the various document similarity methods to shine light on the differences and similarities between Industry Classifications in a dataset of CrunchBase company data. While this classification problem is far from complete, I have a new sense of appreciation for the delicacy with which I need to approach this complex problem, if I ever expect to solve it.

Preprocessing

I decided that my first step ought to be to develop my own dataset, instead of using what was given to me, with the hope that better data may lend itself to developing a better classification algorithm. I downloaded CSV files from 77 different Industry designations from the CrunchBase database. In the first section of the project, I use glob to read in all of those files and then concatenate them into a single dataframe that I would use throughout my project. In this formatting phase, I dropped all the duplicates in this dataframe -- a necessary step since many companies in the CrunchBase database have multiple industry classifications (which is at the heart of why I was interested in this project in the first place).

The CrunchBase database offers a wide range of descriptive features, but I ended up focusing on four features: Organization Name, Industries, Industry Groups, and Full Description. The original goal of this classification problem was to correctly classify a company's industry based

on its description. In this project, I will be creating a digital analytical edition based on this Full Description column. While I explore both the Industries and Industry Groups columns, I end up working primarily with Industry Groups, which has far fewer unique values than the Industries column.

CrunchBase assigns industry classifications to companies; some companies have a single industry classification while others have as many as twelve. I decided to use the companies that had been given only a single industry classification as my dataset. After doing a bit of data exploration, I came back and increased my dataset to those with two industry classifications. I read through about a hundred of these companies and the first industry in the pair made plenty of sense for a classification of the company, so I ended up just splitting the pair and taking the first classification that CrunchBase had given it. This wasn't a perfect approach and required a bit of an assumption on my part that this was ok, but I felt it was important enough to increase the size of my dataset from 3,197 companies to 12,031.

OHCO

In Section 3, I lay out the OHCO model with a hierarchy of Industry (*industry_num*), Company (*company_num*), Sentence (*sent_num*), and Token (*token_num*). I create a LIB dataframe from the Organization Name and Industry Groups columns. With company_num as the index, I used categorical mapping in pandas (cat.codes) to create a "Industry_Index" column of numerical representations of the Industry Groups column.

company_num	Organization Name	Industry Groups	Industry_Index
1102	Chili Technology	Manufacturing	24
1122	Briggs & Stratton	Manufacturing	24
1404	Volvo Trucks North America	Manufacturing	24
1440	Soltec Trackers	Manufacturing	24
1542	Synchrony	Manufacturing	24
...	...	...	...
47187	clearspace	Professional Services	31
47190	ORS Partners, LLC	Administrative Services	0
47192	OPENonline	Privacy and Security	30
47193	Vantage Partners	Internet Services	23
47196	Cadre	Professional Services	31

12031 rows × 3 columns

I created a DOC dataframe using nltk.sent_tokenize to split the Full Description by sentences.

			sent_str
industry_num	**company_num**	**sent_num**	
24	**1102**	**0**	ChiliSleep is a sleep technology brand focused...
		1	As the original inventors of the world's first...
		2	ChiliSleep's award-winning sleep solutions chi...
	1122	**0**	Briggs & Stratton Corporation (Briggs & Stratt...
		1	The Company designs, manufactures, markets and...
...	**...**	**...**	...
31	**47196**	**1**	On the recruiting side, we cater exclusively t...
		2	On the incubator side, we are using artificial...
		3	Our Clients range from Pre-Series A startups t...
		4	We assist with technical recruiting and sustai...
		5	We are building a talent network of mobile eng...

44404 rows × 1 columns

I created a TOKEN table by passing the DOC table and splitting down to the token-level. I also used a Part-Of-Speech tagger to couple each token with its part of speech and lower-cased the token_str to create a new term_str column. Then I merged the LIB table's Industry_Index column with the TOKEN table. Finally, I created a max_pos variable that represented each word's most occurring Part-Of-Speech tag. This idea made me curious: could I apply this same logic to token/industry pairs? So I zipped a tuple of token_str and Industry_index. Here's what the TOKEN table looks like:

				pos_tuple	pos	token_str	term_str	Industry_Index	pos_max	Ind_Word_Tuple	ind_max
industry_num	**company_num**	**sent_num**									
24	**1102**	**0**	**0**	(ChiliSleep, NNP)	NNP	ChiliSleep	chilisleep	24	(ChiliSleep, NNP)	(ChiliSleep, 24)	(ChiliSleep, 24)
			1	(is, VBZ)	VBZ	is	is	24	(is, VBZ)	(is, 24)	(is, 16)
			2	(a, DT)	DT	a	a	24	(a, DT)	(a, 24)	(a, 16)
			3	(sleep, JJ)	JJ	sleep	sleep	24	(sleep, JJ)	(sleep, 24)	(sleep, 21)
			4	(technology, NN)	NN	technology	technology	24	(technology, NN)	(technology, 24)	(technology, 16)
...	**...**	**...**	**...**	...	...	...	...	...	...	...	...
31	**47196**	**5**	**14**	(for, IN)	IN	for	for	31	(for, IN)	(for, 31)	(for, 16)
			15	(our, PRP$)	PRP$	our	our	31	(our, PRP$)	(our, 31)	(our, 16)
			16	(portfolio, NN)	NN	portfolio	portfolio	31	(portfolio, NN)	(portfolio, 31)	(portfolio, 16)
			17	(of, IN)	IN	of	of	31	(of, IN)	(of, 31)	(of, 16)
			18	(companies., NN)	NN	companies.	companies	31	(companies, NNS)	(companies., 31)	(companies, 16)

900180 rows × 8 columns

Extracting a vocabulary from the TOKEN table, I made a VOCAB table that has each token represented in the index (token_num) with the term_str as one column and the number of times that term appears in the TOKEN table as another column, I also added whether or not the term is a number or a stopword. I tried adding a column that represented the Stem version of the word (using PorterStemmer) but that ended up being pretty useless since it seemed like the words that got Stemmed the most were Proper Nouns.

