Build Better: The LinkedIn Edition

McCoy Doherty & Nicolas Ortega
& Roberto Guzman - FA 2020

Introduction

As university students studying informatics at the University of Michigan, we are quite familiar with pre-existing major search engines for professional opportunities, such as LinkedIn and Indeed, but we are likewise quite well aware of the shortcomings of these dominant central monoliths of the online job-finding process. So often, those of us actively seeking out our first (or next) part-time, full-time, or intern experience are compelled to parse and invest ourselves in countless applications, but these search engines fall short on having enough filters to facilitate us curtailing our results to our needs, carrying a very real cost in terms of user experience and time.

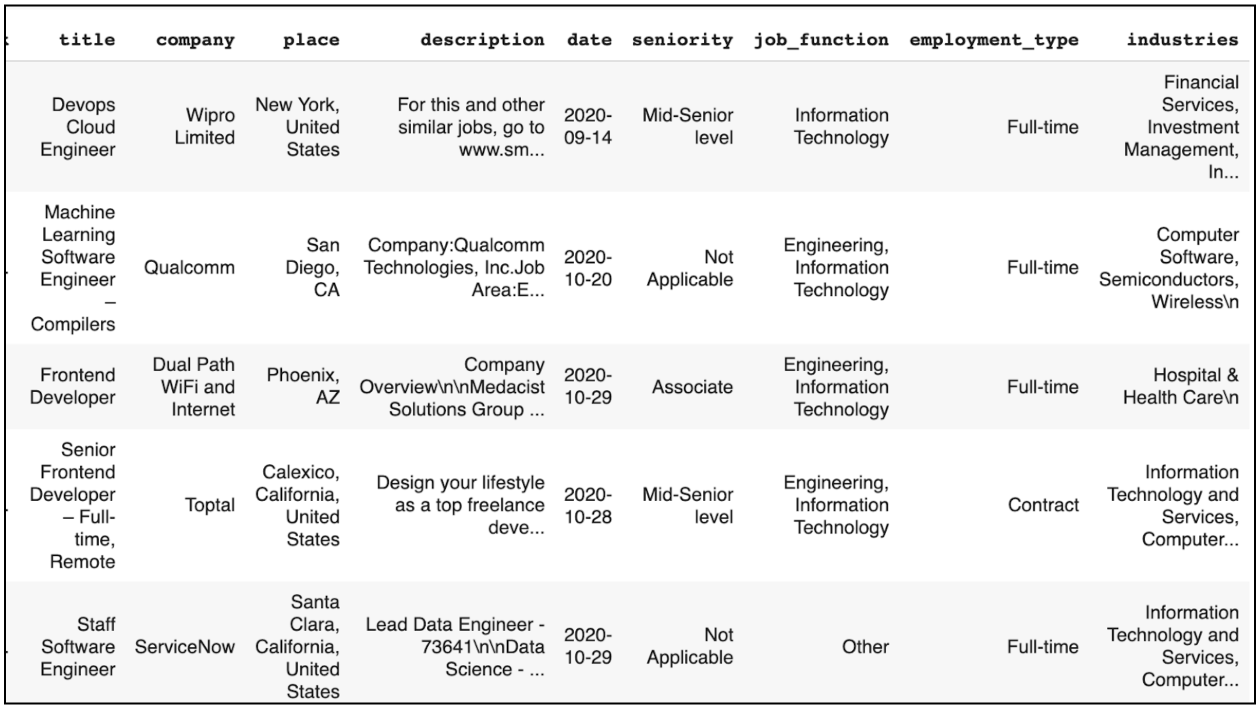
This is perhaps no better exemplified than encountering “entry level” positions that sound exciting, pitch a rather enticing career daydream, only to list in the final lines disqualifiers that rather crudely awaken the readers to the minutes they had just invested, such as the requirement for graduate degrees or a firm statement of unwillingness to sponsor work visas for international applicants. Our intention to remedy this experience led us to reconsider how search engines could be better improved to empower users with more meaningful, popularly-requested filter affordances. And so we decided to, with a focus on technology careers in mind, attempt to engineer a theoretical competitor to the LinkedIn search engine, better tailored for matching candidates to technological positions with fewer exposures to positions from-which they are precluded.

While we admittedly fell short of our loftier dreams for the project, detailed below, our approach to searching through job descriptions incorporated more user-facing affordances in terms of functional parameters that could be be used to minimize poor-fitting positions, as well as a deep learning approach that allowed us to, rather than simply removing stopwords and other conventional cleaning tactics, minimize noise even further by distilling salient keywords from document bodies, allowing just terms of importance to be fed into our scoring/ranking algorithm.

Ultimately, methodological slip-ups in annotation, further detailed in the coming sections, caused great difficulty in the ability to meaningfully interpret the differential impact of our work on, effectively, feature extraction and engineering. However, we were able to build a viable search engine for career opportunities that successfully incorporates filters for potentially-user-disqualifying conditions such as education level or job title, and we were also able to identify a seemingly highly-effective approach in separating critical qualifications keywords from the surrounding noise in job postings.

