January 31, 2022
Elliott Chen

Imputation of Job Hunters' Interest in Advertised Positions

A central problem that has arisen in the course of my professional work has been what content would be appropriate to display to our users. In the broadest terms, our aim is to help connect those looking for jobs to those looking to hire through a mobile application, but constant decisions need to be made about which profiles a given applicant or recruiter would like to see. Older iterations of solutions to this problem were based on inelegant brute force methods that that were inefficient and slow in delivering answers—creating unacceptably long wait times for our users—or did not have enough training data to produce a machine learning model that generated classifications of consistent quality.

There are two stages to this project; the first one being tackled now to attempt to impute classifications using the training data that we do have in order to set ourselves up with enough data to build a neural net and writing them to our database. The second will be actually training aforementioned neural net and deploying the resulting model to our API so that it can leverage machine learning when making decisions on what content to display to which users. The end result will hopefully be a reduction in load times as well as an increase in the "quality" of the content that we show our end users—or, in other words, delivering quality of life improvements in our product.

How to measure success in the long run is ultimately a murky question in that it will require a non-negligible amount of user feedback, but the success of short term midterm project can be considered in terms of how much data was successfully (or usefully) imputed. This is in of itself something of a troublesome metric as it is subjective and will require at least some manual inspection of the imputed results at the conclusion, but there is not another method to our knowledge of determining whether our model's classifications are at least in the ballpark range of sensible and reasonable to a human.

Separate from the subjective judgment of the imputations are the metrics of the model that created those imputations to begin with. We will focus our attention on precision and recall. When deciding on which metrics to emphasize, we need to recall our business context. The cost of a false positive—where we display too many profiles that our users are uninterested in—is high; although we assume that end users will have some tolerance for being shown content that is inappropriate for them, too much such content may result in user frustration and a reduced user base. Therefore, we need to make sure that as many of our positive predictions are correct as possible, and monitor precision. The cost of an algorithm incorrectly deciding that a job that a user would not be interested in a given profile is relatively low in comparison—so long as we have other content to show them that they would appreciate, the impact of these false negatives can be neglected, as the end user will never know that they "missed" a good fitting profile. However, we do not have a sufficiently large enough body of data that it would be prudent to disregard too many true positives; we might run out of profiles to show our users. If it happened too soon or too often, this scenario would be just as bad, if not worse! We therefore also keep an eye on recall. We can also include overall accuracy as a metric for a bird's eye understanding of how well our model is predicting classifications.

One final interesting approach to the problem is to cluster our data and to see if any given clusters are correlated with positive expressions of interest. If there are enough such clusters with strong enough correlations, and if our resume/scan pairings end up consistent with the clusters generated by our swipe data, we might be able to use that to classify the pairings.

The data available to us can be divided into two sets: a body of expressed interest (positive or negative) generated by user activity in our mobile application, and a collection of pairings of resumes and job postings accumulated by a web service that we also provide. There are at the time of this writing roughly 1,300 such expressions of interest (henceforth referred to as "swipes," analogous to the finger gestures made in popular mobile dating apps that our UI/UX takes inspiration from) and almost 14,000 such pairings (henceforth referred to as "scans"). Of the swipes, about 300 are "right swipes",

or positive expressions of interest, and the remainder are left swipes, setting us up with a binary variable. Both are stored in a PostgreSQL database. While the scans have dedicated columns for the text body of job hunters' resumes and job postings, our swipes link to other tables that represent applicants and jobs. To reiterate, our goal here is to use the swipes' recorded expressions of interest to build a model that can classify whether someone who posted their resume might display a similar interest in the the job that they posted.

The first step in this project will be to marry the two sets of data together as closely as possible, so that a model trained on one can work well with the other. Our scans are more or less just strings from which it is difficult to extract more information than the applicants' skill set and qualifications on one side, and the jobs' technical requirements and general responsibilities on the other. Our swipes' applicants and jobs, meanwhile, have that data stored in a string field and can be extracted without too much trouble. The main work in this first step is therefore reducing our data to available and required skills. From there we can use a count vectorizer followed by a term frequency times inverse document frequency transformer (combined into sklearn's useful TfidfVectorizer class) to create our dataframe.

Once here, we need to inspect the strings produced from our swipes and scans to make sure that they are comparable with each other. Given the amount of text in a typical resume or job posting compared with what we ask of our end users, there is no guarantee that their vectorized features will be similar. We separate our dataframe's rows into those which were generated from swipes versus scans and compare each of their columns' means, medians and maxes to quantify any difference. The means are a little different; to try and rectify this, we prepare another dataset formed from the concatenation of the swipe and scan rows vectorized independently of each other. When comparing this set of data's statistics, however, their maximums are severely affected. Since it is unclear which dataset is more appropriate, we will use both of them for model training.

One final question during our data wrangling process is whether our data is structured in the cleanest, most succinct way. Most skills are represented by two features: one tracking how many times

an applicant mentions a given skill, and how many times a job mentions it. As currently set up, our models might not be able to make the necessary connection between these columns. It is possible that by combining these pairs of columns into single columns by only measuring the percent difference between the two; the difference between desired skills and possessed skills are what we are really interested in, after all. In those cases where a skill is mentioned by one side but not by the other, we simply provide a value slightly higher than the highest proportion we otherwise achieved. At the end of our data wrangling process, we thus have four dataframes, divided into those whose scan and swipe rows were vectorized independently or simultaneously, and those whose features have been merged or left independent. In the rest of this paper, the dataframes will be referred to as A, B, C, and D: A is the one with independently vectorized un-merged features, B is the one with simultaneously vectorized un-merged features, C is the one with independently vectorized merged features, and D is the one with simultaneously vectorized merged features.

The model training that follows is relatively straightforward. We use the elbow method to determine the optimal number of clusters for each dataframe, and arrive at 14, 10, 8, and 7 clusters for A, B, C, and D respectively. We also conduct principal component analysis to reduce the number of features. We attempt K-nearest neighbors and Gaussian distributive clustering, but neither produce substantial correlation in between their clusters and expressions of interest; our Gaussian distributive algorithm in particular had trouble generating meaningful clusters at all.

Logistic regression performed somewhat better, returning an acceptable recall and a promising accuracy score for dataframe B. However, all four dataframes suffered from abysmal precisions; dataframe B scored the best with that metric, but only achieved 24%. Some of our logistic regression models failed to converge altogether when trying to predict our resume/job data after training, and so we move on to our final proposed type of model, ensemble decision tree learning.

These models' results were much better across the board. None of our random forest models had a testing accuracy beneath 70%, and their precisions were likewise all above 71%. These models

struggled with recall, averaging only 43%, but the best one achieved 65% with that metric. Happily, that model was also the most accurate overall and had a perfectly acceptable precision; both of these metrics were 83%. When predicting on the resume/job pairings, it classified roughly one in ten of the pairs as an observation where the job hunter would express interest in the job. This statistic passes first glance based on general and the author’s anectodal personal experiences in the job market, and should be a sufficiently high proportion to allow further training with the generated classifications to be performed. We thus save our model and feel confident in using it to update the resume/job pairings in our database.