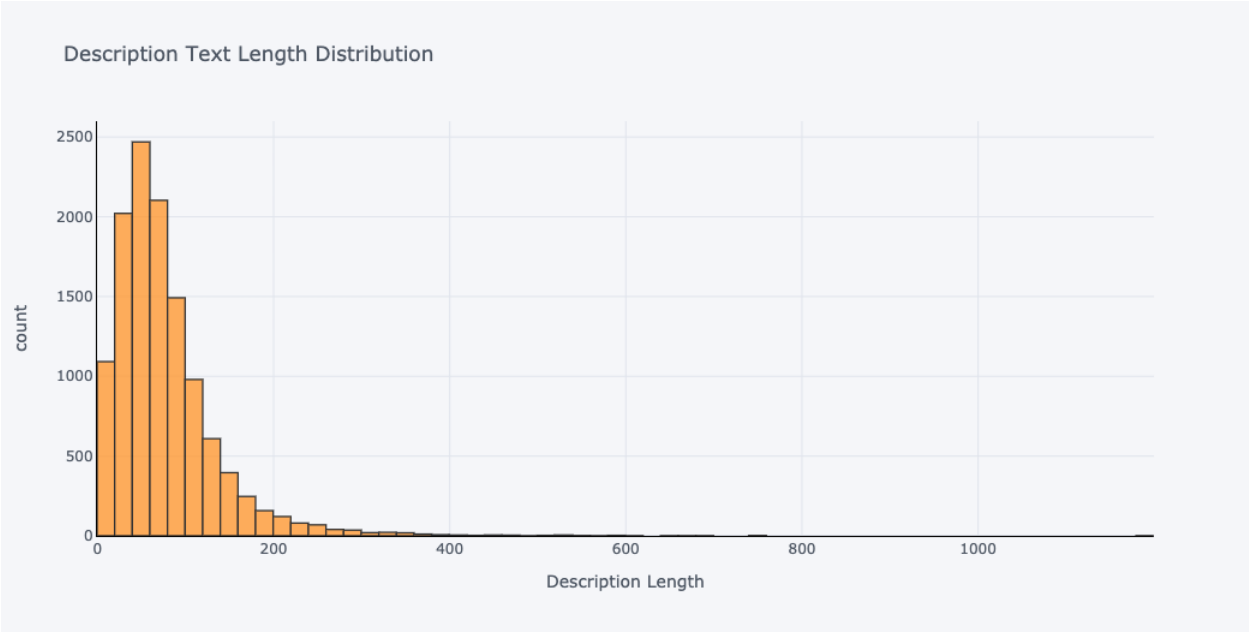
token_num	term_str	n	num	stop	p_stem
460	188	1	1	0	188
8748	chaat	2	0	0	chaat
15514	excursions	3	0	0	excurs
26987	muirhttpswwwcrunchbasecompersonayrmuirentity	1	0	0	muirhttpswwwcrunchbasecompersonayrmuirent
27290	nahjee	1	0	0	nahje
8545	cdn	3	0	0	cdn
8139	career	236	0	0	career
32122	proliferation	1	0	0	prolifer
16783	floefd	1	0	0	floefd
13228	distributor	119	0	0	distributor

Visualizations

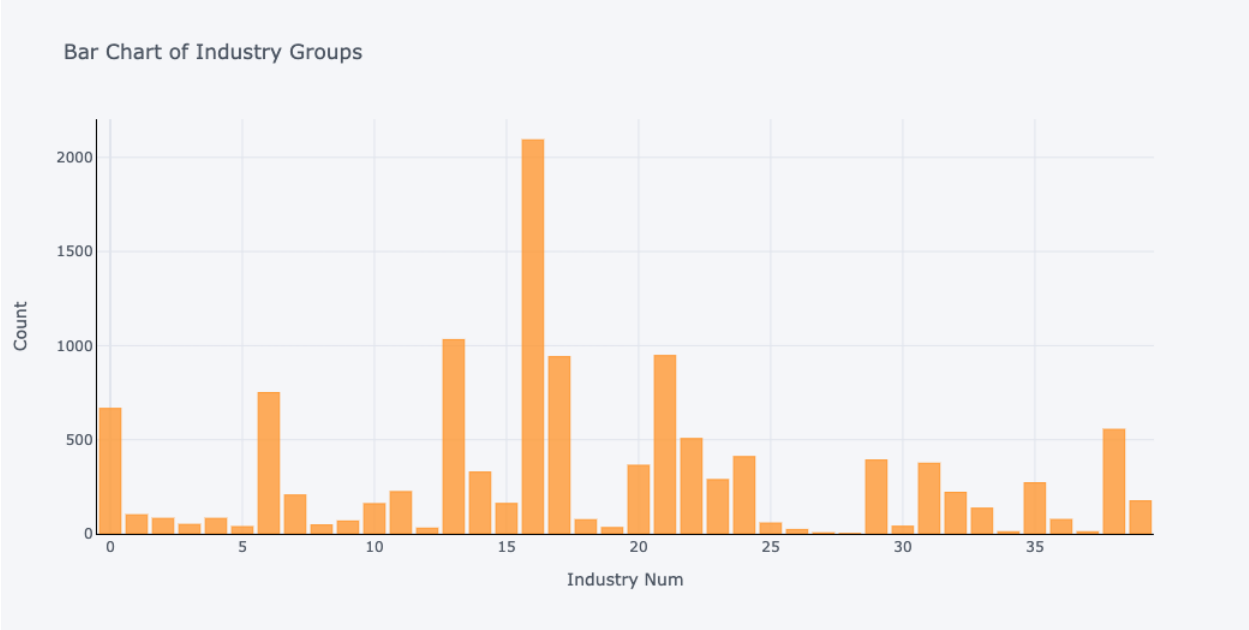
At this point, I wanted to start visualizing my data to better understand what I was working with. Throughout this project, I used cufflinks to bind iplot to plotly, so that I could use the power of the Plotly visualizations with the ease of python scripting.

The first visualization I created was a distribution of the lengths of words in the company descriptions. I was curious to see how long were the words in the dataset. It seemed like the vast majority of words were less than 10 characters long. There is a pretty long tail though, which I attribute to long Company names and the occasional website domain that got added into the company description.

Then I looked at the number of words per description. Just by looking at company descriptions (not sorting by industry), we can see that the vast majority of descriptions are relatively short (below 100 words) while there is a long right tail with a few very long descriptions.