Data

As detailed in the prior report, the relative sparsity and inaccessibility of APIs for other job-post platforms lead to our narrowing the project to focusing on data that could be scraped from the ubiquitous job-hunt monolith that is LinkedIn. The platform is familiar, massive, and well-populated with a diverse array of jobs across domains and specialties.



title	company	place	description	date	seniority	job_function	employment_type	industries
Devops Cloud Engineer	Wipro Limited	New York, United States	For this and other similar jobs, go to www.sm...	2020-09-14	Mid-Senior level	Information Technology	Full-time	Financial Services, Investment Management, In...
Machine Learning Software Engineer – Compilers	Qualcomm	San Diego, CA	Company:Qualcomm Technologies, Inc.Job Area:E...	2020-10-20	Not Applicable	Engineering, Information Technology	Full-time	Computer Software, Semiconductors, Wireless\n
Frontend Developer	Dual Path WiFi and Internet	Phoenix, AZ	Company Overview\n\nMedacist Solutions Group ...	2020-10-29	Associate	Engineering, Information Technology	Full-time	Hospital & Health Care\n
Senior Frontend Developer – Full-time, Remote	Toptal	Calexico, California, United States	Design your lifestyle as a top freelance deve...	2020-10-28	Mid-Senior level	Engineering, Information Technology	Contract	Information Technology and Services, Computer...
Staff Software Engineer	ServiceNow	Santa Clara, California, United States	Lead Data Engineer - 73641\n\nData Science - ...	2020-10-29	Not Applicable	Other	Full-time	Information Technology and Services, Computer...

Above: Sample Records of Harvested Dataset (URL/ID undepicted)

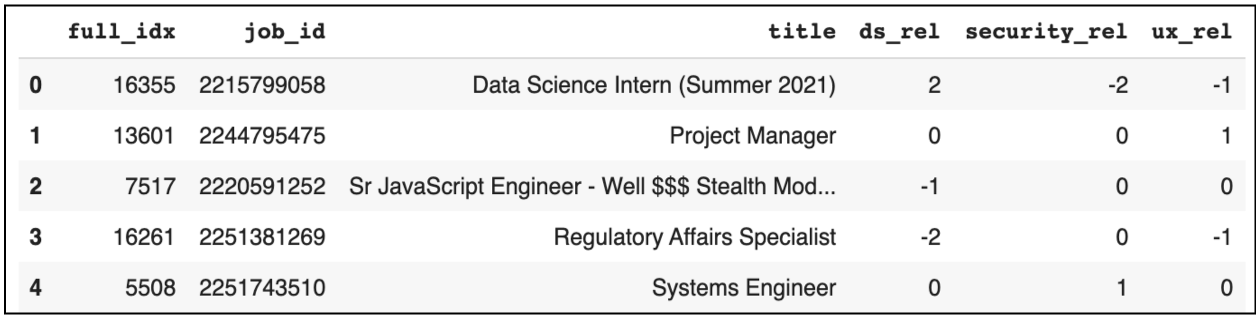
Implementation-wise, we are acquiring our data via an open-source Python package known as "py-linkedin-job-scraper" authored independently by GitHub user "spinludd." This package is being utilized to replicate queries to LinkedIn's job search feature to allow us to gather a corpus of job posting documents that will serve as our dataset for experimenting on and developing our search methods. We jointly curated a list of technical job titles, general keyword terms, and specific skills that would be used as keyword terms by users seeking technical positions involving them. We utilized the "slow-mo" parameter of the package to slow request rate to more ethically obtain the data while minimizing strain to their network resources and, over the course of a night, each ran a loop through all queries, gathering 250 results for each (where the location was "United States"), one of us gathering the first 250 based on LinkedIn's "relevant" ordering and the other based on "recent" ordering. Our script extracted all fields of each job listing (such as ID, title, description, employer, location, field, etc.) and separated them by the cyrillic character "Ж" due to the high occurrence of most traditional delimiter characters and symbols (comma, \t, \n, ~, *, _ , ^, etc.) in the "description" bodies of the job postings we had scraped. We then read the Ж-separated-values files manually into a Pandas dataframe comprised of approximately 16.5k rows and twelve columns/features.

Corpus-Origination Recency & Relevancy LinkedIn Queries				
Data Scientist	Database	Software	Python	C++
Javascript	Compilers	Network Security	Cryptography	Analyst
IT DevOps	Full Stack Security	Backend Developer	Mobile Developer	Web Developer React
Digital Marketing	SEO Social Media	Data Architect	Cloud Engineer	UX
AWS	Kubernetes Docker	Systems Engineer	Project Manager	Agile

Game Developer	Test Engineer	Penetration Cyber Security	Machine Learning	Information Retrieval
Deep Learning	Tensorflow Pytorch	Spark Hadoop	Engineer	Science

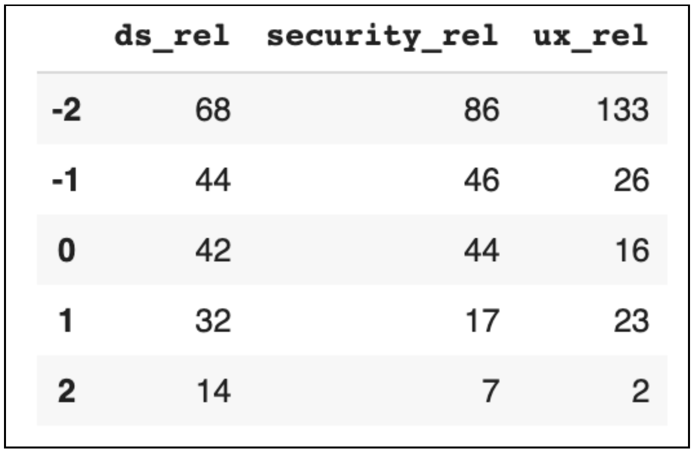
In terms of preprocessing, due to underlying job-mutuality/adjacency of terms queried for, we decided to drop duplicate listings due to simple redundancy, which resulted in the loss of approximately 3.9k records. While most fields exhibit very little noise, the job description field, which is most vital to our intention of developing more sensitive doc-body-based filtration, has the most noise due to being effectively free-text-input fields for posts written differently by each company. Identified content to be stripped included symbols, special characters (\n, \xa0), stopwords, email addresses, URLs, dates (already captured in another field), and remnants scraped from the "show more" / "show less" UI component. In addition to devising regular expressions to handle these issues, we are running terms through a lemmatizer to reduce unnecessary variation of term expressions.

