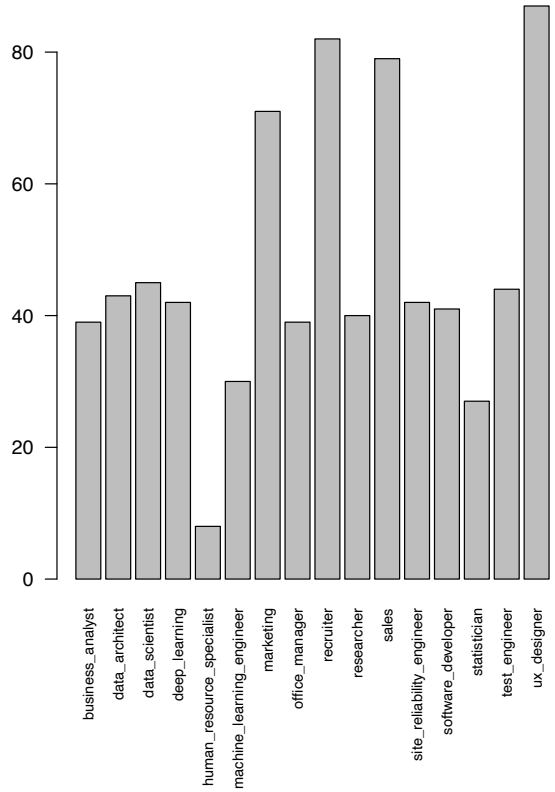
Zachery Macintyre
Nikhil Gopal

Intro:

No matter what point you are in your life you have applied for a job, or will be applying for a job. It can be a long and tedious process, that involves reading through many roles and job descriptions, and the worst part is most might not even interest you. On the flip side, all companies post job descriptions about open positions, and this inevitable it leads to more applicants than they can possibly go through. So, they also need a way to weed through the unlikely candidates. In our project, we create a matching tool that matches your resume to job descriptions, and do it in ways we think companies might match potential employees to roles. This had led to a few insights about how you should write your resume. However, it must be noted this is our best attempt to optimize your chances, and does not guarantee a job. We are very hopeful that it will help the process though.

Data Description:



For the data we used indeed.com job descriptions and info scrapped by our professor Dr. Wayne Lee. The data was broken down by Job, State, Employment type, Description, and Year. In total we were dealing with a little over 500 jobs. As seen in the chart on the left, the data clearly is not evenly distributed. However, for our methods it is not a huge worry because our matching process does not require tons of data.

On to the actual data, and our cleaning techniques. The cleaning of the job descriptions was handled by Zac, and he noticed a few abnormalities that could be fixed. The first being that there were many instances of sentences that would end and a new began without any sort of spacing. For example, “engineer.The” and this could cause problems later on, if not fixed. So, we fixed it by removing any periods and replacing them with a space. Outside of that, the data was relatively well structured for what we were trying to

accomplish in the project. The only other things we did for data wrangling was creating different data frames for different applications. For example, we created a list that mapped job titles to the average number of words in total descriptions for the title. In other words, we got the frequency of words that appeared in every job title, such as data scientist or office manager. Then, we summed those frequencies and divided by total descriptions to get the average frequency for job titles. We additionally made a few more data frames for other instances, in which it we thought it would be helpful. All of those follow similar logic to the thought above.

Methodology:

First, we must discuss the inherent problem and complications that arise in natural language processing (NLP) problems. In many machine learning techniques, the data used is purely numerical, and if not, there are clever ways to change it to a numerical form, such as one hot encoding. However, in NLP problems, words and phrases are what is important. It makes methods like linear regression, or logistic regression more challenging to use. On top of that, we are not trying to predict a continuous variable, we trying to decided best matches for a job based off your resume. We did this by trying out 2 different methods. The first was a comparison between a naïve method and a term frequency – inverse document frequency. This gave us job title matches based on your resume, and then we used a similar thing to get the job descriptions that best matched. For our second method we used a cosine similarity matrix to evaluate matches. The matches were calculated on a distance type score, with an additional diverse feature added. This made sure you got good matches to your resume, but that they were also not all for the same exact thing.

Method 1.