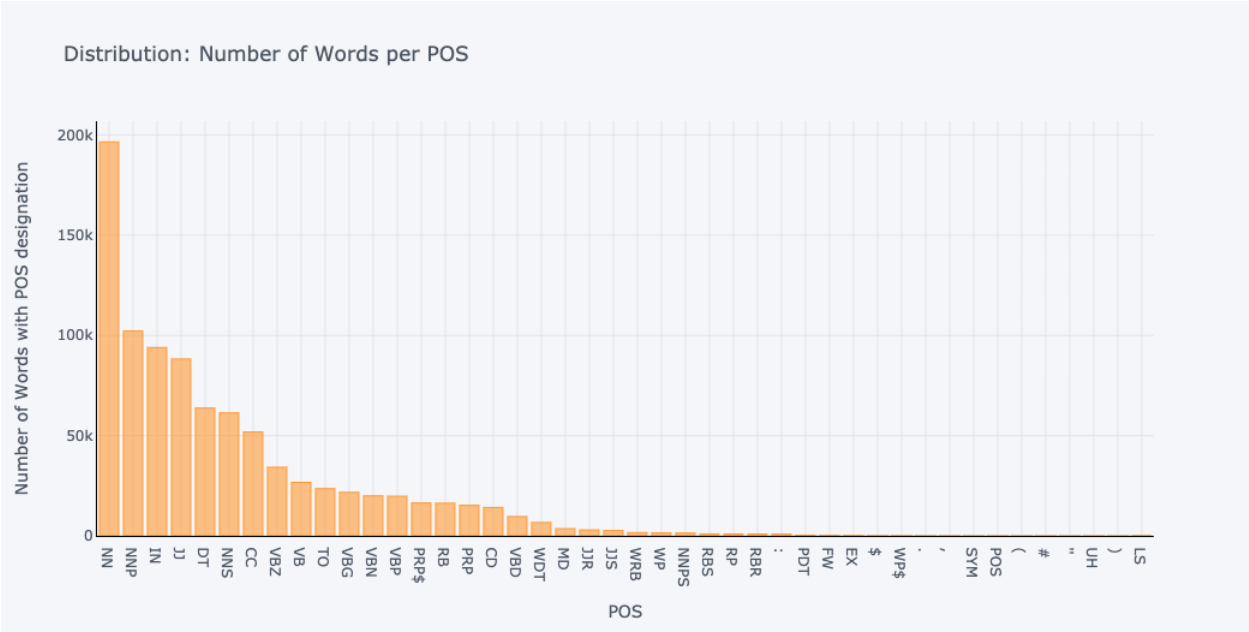


Then I looked at the distribution of industries. By observing this bar chart of Industry Group counts for number of descriptions (and thus number of companies) per Industry Group, it's pretty clear to see that this dataset is unbalanced. Some Industry Groups (like 27==Natural Resources, 28==Navigation and Mapping, 34==Science and Engineering, 37==Sustainability) barely show up on the chart because of how few companies there are in those industries (no more than 11), especially compared to Industry Group 16 (Financial Services) which has +2000 companies under that industry grouping.

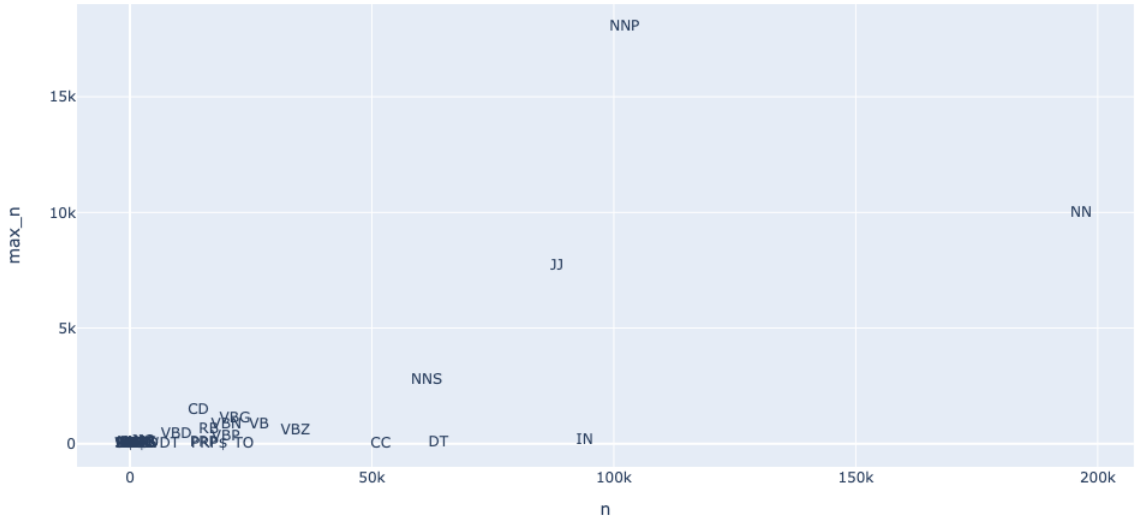


In Section 4, I dive into Vector Space Models, Bag of Words, and TF-IDF. The first visualization I made in this section was the distribution of the number of words per part of the speech tag. A

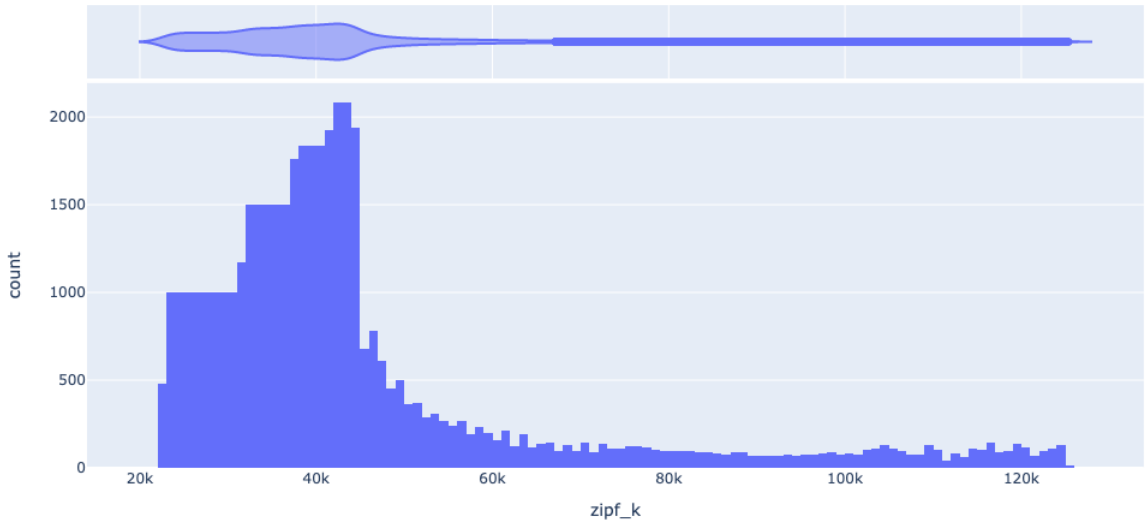
huge portion of words in the dataset are nouns and proper nouns, which is troubling because these are not reproducible on test sets since proper nouns are likely unique to the company. The fact that such a large portion of our dataset are unique words will likely lead to training accuracy scores much higher than test accuracy scores (overfitting).