To establish a baseline for relevancy within our data, two-hundred records were sampled from our assembled dataframe at random and dumped to an external csv file which was then manually tagged for relevance using values between -2 and +2, inclusive, to ascertain whether or not they were a good match for one of our baseline test queries such as "data science," "security," and "ux design." A sliver of the resulting dataframe, holding only identifiers, job title, and example relevancy manual scores, can be seen below. A -2 value, such as that for the Data Science Intern position's relevance to security, represents a very poor match for the query, whereas a +2, such as the same position's relevance score to the query "data scientist," represents a very appropriate result.



	full_idx	job_id	title	ds_rel	security_rel	ux_rel
0	16355	2215799058	Data Science Intern (Summer 2021)	2	-2	-1
1	13601	2244795475	Project Manager	0	0	1
2	7517	2220591252	Sr JavaScript Engineer - Well $$$ Stealth Mod...	-1	0	0
3	16261	2251381269	Regulatory Affairs Specialist	-2	0	-1
4	5508	2251743510	Systems Engineer	0	1	0

Additionally, the distribution of manually-tagged relevancy scores for each of the previously-mentioned fields can be found in the image below; overall, there are generally many poor matches, but not as many good/great (1/2) relevancy matchups. Left-to-right, their respective mean scores, over [-2,+2] are -0.6, -0.935, and -1.325.



	ds_rel	security_rel	ux_rel
-2	68	86	133
-1	44	46	26
0	42	44	16
1	32	17	23
2	14	7	2

[Above: distribution of manually-labelled relevance scores]

Related Work

In "Profile Screening and Recommending using Natural Language Processing (NLP) and leverage Hadoop framework for Bigdata" by Indira & Kumar (2016) they describe what, in many ways, is almost the reverse of the process we are doing, but nonetheless provides salient precedent on a comparable process for skill modeling, extraction, and matching/ranking. While their approach involves utilizing resumes alongside job postings, they propose using Named Entity Recognition in their extraction process, as well as POS-tagging to identify adjective modifiers in description to help inflect weights on different skill requirements communicated in job postings. They also utilize extracted education features on resumes to moderate ranking, which we could reverse by taking in user education levels and checking if they are listed as preferred within the job-posting document bodies. This paper is partially responsible for motivating our early (and less-than-fruitful) attempts to attempt using part-of-speech tagging for breaking down job descriptions.

The 2016 paper "A New Approach for Automated Job Search using Information Retrieval" by Tahseen Al-Ramadin and colleagues similarly tackle job-applicant matching by, in having both resumes and job-postings, using tokenization, tagging, and POS methods, but additionally employed synonym checking and looked for clusters of closely-adjacent titles and, albeit on more industry-diverse basis, this could be later adopted into our model to potentially better curtail recommendations relative to arcs of technical professionality, such as security, web and data science. They don't really have quantitative outcomes, and we didn't get an implementation with it, but we'd hoped to experiment with using word embeddings to try to utilize this idea of job-title adjacency to identify similarly-titled job entries to those being retrieved.

In the 2016 paper “Job Information Retrieval Based on Document Similarity,” Jingfan Wang and colleagues propose a two-step system that first uses a vector-space model to determine similarities between queries and job-posting titles and uses those with a thresholded relevance score as seeds for then running a secondary scoring that reranks all documents based on semantic similarity to those seed document identified in the first step. This was a compromise reached after discussing lack of relative overlap between traditionally short job titles and traditionally verbose job descriptions. While conducted in Chinese-language-contexts, their approach seemed to work well, and they identified monogram as more useful features than bigrams and cosine similarity as preferable to Jaccard similarity. We were really inspired by this idea of a “two layer” search and attempted to implement something similar in terms of first matching titles and then attempting to utilize doc2vec to integrate similar documents into our results to broaden recall for relevant documents, but hiccups of understanding resulted in this being scrapped due struggles integrating this endpiece with our code.

As is the case with prior papers, “End-to-End Resume Parsing and Finding Candidates for a Job Description using BERT” (Bhatia et. al) focuses on resume parsing alongside job-description parsing for matching, but the most interesting component here was that, in addition to using BERT multiple times during in their approach, as we were exposed to in the most recent class lecture, they repeatedly emphasized using it for “sequence” and “sequence-pair” classification to gauge similarity between one's reported job experiences and a given job description. While we didn't end up utilizing BERT, this paper inspired our reflections on how Regular Expressions would likely not be the kind of flexible solution we would need for handling skill-based extraction from documents and later led us to employ a different flavor of unsupervised learning approach in the form of Spacy's

Named Entity Recognition system, which we found to be surprisingly recall-effective, despite occasional false-positives.

In "Benchmarking of a Novel POS Tagging Based Semantic Similarity Approach for Job Description Similarity Computation" by Mondal et. al, they use job description similarity as a basis to compare LDA, doc-to-vec, and a novel doc-comparison method they devised as “POSDC,” which they find exceeds LDA and doc-to-vec “in isolation,” or without a larger corpus of sample data. Their novel POSDC model uses POS tagging to decompile job descriptions into verb-object-attribute sequences, compared for similarity versus one’s reported experiences using “synsets” and Wu-Palmer similarity. While we found the results of the paper greatly compelling, we mutually struggled with our backgrounds and lack of ability to meaningfully interpret the formulae provided / convert said formulae into our own methods, and struggled at working meaningfully with synsets.