Now we are on to the actual way we solved this problem. Our first method used a comparison between a term frequency – inverse document frequency (tf-idf) method and a naïve method. The naïve method used the average frequency of words that appeared in each job title as discussed above. The argument for this was that in a job description you only have so many words to use and want to make them count. So, in a description for a statistician or data science the word data might come numerous times, and that indicates the importance in that line of work. In the tf-idf method we used term frequency and the inverse document frequency to get important words. The argument here is this is a very common method to get important words in a document when dealing with NLP problems. Initially, we thought the naïve method would work best. Our reasoning being the fact that companies only have so many words to present the job and responsibilities and they want to highlight the most important things. Likewise, in your resume you should take a similar approach, since you only have one page to emphasizing your skill set. However, when running different applications, it seems the tf-idf method works best no matter what.

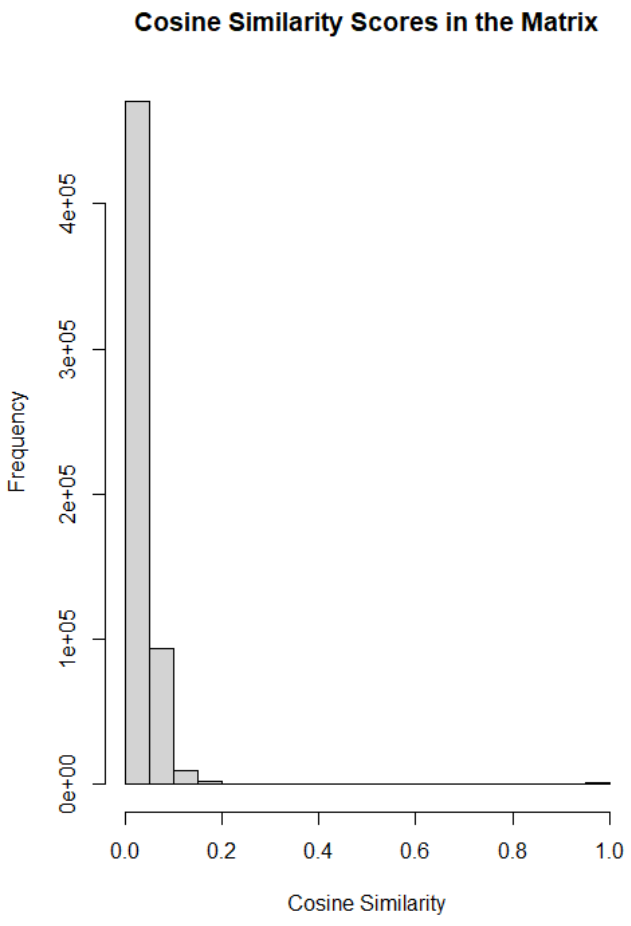
Job Rank	Normal Resume w/naïve method	Normal Resume w/tf-idf method	Spam Resume w/naïve method	Spam Resume w/tf-idf method	Changed Resume w/naïve method	Changed Resume w/tf-idf method
1	Data Architect	Data Scientist	Human Resources	Data Scientist	Data Architect	Ux designer
2	Data Scientist	Ux designer	Data Architect	Ux designer	Data Scientist	Data scientist
3	Machine learning engineer	Software developer	Data Scientist	Software Developer	Machine learning engineer	recruiter
4	statistician	Data architect	Machine learning engineer	Data Architect	statistician	Software developer
5	Deep learning	Deep learning	Statistician	Recruiter	Deep learning	marketing

In this chart we have the normal resume of Zac, who is aspiring to be a data scientist after graduation. In his normal resume both methods appear to work about the same, in terms of best matched jobs. Next, the spam resume. In this resume we added spam words that should match us to human resources. We did this by getting the most frequently used words in human resources descriptions, and added those to the bottom of the resume. For the naïve method human resources was the top ranked job, but in the tf-idf method it only helped include recruiter as the 5th ranked job. This happened because in the tf-idf method the word human and resource are downgraded because they appear in many documents. It shows that tf-idf is resistant to small outliers of words when compared to everything else. In other words, Zac's resume did not indicate he was interested in those positions other than just repeatedly adding "human" and "resources" multiple times. Finally, for changed resume we added other experiences Zac has had that deal with people and relationships. For example, we included mentoring and being an orientation leader positions and descriptions, but kept the bulk of the resume the same. In this situation, the tf-idf better reflected the changes than the naïve method. For all the reasons listed above the tf-idf method does better, as it accurately reflects the desire of the resume and person.

Next, gathering the top 5 job titles we output the top match in each position. This way the recommendation system most accurately fits your resume and gives out diverse recommendations. The way we fit descriptions to the resume was using the tf-idf for each description. The rationale behind this, is that descriptions for a data scientist at many companies should be very similar to one another. Thus, the differences are what will make the matches the best. It must be noted, that this is more a human-in-the model algorithm than a traditional algorithm, and as such you can change many of the features. For example, you could take the top 3 titles that match and get the top 3 best fitting descriptions.