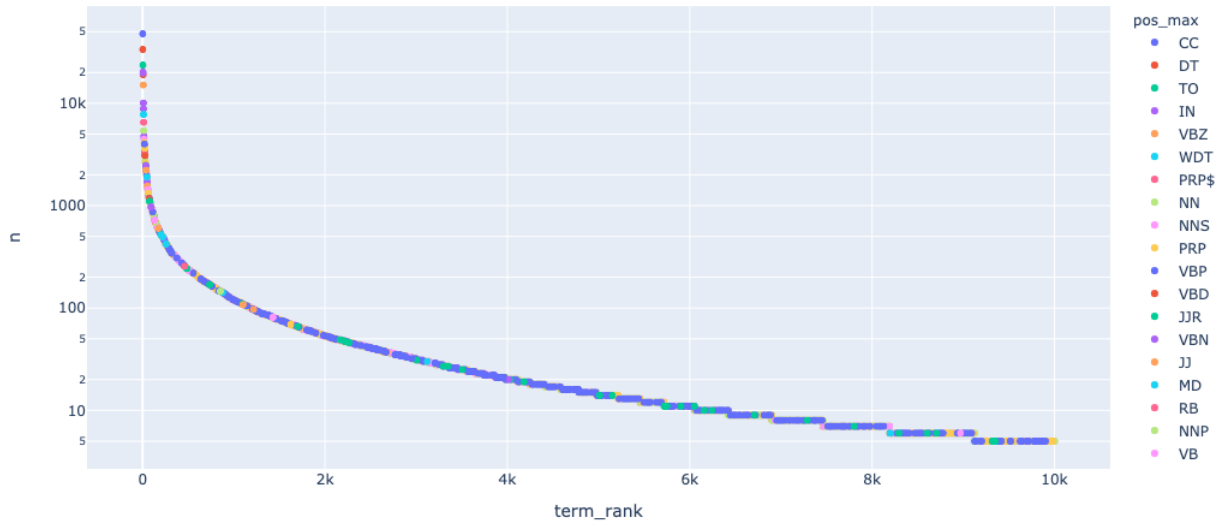
Here's another way of visualizing this problem. This scatterplot has the number of words per POS tag on the x-axis and the number of max_pos words on the y-axis. The axes are saying similar things but calculating the number of instances in different ways. However, both show that Nouns and Proper Nouns are by far the most occurring words in the dataset.



Zipf's law states that the frequency of a word is inversely proportional to its ranking. After calculating Zipf's K for each token and adding the values to a new column in the VOCAB table, I plotted a histogram of Zipf's K. The term_rank, which was used to calculate K, shows the most frequent terms in the VOCAB table ordered from most to least frequent. Since Zipf's K equals this term rank multiplied by the frequency of the words occurrence, looking at a Histogram of this value should give us an idea of how many "important" words there are and to what degree they are "important". With count on the y-axis and zipf_k on the x-axis, the violin plot shows a long right tail where a small number of words have a very large Zipf's K, versus a vast majority of words in the dataset with Zipf's K values below 45k.



As a sanity-check that I had made the correct calculations and that there weren't any issues with the data, I also plotted the relationship between rank and frequency to make sure that I was getting this power distribution:



As Rank increases, n decreases. Zipf_k has a distribution that bunches up where the mean/apex is around some significant-looking terms. The apex of the zipf_k seems to be good but the most occurring words (where n is darkest blue) are mostly stopwords.

Out[28]:

token_num	term_str	term_rank	n	zipf_k	pos_max
3851	and	1	47740	47740	CC
39883	the	2	33537	67074	DT
40328	to	3	23607	70821	TO
39874	that	10	7776	77760	WDT
3793	an	20	3900	78000	DT
39547	technology	30	2620	78600	NN
32015	products	40	2173	86920	NNS
9437	clients	50	1675	83750	NNS
19402	help	60	1419	85140	VB
30317	people	70	1197	83790	NNS
12699	development	80	1088	87040	NN
27543	needs	90	983	88470	NNS
43368	well	100	924	92400	RB

30317	people	70	1197	83790	NNS
12699	development	80	1088	87040	NN
27543	needs	90	983	88470	NNS
43368	well	100	924	92400	RB
17422	full	200	521	104200	JJ
21625	integrated	300	368	110400	JJ
11704	custom	400	290	116000	NN
31479	practice	500	237	118500	NN
38214	storage	600	204	122400	NN
26785	mortgage	700	177	123900	NN
1161	3d	800	156	124800	CD
42575	view	900	138	124200	NN
4674	assessment	1000	120	120000	NN
38529	subscription	2000	54	108000	NN
29266	originally	3000	32	96000	RB

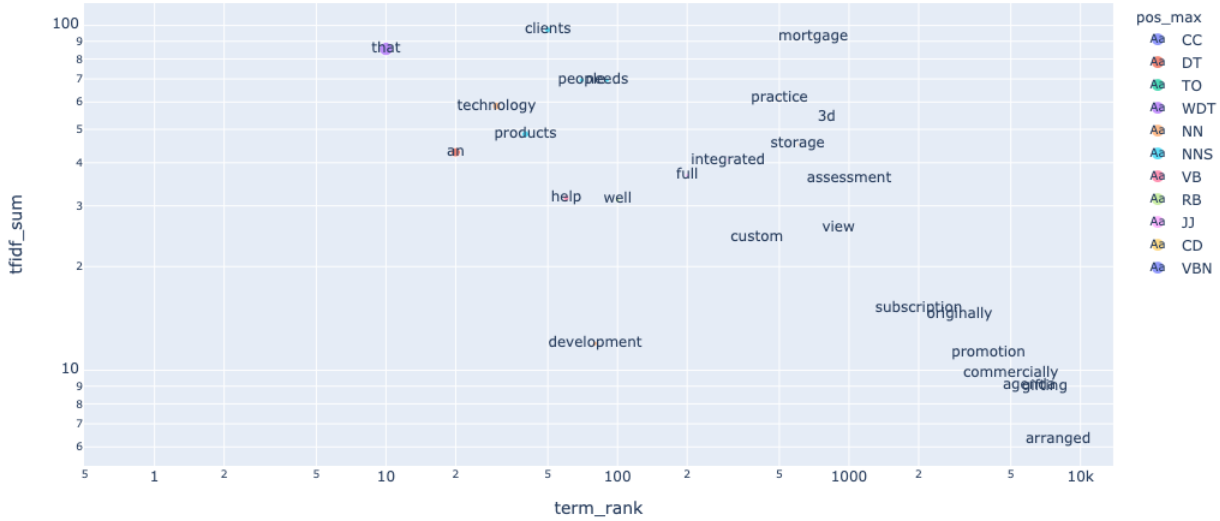
Next, I built a Bag of Words (BOW), a Document-Term Matrix (DTM), and set the stage for building a TF-IDF table. I used the count method instead of binary 0/1 for the DTM because I wanted to be able to track the number of times a word was present in a company description instead of just noting whether the word was present or not. My thought was that companies describe themselves using the same words, but if there were differences in the frequency of words across industries then that would be helpful to know. After building these tables and calculating some important metrics, I added them to the VOCAB table. Sorting by tfidf_sum, the most significant terms here are bank, wine, staffing, medical, students, banking, etc.