Last but not least, in "Job Skills Extraction With LSTM and Word Embeddings," Nikita Sharma discusses their experiences trying different models for extracting skill expectations from job descriptions with different methods such as topic modeling, word2vec, word embeddings, and, most successfully, LSTM with word embeddings. The reasons for this deep learning approach’s relative success (high accuracy) was argued as inheriting from modern job postings having relatively pseudo-standardized grammatic tendencies. In addition to attempting more regular-expression-oriented routes, this study inspires us to plan on looking heavily at LSTM combined with word-embeddings as a major starting option for handling skill-based extraction to support our search engine’s development. The author also acknowledges that their very small dataset size limited their neural-network-approach’s learning capacity, but we found that with our larger corpus of documents, we were able to achieve reasonable success at extracting entities (primarily software languages and skills) from our documents for reasons we believe align with the arguments of this paper: while not entirely a grammatically complete (full-sentences) context, there is pseudo-standardized language commonly employed in modern online job posting descriptions, making it easy, we believe, for deep learning models to identify common anchors where linguistic patterns emerge.

Ultimately, most pre-existing work in the domain seems to integrate additional documents, such as one's resume, cover letter, or LinkedIn profile, allowing for better comparison between the job descriptions and the candidates' experiences. While we had no interest in comparing job posts to candidates' experiential documents, we initially had intended to design an approach for search parameters to better incorporate enumerated degrees of proficiency with particular languages or technologies in the form of years or vocabulary expressions to facilitate better tuning qualifications to experiences as part of the search process. Implementing this remains a "reach goal" for achievement after the semester, as we believe it would be most meaningful combined with sensitive extraction of "minimum" and "preferred" qualifications and extraction of years-of-experience / proficiency requirements from description bodies, which we now have a better roadmap for approaching, but one that was formulated too late for this to be achieved within the semester.

Ultimately, these papers provided us many avenues for our consideration to solving various aspects of the search engine development process and, while we sometimes failed to mount the shoulders of giants, we were still able to meaningfully architect solution attempts from the trails left in their wake.

Methods

After getting our data scraped and established as a conjoined CSV file and imported as a Pandas Dataframe through the read_csv method, we divided our search engine building process into two phases: a) retrieval on our ground truth dataset for empirical evaluation, and b) using entire corpus to index using BM25 (and variants) to get (hopefully) relevant results. We used an existing library for ranking using BM25, written by dorianbrown and found on GitHub.

In order to aid our process, we defined several helper functions that will allow us to employ various NLP techniques to preprocess the textual information such as job title and length job-description passages, as well as specific functions to compute relevance scores for our given corpus and retrieving all relevant documents, according to our model. All of these helper functions can be found documented and defined in the project notebooks adjoining this report.

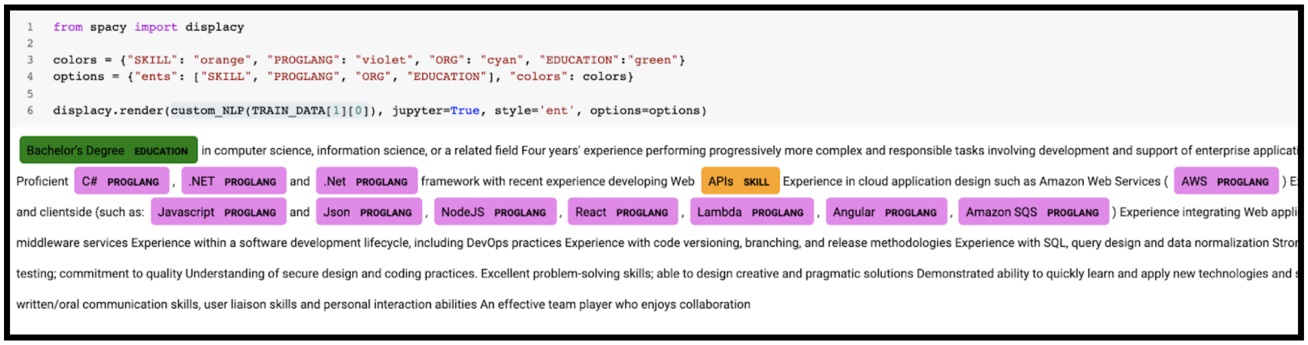
While the majority of these functions were simply for standardization of case, removal of symbols, or using Pandas and regular expressions to detect if conditions such as education level, immigration requirements, and other potentially key words seemed expressed in the features, the most interesting and involved feature was our training of a Named Entity Recognition system.

Named Entity Recognition

While experimenting with regular-expression-based approaches for extracting skills in job descriptions, it was recognized that, for the sake of wide-spectrum identification of contained skills, some type of unsupervised learning system would likely be more appropriate and performant than manually establishing a lengthy list of regular expressions intended to encapsulate all major skills of a variety of domain and specialties. This led us to utilizing the Spacy (2.0+) natural language processing library to facilitate the training of a named-entity recognition system. Our interest in doing this was to achieve a more effective distillation of the most salient and meaningful components of the often-lengthy, often-noisy description fields of online job postings. Rather than more classical "remove the stop words and symbols" preprocessing, our hope was to be left with almost strictly keywords.

We were able to instantiate a standard Spacy NER pipeline, a tokenizer that tags and parses full-string data representations before feeding it into a fully-connected one-dimensional convolutional neural network and, rather than utilizing the pretrained default model, begin with a blank model that we could train ourselves.