Finally, the validation of the process. Our first validation comes from the process of matching. The process of matching involves using the tf-idf values and multiplies the word frequency of your resume, and thus returns a score. As seen above, these scores will generate good matches from your resume. The score is calculated by multiplying the values from the tf-idf to your frequency of words in the resume. The higher the score, the better the match. For example, Zac wants to be a data scientist and has the word data in his resume multiple times, this turns out not to be super important because data is used in many job titles. However, his mentioning of SQL and other programming languages create a big score because they are only important to certain jobs, such as data science, and this gives him that match. Next, diversity is attained because we take 5 Job titles that you best match with, and using the same process return the job description with the best score. This is truly a diverse method, as seen above in resumes where there is no certainty in what a person wants to do, the recommendation system returns varying jobs, such as marketing and software development. Additionally, in resumes more tailored to certain fields it still gives a diverse recommendation of 5 different job titles.

Method 2:



The second method builds upon tf-idf and seeks to provide more precise and accurate recommendations to the user. For this method, we built a cosine similarity matrix of all of the job descriptions in the database, and then returned to the user the top 3 jobs with the highest cosine similarity. Cosine similarity is calculated by using the tf-idf scores to turn each of the job descriptions into unique vectors. Tf-idf gives a unique numeric score for each word in the dataset, and who job descriptions can be turned into vectors by simply stringing together the tf-idf scores for each word in the job description. An example of a vector would be: [0, 0, 0, 0.019, 0, 0, 0.052, 0.052, 0.052, 0.052, 0.052]. Since we are using whole descriptions instead of simple sentences, the vectors we used would be larger than that one.

Vectors are useful because they allow us to numerically represent complicated data like sentences and paragraphs. Once the job descriptions were vectorized, cosine similarity scores were calculated by taking the dot product of the vectors and dividing this by their cross products. This method is useful because it

also allows us to compare vectors and thus job descriptions of different lengths. Cosine similarity is a measure of distance, and can be conceptualized as a representation of the distance between two vectors.

Eventually, we created a matrix of cosine similarities, giving the similarities between every job description to all of the other ones, and to the resume as it was later added to the dataset. To identify the best and worst jobs, we simply sorted the column of the matrix that corresponded to the resume, and identified the indexes of the data frame that corresponded to those best/worst cosine scores. We were then able to use those indexes to retrieve for the user the best and worst matches.

Above is a histogram showing the distribution of the cosine similarity scores. The scores corresponding to the jobs that we recommended were 0.05990314, 0.05196184, and 0.04931347. Close to 95% of the data falls between Scores of 0-0.2, meaning that our algorithm recommended jobs that were definitely much better than average matches to the user's inputted resume. The jobs included a Software Engineering Job, an Apple job that mentioned veteran status in the job description (Zac is a veteran) and finally a data science job, all of which are great matches for Zac. The "worst" job descriptions included sales and human resources jobs, which are intuitively not good matches.

To generate diverse recommendations, we added a function that identifies the job titles of the 3 best jobs as identified by the cosine similarity matrix. The function then subsets the data and removes all of those job titles from the database. The function then remakes a new cosine similarity matrix, identifies the three best matches using the same method, and then prints out those job descriptions for the user. This ensures that diverse recommendations are made by removing jobs with the same job title from consideration, thus giving the user different types of jobs to pick from. In a future iteration of this project, I would like to customize this function to allow the user to choose if they want to only consider different job titles, or if they would like to focus on other factors such as location or full vs part-time status.

Conclusion:

Our cosine similarity method is the method we felt performs the best, as it returned job matches that were much more aligned with Zac's computer science and Data Science backgrounds, while also highlighting his military experience. Intuitively, this makes sense as Cosine similarity attempts to find the actual "distance" between vectors. Whereas, the other methods matched tokenized words with words in resumes. With this in mind, the big take-away is that it is very important what you put in your resume. The age-old advice of tailoring your resume to each job sounds tedious, but is said for a reason. In the way that we matched resumes to jobs, jobs match candidates to descriptions. Most companies receive way too many applicants to go through, and must use some sort of filtering algorithm. In conclusion, when we matched jobs to descriptions, we felt the cosine matrix was the best matcher, and should make you think about what you want your resume to say about you!