This is interesting to see but it's not exactly helpful in our ultimate goal of classifying companies into industries based on their descriptions. Understanding a term's significance within the corpus isn't what we're looking for, we need to understand a term's significance within the corpus of industry-specific company descriptions.

token_num	term_rank	term_str	n	num	stop	p_stem	pos_max	zipf_k	p	h	df	idf	tfidf_mean	tfidf_sum	tfidf_median	tfidf_max
5508	102	bank	908	0	0	bank	NNP	92616	0.001009	0.010040	18	0.346787	17.493502	314.883037	0.693575	299.277601
43745	238	wine	457	0	0	wine	NN	108766	0.000508	0.005556	12	0.522879	19.912966	238.955587	1.045757	165.229684
37846	187	staffing	553	0	0	staf	VBG	103411	0.000614	0.006554	15	0.425969	15.704047	235.560709	1.277906	207.020804
25627	95	medical	949	0	0	medic	JJ	90155	0.001054	0.010426	23	0.240332	9.916314	228.075215	1.441993	155.975569
38423	120	students	829	0	0	student	NNS	99480	0.000921	0.009287	22	0.259637	9.783606	215.239330	1.038549	188.496687
5515	163	banking	617	0	0	bank	NN	100571	0.000685	0.007204	18	0.346787	11.887104	213.967879	0.693575	200.443167
21595	67	insurance	1242	0	0	insur	NN	83214	0.001380	0.013109	27	0.170696	7.852026	212.004714	0.853481	173.939455
24295	358	loans	322	0	0	loan	NNS	115276	0.000358	0.004095	9	0.647817	23.177470	208.597229	1.295635	188.514887
30067	197	patients	532	0	0	patient	NNS	104804	0.000591	0.006338	17	0.371611	11.629241	197.697089	1.114833	159.792760
39369	264	tax	414	0	0	tax	NN	109296	0.000460	0.005099	15	0.425969	11.756737	176.351055	1.703875	146.959213
16440	52	financial	1555	0	0	financi	JJ	80860	0.001727	0.015853	31	0.110698	5.552769	172.135853	0.996285	128.963517
5845	544	beer	223	0	0	beer	NN	121312	0.000248	0.002968	7	0.756962	24.114645	168.802515	6.055696	130.954418
37642	572	spirits	215	0	0	spirit	NNS	122980	0.000239	0.002874	7	0.756962	23.249546	162.746820	0.756962	140.037961

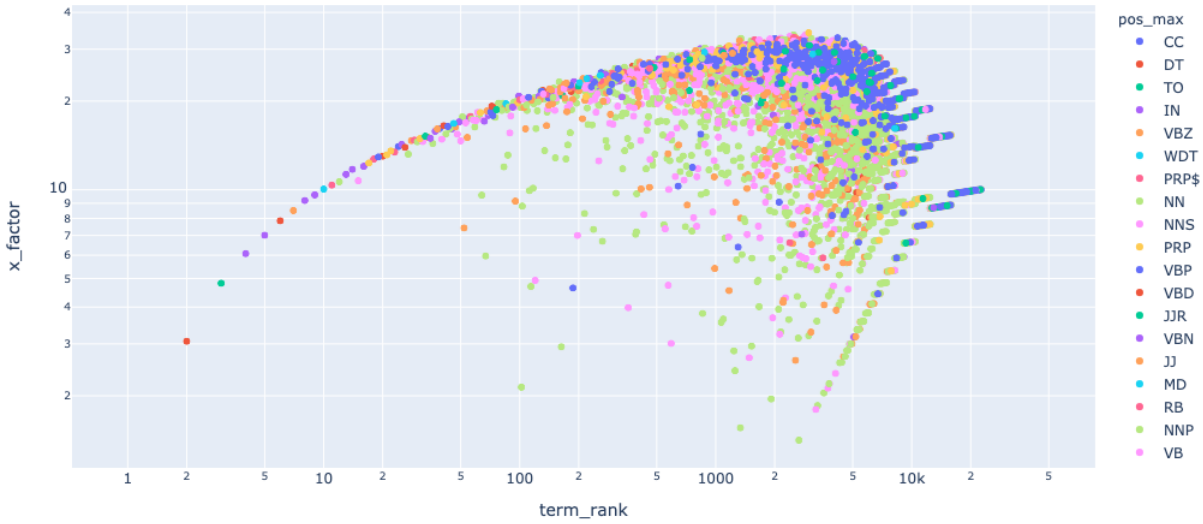
After replotting the Demo Table with TF-IDF, it was clear that Zipf's K and TFIDF sum actually performed comparably in this dataset. The apex for both were centered around words that were not stopwords and that seemed pretty significant (practice, storage, mortgage, 3d, etc.). TFIDF Mean gives too much credit to infrequent words, TFIDF sum does a better job of producing actually significant words across the corpus (higher n values and generally more predictive significance). Thus, I used tfidf_sum moving forward in the project.

Here is a scatterplot showing term rank on the x-axis and TFIDF sum on the y-axis. This is a good representation of some of the words that are considered most important from these calculations.

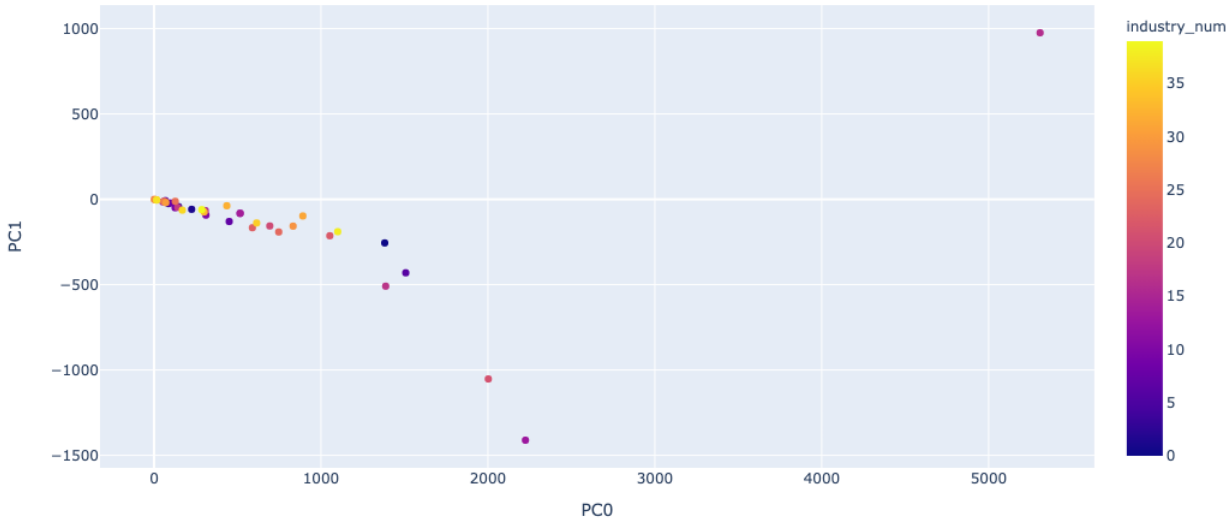


One of the best looking plots that came out of this section is the scatterplot showing the relationship between term rank and x_factor. This is a really nice curve that seems to perform