In terms of training our named entity recognizer, the required data structure is a list of two-element iterables, the first element of which is a parseable string of text and the second of which is a dictionary keyed to contain a list identifying the starting and stopping indexes of each target entity in the full textual body, as well as what label it should be affiliated with.



```python
from spacy import displacy

colors = {"SKILL": "orange", "PROGLANG": "violet", "ORG": "cyan", "EDUCATION":"green"}
options = {"ents": ["SKILL", "PROGLANG", "ORG", "EDUCATION"], "colors": colors}

displacy.render(custom_NLP(TRAIN_DATA[1][0]), jupyter=True, style='ent', options=options)
```

```
Bachelor's Degree EDUCATION in computer science, information science, or a related field Four years' experience performing progressively more complex and responsible tasks involving development and support of enterprise applicati
Proficient C# PROGLANG , .NET PROGLANG and .Net PROGLANG framework with recent experience developing Web APIs SKILL Experience in cloud application design such as Amazon Web Services ( AWS PROGLANG ) E
and clientside (such as: Javascript PROGLANG and Json PROGLANG , NodeJS PROGLANG , React PROGLANG , Lambda PROGLANG , Angular PROGLANG , Amazon SQS PROGLANG ) Experience integrating Web appli
middleware services Experience within a software development lifecycle, including DevOps practices Experience with code versioning, branching, and release methodologies Experience with SQL, query design and data normalization Stror
testing; commitment to quality Understanding of secure design and coding practices. Excellent problem-solving skills; able to design creative and pragmatic solutions Demonstrated ability to quickly learn and apply new technologies and s
written/oral communication skills, user liaison skills and personal interaction abilities An effective team player who enjoys collaboration
```

We were able to jointly, manually establish a training corpus of 63 tagged documents of varying length, representing the labels “ProgLang” and “Skill” for identifying general tech skills and specific programming languages or libraries, as well as, more experimentally, “Org,” “Education,” and “NonSponsor” to identify organization names, degree expectations, and potential expression of unwillingness to sponsor worker authorization such as H-1B visas. The trained NER model enabled us to add an additional feature to each job listing in the Pandas dataframe, assigned to contain the entities recognized by the NER model. We were then able to append different fields/columns to achieve a more distilled set of tokens and terms to meaningfully represent each document.

Ground Truth

To evaluate our model performance, we first read our ground truth data CSV and the full dataset in order to merge them based on our ground truth annotation availability. After merging, our corpus consisted of 200 job postings with the original features included in our scraped data, alongside annotated relevance scores given three queries: 1) *“data science”*, 2) *“ux design”*, and 3) *“security”*. This facilitated our ability to utilize mean average precision (at n=20) to evaluate the performance of different algorithms on the data subset, differentiated across three different technical specialties.

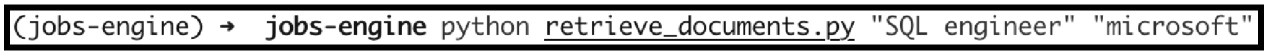
Pre-processing consisted of proper formatting of already existing columns. These included cleaning the title and description of a given job posting using pre-defined helper functions, as well as concatenating different text columns within our data to create new text features and test out our model’s performance given different processed text to index. At this point, we also used our spaCy NER model to extract entities from our job descriptions. In the end, after getting our data into the proper format, we created new dimensions by concatenating these pieces of text, and ultimately defined several variations to be defined as the corpus to be fit into our model, including: *job title, job description, job title+description, entities, title+entities, and company name+title+entities.* We chose to engineer these new features because of the differing performance of the BM25 variations given the document length. Our initial hypothesis was that, as we concatenate more text features to create new dimensions, the more appropriate it'd be to use BM25L to normalize document length. Further, we intuited that an improvement in performance could be achieved by reducing the noise of our already pre-processed job description, and our attempt in doing so can be seen in our entity extraction attempts using a custom-built spaCy model.

In order to observe the performance of our retrieval model, we used the previously defined dimensions in our data to fit into our model as the corpus, and further, we tried out three different variations of the BM25 algorithm, including 1) *BM25 Okap*i, 2) *BM25 Plus*, and 3) *BM25 Long*. Using every one of these variations, we fit the previously defined corpus variants to observe the performance of our retrieval model. Our results will be discussed in a future section, but we observed that using the company name, job title, and entities extracted from the job's description and the Long variation of the BM25 algorithm yields satisfactory results.

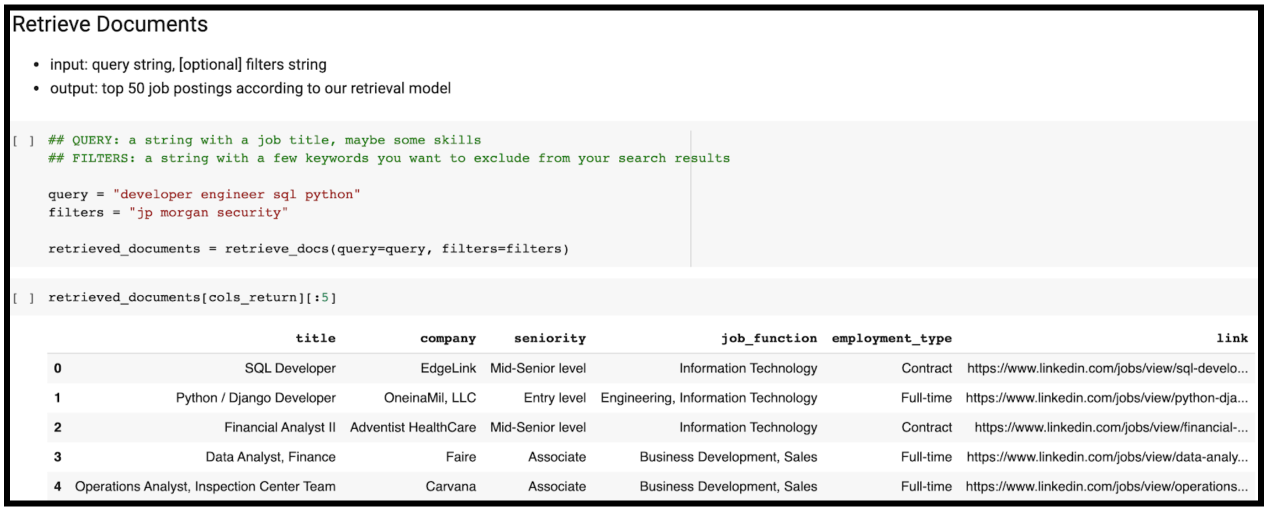
Retrieval of Whole Corpus

Once we have observed and intuited certain results from our experimentation with ground truth data, we determined that using BM25L and the "company+title+entities" dimension was the path forward when fitting our entire dataset as a corpus to be indexed by our retrieval model. This finding supported our initial hypothesis that including more text (i.e. using a longer string) and reducing the noise in our dataset using our entity extraction model would prove beneficial and may be the best performing variation of our model. Overall, the process of building the retrieval model was similar, although we now use our entire corpus (12,546 documents) to be indexed, rather than our 200 document ground truth subset.

