Roshi: Machine Learning Powered Job Recommendations

Dhrupad Bhardwaj  
Candidate for Bachelor of Science in Engineering  
Department of Computer and Information Science  
School of Engineering and Applied Science  
University of Pennsylvania  
Email: dhrupadb@seas.upenn.edu

Shreshth Khilani  
Candidate for Bachelor of Science in Engineering  
Department of Computer and Information Science  
School of Engineering and Applied Science  
University of Pennsylvania  
Email: shreshth@seas.upenn.edu

Adviser: Andreas Haeberlen (ahae@cis.upenn.edu)

Abstract—Recruiting portals today are exclusively focused on job search and filtering. Roshi is a smart web application that uses Machine Learning to learn a student’s preferences and accordingly recommend employment opportunities. Roshi uses a combination of revealed preferences and implicit choices to train an ML engine and iteratively update recommendations.

I. INTRODUCTION

From the very beginning we wanted to create a Senior Design project which used Machine Learning to build a recommendation platform of some sort. One major source of inefficiency that we’ve witnessed first hand is the job search and recruiting process. Students spend hours pouring over thousands of positions looking for an ideal match. Similarly, recruiters across a host of industries complain that finding the right match in terms of profiles requires going through hundreds of profiles. In each case the industry norm relies on basic search and filtering, looking for the presence or absence of certain skills or filtering jobs based on high level features such as location, experience required etc. Roshi was aimed to be a proof of concept to try and evaluate the possibility of using ML to simplify this matching process. We hoped to find features which were relatively easy to extract and see if our system produced meaningful recommendations that would narrow down the space of interested jobs faster than a conventional search and filter platform.

We began by attempting to build a system which recommended students to recruiters for a certain position. In the following section we talk about in greater detail the need to abandon this idea and instead reversing the direction of the recommendation. Roshi’s primary objective was to take a job seeker’s profile and recommend positions that they would be interested in. It used not only profile information furnished by the user, but also feedback from the user and other similar candidates to recommend a list of 30 or fewer jobs. The user would iteratively pick jobs they were interested in and the algorithm would use this feedback to narrow down the scope of the recommendation.

II. INITIAL PLAN AND PIVOT

The initial objective for Roshi was to be able to match recruiters with suitable candidates for a given position, and support a clean and simple application platform for candidates. From a commercial viability standpoint, it made more sense to suggest students to recruiters. Companies spend millions of dollars on their respective Human Resource departments and tools to streamline the process are in high demand.

However, we faced two main challenges. Firstly, we soon discovered HR departments were quite unimaginative and the inertia of change was considerable. Most of them seemed reluctant to even consider a new system and were even more reluctant to reveal the specific attributes they valued when evaluating candidates. Secondly, to be able to match students with jobs, we would need a lot student profiles to accurately be able to predict matches. Because of the timeline of our project and the lack of reputation, students were reluctant to give us their personal information such as GPA and work experience etc. Scraping LinkedIn was an option we considered but that opened us up to a number of possible privacy concerns.

In conclusion, we decided to pivot our project and reversed the direction of the prediction. Instead of matching students to a job we decided to match jobs to students. While this may be less commercially viable, the theoretical considerations were virtually identical. Secondly, job data was far easier to come by and even with a few student profiles we could make meaningful predictions.

III. GOALS

The larger goal of this project is to build a recruiting application to help students find a job that they’ll actually love. Job search decisions for Penn students are seldom dictated by love of the profession. Consulting, banking, and software development are professions people blindly choose because it seems like the obvious next step in a successful person’s life. This project tries to take a step in the direction of job search sites which want to help connect people with jobs that they will actually enjoy.

Roshi was made with the intention of being a more usable job search website. One that is easier to navigate, without dozens of filters and search bars, and quick to interact with. To do this, the project had to recommend jobs well enough that search and filters weren’t required. It also had to be low-latency and scalable in order to calculate and refresh recommendations.
IV. WORK FLOW

A. Creating an account

The process to create an account is simple. We ask the user for an email ID and password. To restrict the scope of our recommendations we limited the project to Penn students only. We used their email ID (@seas.upenn.edu, @wharton.upenn.edu etc.) to come up with naive set of initial recommendations.

![Main Signup page](image1)

**Fig. 1. Main Signup page**

B. Populating your profile

Once the new user has confirmed the creation of their profile, they will be taken to a basic homepage with their profile which looks like this. It would have their school and email ID filled out with the rest of the fields left for the user to populate.

![User Profile page](image2)

**Fig. 2. User Profile page**

The user would ideally fill in their profile information at this point. This information would give the algorithm a more detailed insight into the specific type of experience this user has had in the past and the sort of job opportunities they are looking for.

C. Selecting interested jobs

With this information the app calculates an initial set of recommendations which can be found on the Jobs tab.

![Recommended jobs page](image3)

**Fig. 3. Recommended jobs page**

A user then reviews each of these jobs that are suggested by the platform. The platform suggest at most 30 jobs at any given time and these are shuffled whenever the recommendations are recalculated. When the user finds a job they would like to apply to, they hit the "Interested" tab, which moves this particular posting to their "Interested" tab. This provides secondary feedback where the app matches the user’s profile to this particular position. It uses this information to suggest similar postings based on the profile of others who also marked this position as "Interested". Users may at any time remove jobs from their "Interested" tab.