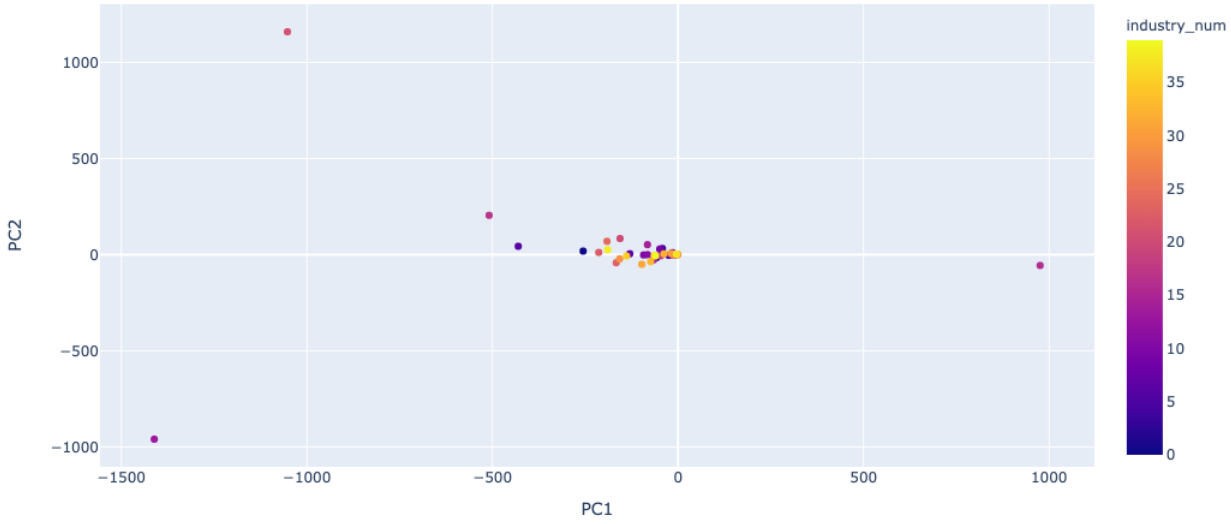
well the function that we're looking for. If you zoom in at the apex you can see what the function has determined to be the most significant words. Also, NNS and NN seem to fall out of the arch.



In Section 5, I decided to get rid of NNP and NNPS (PennTreebank codes for proper nouns) when running principal component analysis (PCA). As discussed earlier, having so many proper nouns in the model is only going to lead to overfitting. Filtering on NNP and NNPS reduced the set from 900,180 to 796,588. Aftering building the PCA table, I made a scatter chart showing the distribution of the Document-Component Matrix table with Principal Component 0 on the x-axis and Principal Component 1 on the y-axis. This is another good description of how tightly packed these various industries are.



It's interesting to see the large cluster on the left side. This illustrates the challenge of solving this problem. There is so much similarity and overlap in how different companies describe themselves, even when they are in different industries. Here is the PC1 and PC2 scatterplot:



Moving on from PCA, I use SciKitLearn's CountVectorizer and LatentDirichletAllocation algorithms to create vector spaces and generate topic models. This table shows the top 10 most common words for each industry classification. This is arguably the most important finding from this data exploration project. After doing a quick scan through the table, it becomes clear that there are so many similarities between how companies in different industries describe themselves. This is why the multilabel classification problem for industry classification is so difficult: everyone uses the same words to describe themselves!

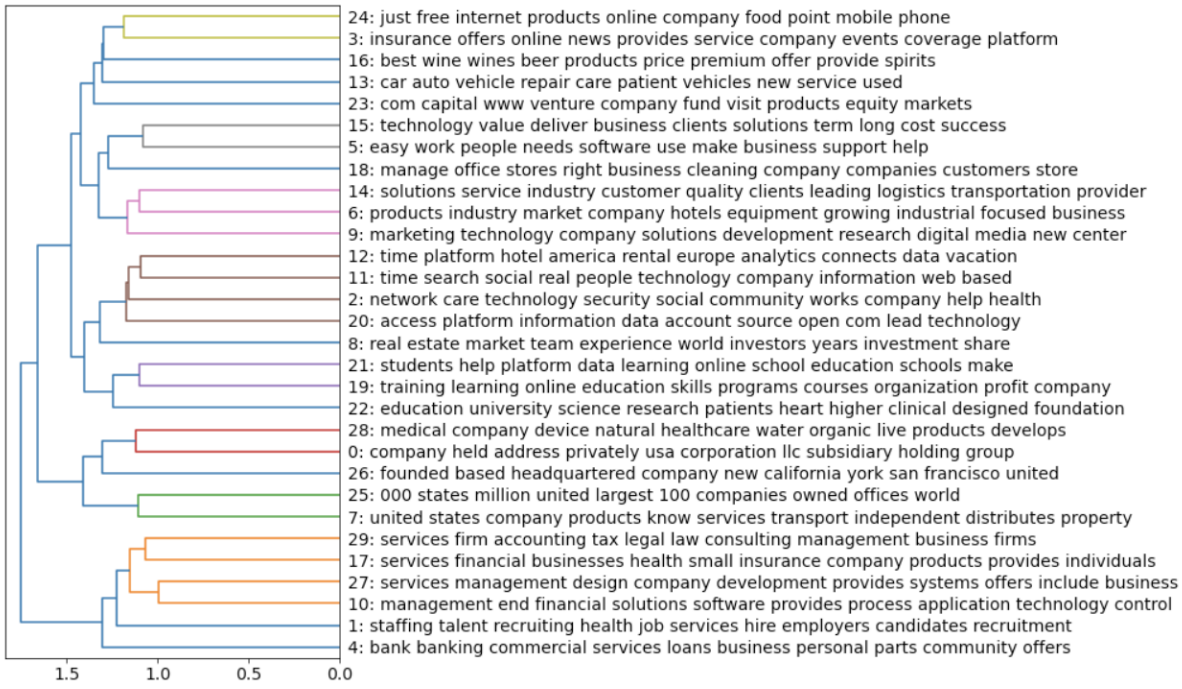
term_str topic_id	0	1	2	3	4	5	6	7	8	9
0	company	held	address	privately	usa	corporation	llc	subsidiary	holding	group
1	staffing	talent	recruiting	health	job	services	hire	employers	candidates	recruitment
2	network	care	technology	security	social	community	works	company	help	health
3	insurance	offers	online	news	provides	service	company	events	coverage	platform
4	bank	banking	commercial	services	loans	business	personal	parts	community	offers
5	easy	work	people	needs	software	use	make	business	support	help
6	products	industry	market	company	hotels	equipment	growing	industrial	focused	business
7	united	states	company	products	know	services	transport	independent	distributes	property
8	real	estate	market	team	experience	world	investors	years	investment	share
9	marketing	technology	company	solutions	development	research	digital	media	new	center
10	management	end	financial	solutions	software	provides	process	application	technology	control
11	time	search	social	real	people	technology	company	information	web	based
12	time	platform	hotel	america	rental	europe	analytics	connects	data	vacation
13	car	auto	vehicle	repair	care	patient	vehicles	new	service	used
14	solutions	service	industry	customer	quality	clients	leading	logistics	transportation	provider
15	technology	value	deliver	business	clients	solutions	term	long	cost	success
16	best	wine	wines	beer	products	price	premium	offer	provide	spirits
17	services	financial	businesses	health	small	insurance	company	products	provides	individuals
18	manage	office	stores	right	business	cleaning	company	companies	customers	store
19	training	learning	online	education	skills	programs	courses	organization	profit	company
20	access	platform	information	data	account	source	open	com	lead	technology
21	students	help	platform	data	learning	online	school	education	schools	make
22	education	university	science	research	patients	heart	higher	clinical	designed	foundation
23	com	capital	www	venture	company	fund	visit	products	equity	markets
24	just	free	internet	products	online	company	food	point	mobile	phone
25	000	states	million	united	largest	100	companies	owned	offices	world
26	founded	based	headquartered	company	new	california	york	san	francisco	united
27	services	management	design	company	development	provides	systems	offers	include	business
28	medical	company	device	natural	healthcare	water	organic	live	products	develops
29	services	firm	accounting	tax	legal	law	consulting	management	business	firms