Further, in order to make our retrieval engine better for the end-user, we decided to implement a filtering mechanism in order to exclude certain keywords that may appear in our “company+title+entities” column. For example, given a query “data science”, the user will also input another string with the keywords to be excluded from our results, like “Geico engineer”. Effectively, the user would be searching for data science jobs that are not in Geico, and not an engineering position. We hope that this affordance can aid end users in avoiding unwanted positions and therefore increase, if not empirical relevance, at least relevance for the end user. Although we cannot empirically evaluate the performance of the model fitted with the entire corpus obtained, we see satisfactory results when providing a query, and even better results by using the filtering affordance included in our retrieval model.



```
(jobs-engine) ➜  jobs-engine python retrieve_documents.py "SQL engineer" "microsoft"
```



Retrieve Documents

- input: query string, [optional] filters string
- output: top 50 job postings according to our retrieval model

```
[ ] ## QUERY: a string with a job title, maybe some skills
    ## FILTERS: a string with a few keywords you want to exclude from your search results

    query = "developer engineer sql python"
    filters = "jp morgan security"

    retrieved_documents = retrieve_docs(query=query, filters=filters)
```

```
[ ] retrieved_documents[cols_return][:5]
```

	title	company	seniority	job_function	employment_type	link
0	SQL Developer	EdgeLink	Mid-Senior level	Information Technology	Contract	https://www.linkedin.com/jobs/view/sql-develo...
1	Python / Django Developer	OneinaMil, LLC	Entry level	Engineering, Information Technology	Full-time	https://www.linkedin.com/jobs/view/python-dja...
2	Financial Analyst II	Adventist HealthCare	Mid-Senior level	Information Technology	Contract	https://www.linkedin.com/jobs/view/financial-...
3	Data Analyst, Finance	Faire	Associate	Business Development, Sales	Full-time	https://www.linkedin.com/jobs/view/data-analy...
4	Operations Analyst, Inspection Center Team	Carvana	Associate	Business Development, Sales	Full-time	https://www.linkedin.com/jobs/view/operations...

Above: Examples of the CLI Interface and ipynb-invoked search results

Evaluation & Results

To evaluate the performance of our search engine, we established a sense of ground truth by manually annotating two hundred entries, constituting a subset of our full dataset, and giving each a score from -2 to +2 for their relevance respective of a given query, of which we selected three, retrospectively-very-insufficiently-variant search terms: Data Science, UX Design, and Security. This manually-annotated subset of two-hundred job postings was used to establish a sense of ground truth. While we produced these annotations rather early in the process and forgot to include more theoretical query terms that were more inclusive of keywords aligned with corporate entity names and specific skill terms (such as “wireframing” or “x86 assembly”), meaning that our results are unfortunately less meaningful than intended for reflecting potential improvement due to extracting skill-centric features due to that oversight.

For measuring performance, we elected to utilize mean average precision of the top twenty returned results as our primary metric of focus due to the interpretability of the metric and to avoid mistaken or odd model performances when using precision @ k. Further, although our annotations are ranged [-2, 2], we considered anything greater than 0 relevant, else, irrelevant for our mean average precision computation. We found that averaging precision scores for a given query resulted in a more ‘realistic’ performance outcome. For each of the three queries for which we performed manual annotations, and for each algorithm variant (BM25 Okapi, Plus, and Long), the mean average precision was taken based on comparing query string to the position title, description, description with title appended, and also with the entities extracted by our named-entity recognition model in isolation, appended with job title, and appended with company name; our results are recorded in the tables below:

Mean Average Precision (N=20) for Query: "Security"						
	title	**description**	**title + description**	**entities**	**title + entities**	**company + title + entities**
BM25 Okapi	<u>0.986111</u>	0.602005	0.628869	0.581481	<u>0.70123</u>	<u>0.70123</u>
BM25 Plus	<u>0.986111</u>	0.602005	0.628869	0.581481	<u>0.70123</u>	<u>0.70123</u>
BM25 Long	<u>0.986111</u>	0.618885	0.655628	0.581481	<u>0.70123</u>	<u>0.70123</u>

The security query rendered our most unwavering results, with surprising degrees of consistency in all scored ranges aside from a slight improvement in the BM25 long variant’s performance in the description-only iteration, which seems relatively intuitive due to the substantial length of description-feature lengths. We believe that the mean average precision is strongest in titles due to a trend of the term “security” having very high organic occurrence in the job titles of security-subsector positions, rendering it relatively difficult to gather much from the results of this particular query.

Mean Average Precision (N=20) for Query: "UX Design"						
	title	**description**	**title + description**	**entities**	**title + entities**	**company + title + entities**
BM25 Okapi	0.611111	0.08125	0.084967	0.24359	0.52549	0.52549
BM25 Plus	0.611111	0.08125	0.084967	0.24359	0.52549	0.52549
BM25 Long	0.611111	0.08388	0.084967	0.24359	0.35882	0.35882

In terms of the UX Design query results, our strongest performance is again, per model, observed in the title-only performance, however secondarily followed by title-plus-entities performance, where entities were those extracted by our named entity recognition model. We believe this is again due to accidentally underthinking the degree to which our queries would correlate with titles due to underrepresentation of non titular, skill-centric terms in our search results.