![Interested jobs page](image4)

**Fig. 4. Interested jobs page**

V. APPROACH

A. Data Collection and Cleaning

Populating structured job posting data with relevant features was the primary objective of this step. We used PennLink as our primary source of job data. We were able to retrieve approximately 3,300 job postings from PennLink. A large subset of these postings had the relevant metadata for us to be able to extract the features we needed for our prediction. The final corpus of postings we used for the platform included 2,672 jobs.

Basic job metadata such as Company Name, Location, position type etc. were easily available and algorithmically
extracted. In some cases we needed to use specialized packages to retrieve this information. The Python Geodict package was especially useful in extracting location information. However, given the unstructured nature of job postings and the complexities arising from algorithmically extracting features, we decided to use crowd workers to extract large amounts of relevant information for us. We used the Amazon Mechanical Turk to post Human Interaction Tasks (HITs) where they were provided with a job posting and they extracted the relevant features such as company name, job location, company headquarters, industry, type of position, experience required etc. Some of these were intentionally redundant so that we could use them as a gold standard to test worker results. In most cases, workers who passed the test questions with regards to company location, company name and position type also managed to effectively extract the other features mentioned above.

One feature we that was particularly important for our analysis was the speciality or technicality rating of different positions. This feature, called "technicality" was a number on a scale 1 - 10 where 1 was a highly artistic or creative position such as an actor, designer or writer and 10 was a highly technical position such as a software developer, financial quantitative analyst or researcher. We asked crowd workers to rate each position on this scale giving them the rough scale mentioned in the previous sentence. Given the lack of accurate gold standard data and relative subjectivity of this field, we used a more sophisticated method to aggregate results. Instead of simply taking the majority vote we used the EM Algorithm to aggregate worker results by weighting worker responses by worker quality and iteratively updating quality till a stable prediction was achieved (See Appendix).

B. The Algorithm

Once we had structured job data we had to create a prediction platform to recommend jobs to individual students. This was done using two ML algorithms and then aggregated to get a final list of recommendations. Each of the algorithms used the implementation found in the Python Scikit-Learn package.

1) k-Means Clustering: Once we populated our job corpus, we ran the k-Means clustering algorithm to find groups of positions which were similar. The algorithm used features extracted from each job posting which could easily be made ordinal and clustered them into groups based on features such as location, technicality rating, industry and required experience. To compensate for particular features having a wider spread, we weighted each feature by a weighting score to ensure the clusters were meaningful. For example, we used the Google Maps Geocoding API to get a coordinate for each location and then compared Euclidean distance to cluster jobs which were in the same region. In general the spread for this feature was much higher than differences in technicality, and thus was weighted far lower than features such as experience required, technicality etc. We manually tuned the weights assigned to get meaningful clusters. We also roughly found five clusters in our corpus that were relatively closely related.

Once a user populated a profile, we matched major, school and keywords in previous positions held to determine similarities with other candidates in our system. We then made an initial recommendation of jobs consisting of the two most frequently occurring clusters from which the similar candidates had marked interested jobs.

2) Decision Tree: The second method we used to recommend jobs used a Decision Tree. To train the tree, for every job that the user marked as interested, we found a set of users who had marked the same posting as "Interested". This gave us a set of users with similar job interests. Once that was collected, we aggregated features from each of the jobs those profiles and our user had marked as "Interested" and for each job in our corpus assigned a 1/0 label based on whether or not our user or any of the users from the set of similar users had marked that job as interesting. The trained Decision Tree was then run on the corpus of jobs and asked to predict whether or not a particular job would be interesting to our user. This method allowed us to cast a wide net with sufficiently small input feedback from the primary user. We used Scikit-Learn’s native Decision Tree algorithm which uses an optimized version of the CART (Classification and Regression Trees) algorithm.

3) Aggregating Results: We used a simple weighted average method populate the final list of recommended jobs. Initially we gave the entire priority to the clustering method as the user didn’t have any jobs marked “Interested” for us to be able to populate a decision tree. Once the user had picked three or more jobs as interested, we updated the weight to be distributed 65:35 between the clustering and Decision Tree algorithm. For six or more interested jobs, the weightage was changed to 35:65. This weightage was used to score each job. For example, if the user marked only four jobs as interested, then if a job posting was recommended by the clustering pipeline, it would receive a score of 0.65, a score of 0.35 for only being recommended by the Decision Tree and 1 if it was recommended by both. Each case we would take the top 30 jobs for the final recommendation.

VI. RESULTS / MEASUREMENT

The first way in which the algorithm was evaluated was by asking a few users to use the website in a specific way and provide feedback. Selected users were asked to rate how interesting the job recommendations were to them on a scale of 1 to 5 (1 being not at all interesting and 5 being extremely interesting). They were asked to provide a rating four times: (1) right after creating their profile, (2) after marking one interest in one job, (3) after marking interest in two jobs in total, and (4) after marking interest in two jobs in total. This was done for a group of 10 users. Users were divided into categories by their primary school and these numbers were aggregated. Interest It was found that there was a positive change in interest for the first two iterations for Engineering and Wharton students. College students remained as interested as they initially were through iterations.