To really drive this point home, I looped through the TOPICS table and built word clouds for each industry. Here's an example of the word cloud for Administrative Services. There are 29 other word clouds in the jupyter notebook I submitted along with my project.

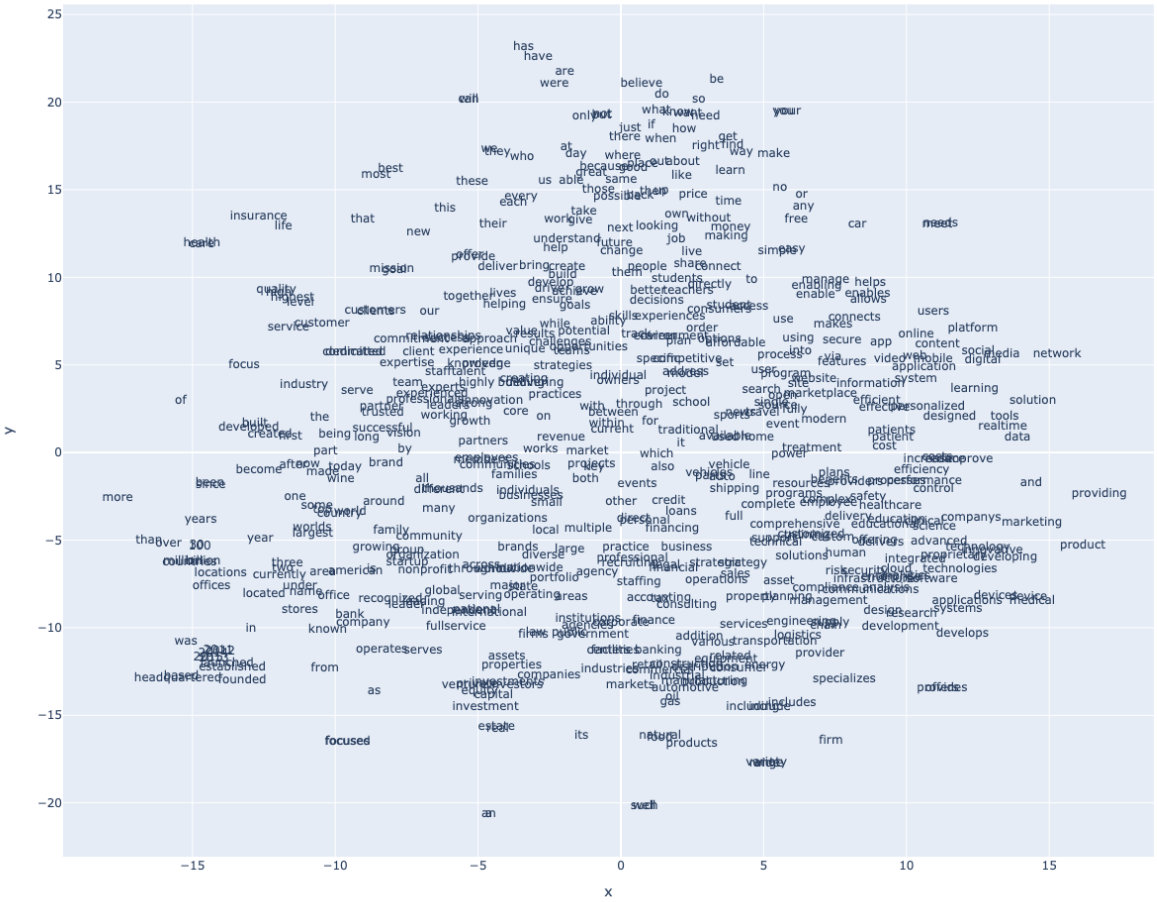
Here I order the topics by document weight. As this bar plot shows, some industry classifications get much more weight than others.

I thought that this scatterplot was another interesting way to look at how some of the industries are clustered. My takeaway from this was that some industries (specifically, 26, 21, 20, 14, 7) should be easy to classify since they have some distance from others.

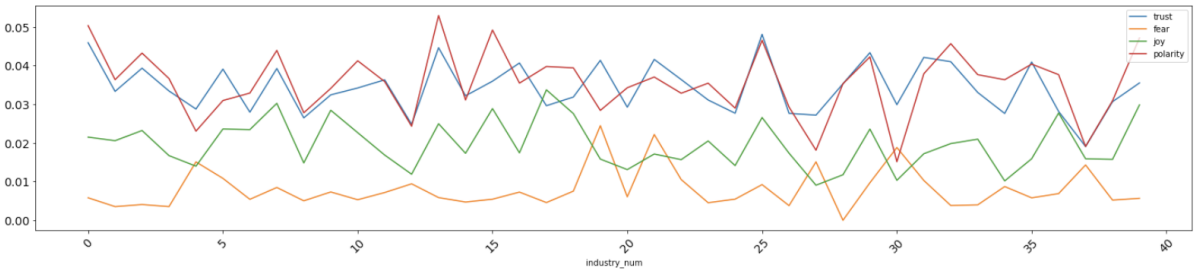
This hierarchical clustering makes the clustering we just saw in the scatter plot even clearer. There are many ways to calculate distance. For this, I chose euclidean. One thought I had that I didn't end up implementing but might be helpful is to use these distance clusters to consolidate the number of industry classifications from 40 down to some smaller number. The problem with that approach is that different distance functions lead to different clusterings, so I'd be putting a lot of faith in that particular distance measure. But the idea to use these clustering methods to help combine some industries isn't a bad idea, I would just need to be careful to cluster industries together in a way that makes sense.