Mean Average Precision (N=20) for Query: "Data Science"						
Corpus Variations >>>	**title**	**description**	**title + description**	**entities**	**title + entities**	**company + title + entities**
BM25 Okapi	0.767529	0.72342	0.740322	0.470652	0.599992	0.612404
BM25 Plus	0.776036	0.501006	0.540414	0.529967	0.64682	0.649084
BM25 Long	0.767529	0.688453	0.70947	0.741892	0.891453	0.891453

The Data Science query is that for which we observed the strongest performance differentiation and, while models that focused on title alone were the most effective for the Okapi and Plus variants of the BM25 model, our overall net strongest performance was for that of the BM25 Long variant acting on title combined with extracted entities, with or without adjoining company name data.

The substantial variance of performance from query to query, while most exhibit optimal performance on title-alone-basis makes us consider that the substantial variation between these models could be attributable to both the degree to which query terms are reflected explicitly in the job-post titles alone. In a sense, using only the title column could be “cheating” and is not really leveraging the capabilities of the BM25 algorithm. Further, in the final model, if we had used only the title text (given that it returns the highest mAP scores), we couldn't have leveraged a filtering option since only the title of the job is considered. We definitely expected mAP scores to decrease as we added text to the corpus, but it was also our hope that adding "cleaner", less noisy text would benefit the model, particularly using the Long variation of BM25.

Discussion

As we delved deeper into our model building phase, we began realizing the potential improvements we could have made by scraping a more balanced dataset and creating a more balanced ground truth subset, keeping in mind the distribution of both score and types of job listings. Our best model performance uses job title, company name, and extracted entities, but this was not the best performing model for all three of our ground truth queries ("data science", "ux design", "security"). We did see a really good performance for data science, though. We suspect that too many jobs may contain the keywords/skills/entities needed for a data science or analyst job. Further, if we had devised a better sampling strategy, we believe we could have avoided this discrepancy in model performance given the query. Our initial sampling consisted of entering the previously mentioned queries into the LinkedIn scraper, collected all the data, and the randomly sampled 200 documents, not taking into account the imbalance of job categories.

Overall, our model performance was relatively satisfactory, but only for the "data science" query. This makes sense in hindsight. Model performance for the other queries was not great. At first glance, it may seem that all of our variations perform the best using only job title, but in a sense this is just the baseline because, realistically, people looking for jobs are going to want to filter out certain postings they know they don't want, as well as learning more about a job than the name of the role itself. Further, by looking at the title performance across all three of our queries, we can get a sense as to how frequently any word in our given queries appear in the titles of the jobs themselves. Ideally, we would have expected a relatively similar performance across queries (if imbalance was not an issue). Currently we see a very high mAP for "security" using only the title, but this would most certainly have decreased if we used a slight modifier on the query like "security analyst". For this reason, if we consider fitting BM25 with only the title text as the baseline, we only beat it on the "data science" query (albeit significantly).

We do see promising results using entity extraction, although the temporal investment of making manual annotations to create a spaCy model from scratch is quite high, especially when compared to other forms of annotation, due to the requirement of explicitly noting all entities, decisively establishing which labels they should belong under, and perhaps most time-consuming, the requirement to specifically provide the starting and ending character indexes of the training string to establish the barriers of the entity. It would be interesting to continually train the model to include more granular tags, such as decompiling Education into Bachelors, Masters, PhD, and Certificate label representations, especially due to the prolific nature of certification-based credentialing in the cybersecurity subdiscipline, which are often textually rendered as capitalized abbreviations that could be considered easily mistakable for programming languages, a possible complication in further development of entity-extraction model training.

While the time investment and great care for detail create a clear bottleneck to the entity extraction approach we have selected in spaCy, we believe such improved granularity of extraction could create scaffolding upon which one could very easily transfer said granularity into the filter-affordances of the engine we hope to continue developing. In closing this section, we feel we have achieved setting the scaffolding for our search engine, and solved perhaps the most significant ambiguity to the process, however we feel a mere day short of time to correct our annotations / performance benchmarking missteps and, consequently, struggle to definitively report quantitatively the success (or lack thereof) from our methods, but nonetheless feel very much has been personally learned and reinforced in the course of this project.

Conclusion

In summary, we began this project with two primary intentions to “solve,” the first of which was to develop a search engine for job posting information that would be able to return reasonable result quality in a way that supports filters for features often hidden within document body-text and thus not filterable in existing competitors (such as Indeed or LinkedIn) and, pursuant to that, we became equally fixated on the search for a method or framework through-which we could effectively distill the most salient keywords from these relatively verbose textual bodies communicating expected qualifications.

While our methods retrospectively leave much to be desired, recognizing our ground truth is established by annotations near-reflective of our goals but tragically not overlapping with where a meaningful difference would actually be observed: query terms with greater representation of specific practical skills, we were still able to establish a functional search engine for our corpus of scraped data and implement filters that seek to remedy the issue of hiding from users the job posts that embed qualification positions that would be pre-disqualifying. We were also able to, albeit last-minute and insufficiently measured, able to identify what was observationally appearing to be a relatively effective approach for distilling these key terms / qualifications and separating them from the otherwise noisy textual environment present in the job descriptions.

Our many lessons learned and intended ambitions moving forward are covered in the sections to come, however our grandest takeaways from this project are that our methodology could have improved greatly from further preplanning, but that we have identified a framework for the extraction of qualification information that appears both elegant and effective, which we believe could be effectively embraced for extracting not only skill keywords but also other qualifications.

While our methodical missteps may have rendered it difficult to gauge the true impact of our work in terms of ranking algorithm performance in cases where query-string parameters are intended to match beyond job-title-text, identifying a well-functioning approach for entity extraction often felt like half of the battle, and consequently was the sole focus to-which half of our team was allocated. The Spacy-based approach we identified to custom-training a natural entity recognition model appears a greatly-flexible solution that could be reconfigured further to differentially extract labels for specific degrees of education, as well as potentially extract skills differentially listed as either preferred or minimum qualifications, which could be incorporated into future user-filters and be extracted as separate features (where applicable).