Next method of evaluation was using crowd workers. Since jobs with higher technicality were more common in
the dataset, those were used in this part. Tasks were designed to display a job that a user is interested in and also display 5 recommendations the algorithm generated based on that interest. Questions asked were: (1) How many jobs are similar to the one selected job? (2) How many jobs have similar titles (share keywords or sound alike)? The one interested job was randomly assigned to workers. 100 hits showed us that on average 3.4 out of 5 jobs were similar and 3.2 had similar titles.

Lastly, the latency of the recommendation system was measured. We measured on average for 5 different profiles the time it took to query interested jobs from the DynamoDB table when running the recommendation script. We also timed how long the script itself took. The idea was to get an understanding which part of the application was a bottleneck and where future optimizations would have to be concentrated. Unsurprisingly, the recommendation script took far longer than the database query. That said both latencies increased relatively linearly. The graph below shows in detail the relative latencies of both parts of the application.

VII. ETHICAL / PRIVACY CONCERNS

The job data collected was scraped from PennLink. While scraping data from websites is not unethical, exposing data available only to someone with a Pennkey could be so and care was taken to only ever allow a user with a Penn email access to this data. We made sure to include standard email verification of users on account creation so that this data is kept restricted to Penn students only. Standard password hashing and other security measures were taken so user accounts cannot be accessed by anyone apart from the user.

A privacy concern could be that the jobs a user marked as interested to train the decision tree part of our algorithm. This utilization of user data is pretty standard in the industry and an agreement could easily be added to ensure that the user is aware and fine with such data usage.

The final ethical concern with this project was using the help of crowd workers to clean data and to help evaluate the recommendation system. Care was taken to ensure that fair wages were paid by estimating duration of the task and difficulty. This project would have been extremely challenging without the help of these workers and the hits were priced to ensure that they get an above average hourly wage.

VIII. DISCUSSION

There were cases where the job recommendations seemed to work better. For technical jobs like developer, analyst, and consultant jobs, there seemed to be better (and more similar) suggestions than for creative jobs like writing. This is probably because the job data was skewed to the technical jobs (80% of the jobs are such). Construction and other blue collar jobs do not even feature in the data set and were beyond the purview recruiting system as it was tailored to find jobs for Penn undergraduate students.

The system would also yield better results with more and better data. More user data could enable adding more components to the machine learning algorithm like matching users using a Bayesian Classifier and recommending jobs one user selects to other similar users. Better job data with keywords and tags could help cluster jobs together and find similar jobs that could be in a different cluster.

Usability of the system was a big factor under consideration. A/B Testing and more users could help improve usability. This would be especially helpful to contrast and compare this application to a traditional job site with search bars and filters. Finding whether recommendations only is more usable than
search/filter sites would be possible with such testing and so would finding a good balance between the two models.

IX. Future Possibilities

The very first next-step to this project would involve using real user data to improve the current algorithm. This could happen in two parts. First, user feedback of the recommendations could be surveyed by the application. This would enable us to know the efficacy of the recommendation at different iterations of the process and would help tune how to better aggregate the machine learning pipelines. Second, a third pipeline could be added with matches users using a Bayesian Classifier. This could help by allowing recommendation of jobs a user is interested in to other similar users.

The logical build up for this project would be to actually get companies to post jobs on this application and create a system to allow users to apply to these jobs. Job postings inputted could take in keywords, tags, and other objects that would help the recommendation system.

Finally, a similar recommendation engine could be built to recommend candidates to recruiters, as was the initial goal of the project. More user data to make this possible could be collected by implementing the other possibilities and opening up Roshi for students at Penn to use.

APPENDIX

EM ALGORITHM

Algorithm 1 Expectation Maximization (EM) algorithm

1: procedure EM–ALGORITHM
2: Input: Labels $l[[k][n]]$ from worker $(k)$ to object $o_n$.
3: Output: Confusion matrix $\pi_{ij}^{(k)}$ for each worker $(k)$, Correct labels $T(o_n)$ for each object $o_n$, Class priors $Pr\{C\}$ for each class $C$.
4: Initialize error rates $\pi_{ij}^{(k)}$ for each worker $(k)$ (e.g., assume each worker is perfect);
5: Initialize correct label for each object $T(o_n)$ (e.g., using majority vote);
6: while not converged do
7:   while not converged do
8:     Estimate the correct label $T(o_n)$ for each object, using the labels $l[[k][n]]$ assigned to $o_n$ by workers, weighting the votes using the error rates $\pi_{ij}^{(k)}$;
9:     Estimate the error rates $\pi_{ij}^{(k)}$, for each worker $(k)$, using the correct labels $T(o_n)$ and the assigned labels $l[[k][n]]$;
10:    Estimate the class priors $Pr\{C\}$, for each class $C$;
11: end while
12: end procedure

REFERENCES