I built a tSNE plot but I'm not sure how helpful it is for me to think about word neighbors across all of the industries in the dataset. If I had made 40 different tSNE plots for the 40 different industries, that might have been more helpful.



I chose to plot sentiment along a graph like this. The downside to thinking about it this way is that it's somewhat misleading. This chart was designed for tracking sentiment across time or across chapters within a corpus. However, I'm using the x-axis to denote the different industries, so instead of a continuous function, this is actually more of a scatter plot where the points are the sentiment scores for each industry and the lines are simply connecting the dots. I'm not sure how useful it is to think about sentiment this way but I was curious to see what it looked like and if I could extract any useful information from these charts. I also used Vader to track the sentiment across sentences in the company descriptions. This was cool to see but I'm not sure how helpful it is for classification. Unsurprisingly, the sentiment scores trend pretty positive. These are descriptions written about a company by people working at the company, so it would be surprising if the sentiments were negative.

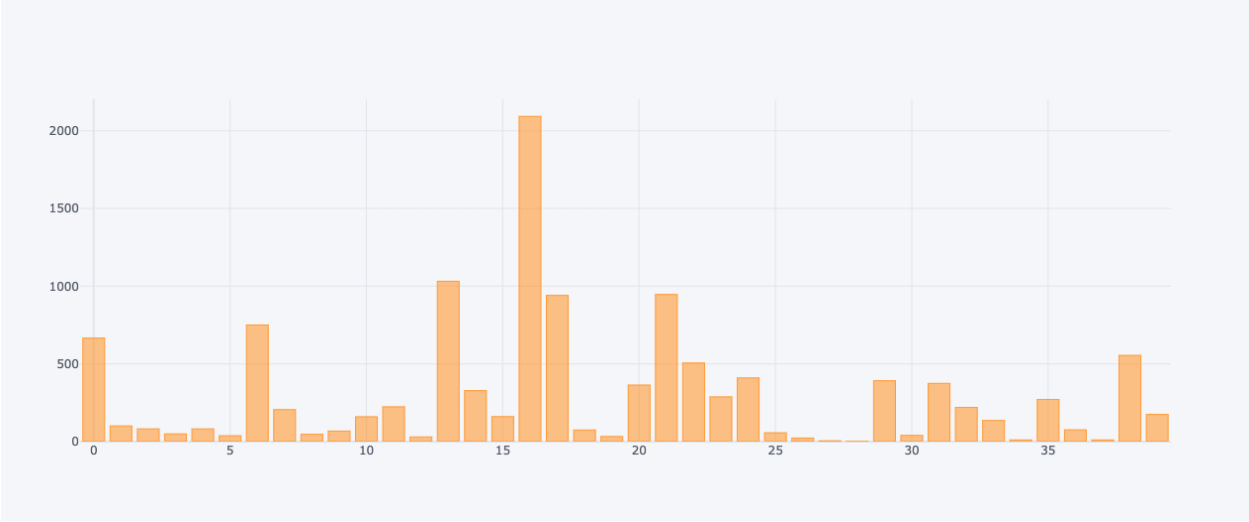


Classification

Using a 70/30 split, I ran a few models to see how they performed on the data.

Multinomial Naive Bayes

The first model I ran was a Multinomial Naive Bayes Classifier. When I fed in different company descriptions to the model to predict the industry classification, it kept on guessing Industry_Index of 16, perfectly exemplifying the problem with working with unbalanced datasets. Industry_Index==16 is by far the most dominant (see bar chart below), which means that the Naive Bayes Classifier was trained more on that than on anything else, ultimately weighting things more as if they were part of Industry_Index==16 than any other industry classification.



Naive Bayes got a 0.3091412742382271 accuracy score.

LSTM Multi-Class Text Classification

I wanted to apply some of what I learned in the DS5559 (Big Data) course this semester into my DS5001 Final Project. We got a brief introduction to Keras and the power of using pipelines, so I tried to implement a few ML models on top of Keras. The first one I ran was a Long-Short Term Memory model with Categorical Cross Entropy. While the model itself was pretty powerful and I enjoyed learning how to use Keras, it wasn't the right approach for the job. The LSTM ended up getting a 0.419 accuracy score.

I wanted to test some examples to see how the model was predicting, so I passed in the description for an Education company and it incorrectly classified it as Manufacturing. I tried a few other examples but that one stuck out in my mind. If it can't correctly classify an Education company where "education" is used multiple times and the words "college" and "universities" are in the description, then this model is useless.

Logistic Regression

I fit a Logistic Regression model with TF-IDF, which scored a test accuracy of 0.4368. That isn't a terrible accuracy score but when you look at the classification_report it becomes clear that this is not the right approach. Industry Groups with low support are almost entirely unrepresented in prediction. There are big mismatches in accuracy and recall, which tells me that some industry groups are being predicted at a rate far higher than others. A broken clock is still right twice per day. If your model just always predicts the industry that is most represented, then the accuracy scores are still going to be pretty high because there are so many companies in that industry classification. This is part of the problem of using unbalanced datasets.

I also fit a Logistic Regression model on the BOW, which had an accuracy of 0.6155. That's actually a pretty substantial improvement.

OneVsRestClassifier

I built a OneVsRestClassifier by building a pipeline with Count Vectorizer, TfidfTransformer, and SVC. This ended up performing quite well, with an accuracy of 0.6476. I attributed its success to the unique binary classification approach of the OneVsRest classifier, but it piqued my interest in exploring how a SVC model would perform on its own.

Support Vector Classifier (SVC)

The SVC applied to TF-IDF had an accuracy of 0.615, while the SVC applied to BOW had an accuracy of 0.55. Which is curious, given that the Logistic Regression model performed better on the BOW than it did on the TF-IDF, the exact opposite of the SVC's performance.

Random Forest

Random Forest scored a test accuracy of 0.555 when applied to TF-IDF and a test accuracy of 0.556 applied to BOW.

Conclusion

It's important to approach classifying industries differently; some will be easy to classify, others will be difficult. The accuracy scores don't tell the entire picture because there were some industries that had near perfect classification and some that had almost nothing correctly classified. This is likely an indication that some industries are being given more weight than others. We saw this in the document weighting chart and is further evidenced by the fact that there are large differences between precision, recall, and f1-scores for many of the underrepresented industry groups.

My big takeaway from this project is that a complex model is needed to solve this kind of multiclass classification problem. Some models are better than others but no model is good enough to deal with this messy and overlapping data. OneVsRest is the closest I got to that, where at each iteration, the possibility of a single industry was compared to all others. More thought needs to be put into each industry classification in order to reliably predict industry classifications for this company dataset. That is the idea behind OneVsRest, so it's unsuprising that it performed better than anything else.