The Code Graveyard: Gone but Not Forgotten

Along the duration of this project, we had a few different attempts at methods that didn't quite work out. Perhaps our largest broken time investment was in an attempt to recreate the Jingfan Wang (and colleague)'s dual-layer search method where search is first conducted with focus on job titles and then using doc2vec in order to find similar documents. The intention here was that the first search would use TF-IDF or a BM25 variant to yield documents that effectively act as a definition for what a given job type should "look like" and, after taking the top N results to broaden said definition, doc2vec would have acted as a method to find documents labelled under different job titles but with similar skillsets, effectively a pursuit of broadening recall. While we were able to get TF-IDF and BM25 variants working for the front half of the pipeline, and eventually got word2vec and doc2vec working in isolation, we had issues figuring out how to get it properly incorporated and began exploring alternative options.

Early in our project, a substantial degree of time was spent on attempting to devise regular expressions to identify skills and better extract them from documents before we eventually accepted that a more complex but scalable solution would be required in order to flexibly handle the myriad of technical skills that exist or could exist in the future. While most of these skill-targeting expressions were simple and effective, they were a brutally hardcode-esque solution that constantly needed updating when new documents reflected more forgotten/missed skills. We also experienced some weirdness with a regular expression initially written to identify URLs within job description bodies (for deletion/cleaning purposes), by which we mean we'd observed problematically-long runtimes, which we attribute to document length and the amount of conditionals in the way the statements were structured. Ultimately, regex statements for things like URL detection/removal and removing Equal Opportunity Employer legal statements that mark the end of countless posts were recognized as relatively unuseful in the structure of our final approach and were effectively sunk/wasted implementations.

Last but not least, some ironically blessed buggy behavior with NLTK is what actually led us into working with Spacy. When attempting to investigate part-of-speech modeling more carefully, intending to pseudo-replicate part of Mondal's approach in effectively isolating verb-noun / verb-adjective-noun phrases, hoping to decompile descriptions into sets of statements like "implementing - scalable web applications" and "designing - accessible - user interfaces," we were attempting to utilize a specific functionality within NLTK that involved chunking documents into these types of structures based on a given part-of-speech tag schema, but observed some strange issues. While we primarily worked on Google Collab to facilitate collaboration, running the specific function as shown in the documentation was throwing errors with traceback and outputs that we were not able to meaningfully debug. On attempting similar code to the documentation, but run in a local environment, different issues were encountered revolving around something called "ghostscript," which we were similarly unable to debug.

Lessons Learned & What's Left

Honestly, as our final final project of the semester, born primarily of caffeine, anxiety, and chronic sustained burnout, we recognize that there are many, many things we could have, and wish that we would have, done better under less chaotic schedules, including implementing better solutions, recording more data about performance of different components and approaches, and improvements to data used.

In terms of the implementation routes, given more time, we definitely would've liked to have, as previously mentioned, tested the effectiveness of Doc2Vec in a Jingfan-esque multi-layer search approach. We also would have liked to have done more work in reconstructing documents into grammatically-parsed chunks, but were afraid of over-investing time into fixing our local-and-cloud issues observed with the particular NLTK function we had problems with and instead moved onto Spacy for a different approach due to time constraints.

In terms of improving our data itself, given more time, we would have expanded the size of the data subset that received manual annotation to support our establishment of a sense of ground truth, both in terms of the count of records and expanding to encapsulate a few more different professions. Regarding ground-truth annotations, we believe we would have observed more meaningful differentiation between methods/features incorporated if search terms contained more skill-based terms (such as "data scientist scala" or "software developer c++"). As our first experience really building a search engine project from the ground up, this is easily our largest project take-away: more than any other domain we've experienced, the manual annotation aspect (and even, to some degree, NER training data manual-creation aspect) makes careful pre-planning of methods so much more important than was the case in our prior experiences. By the time we collectively realized our shortcoming in this regard, it felt too late to go back and reconstitute annotations due to time crunch.

This was additionally our first experience working with entity recognition systems as we did in this project and, as a result of our initial experiences, we are recognizing that we likely could have provided better training data for training the NER model in Spacy, potentially teaching it to extract more categories than simply organizations / programming languages / skills / education. We retrospectively recognize that we could have probably trained it to handle tasks such as extracting specifically references to bachelors / masters / doctoral degrees (rather than relying on regex), identifying sections differentiable as "minimum qualifications" versus "preferred qualifications," and noting any explicit security-clearance or citizenship requirements.

We also believe that, with respect to the entity-recognition training process, we would have benefitted from comparatively exploring the performance of utilizing a pretrained model with the additional programming / skills labels added rather than training from the ground up, which we believe would have helped minimize misclassification of other textual entities as programming languages or skills, something anecdotally witnessed when visually inspecting performance with the "Displacy" rendering tool. We also believe that we could have potentially augmented the recognition pipeline by adding a rule-based matching provision set to help take terms commonly mislabelled and post-label correct them by identifying them with regular expression patterns and mapping their label to "None."

Last but not least, we recognize that we could have been better about collecting performance data to quantify the degree to-which our entity recognizer was achieving recall in both labelled and not-manually-labelled data, to better understand if performance was better achieved with longer-passage training data, shorter-passage training data, or a mix, as well as settle the debate between potential performance of a pre-trained model versus a trained-from-blank model. We also think it could have been worthwhile to consider attempting to construct a better baseline with our differentiating "star feature," the filters, in mind, such as comparing in the Top K returns from LinkedIn, how many had certain requirements of disqualification conditions, so we could better attempt to quantify the degree to which our approach, allowing filtration of such listings, actually would pull redundant listings out of the users' view.