Networked

Team Members: Soham Dharmadhikary, Neel Shroff, Nikhil Kokra, Siddarth Sankhe

Faculty Advisor: Swapneel Sheth

Background and Market Opportunity

As automation increases globally, jobs are becoming more and more specialized / creative. Factory workers, phone operators, cashiers, etc. have been replaced by machines, leaving the more complex jobs for humans; increasing demand for highly skilled labor goes hand in hand with intelligent recruiting. The global market for recruitment as it stands today is valued at $5.48 billion, expected to grow at a whopping 15.8% compound annual growth rate[1]. Now, recruitment services are primarily conducted qualitatively, via headhunters and recruitment agencies run by recruiters who specialize in certain industries. As the world digitalizes, it's important to take advantage of big data and find opportunities in employing machines / data analysis to complement recruiters. With this view, we propose a mix of the incumbent human-centered approach, and a novel data-centered approach as an improvement upon current recruitment practices.

Our product

Networked is a platform designed to connect employed alumni / recruiters with job-seeking students. It provides alumni with a clear view of what students from their school are applying to their company, and allows them to communicate (and refer) those students. It provides students with a dashboard of alumni that work at their desired company, and allows them to reach out without ever having to leave the platform. Recruiters on Networked can view all of this activity, along with a data-driven recommendation system, which classifies students based on attributes such as GPA, experience (quantifying using number of previous internships), major, interests, skills, etc.. The goal of the platform is to make the referral process easier / more transparent, and to use accumulated data to assist recruiters.

Value proposition

Our project seeks to solve a problem many students face every year. As internships become more and more competitive, it is increasingly difficult to get in touch with alumni and build meaningful relationships. To help this, we hope to create NetWorked, which will be able to more intelligently match alumni, many of whom are already looking for people to connect with, with students who are hoping to get hired at their companies. We will create a system that allows alumni to refer students, after entering their preferences for potential interns/new-hires. Students can enter their resumes, GPAs, classes taken, and more, and we will allow alumni to match with students most suited to their needs. Overall, we hope to make the recruiting process more transparent, simple, and meritocratic.

So, our central value proposition is twofold. First, we create an ecosystem in which both students and alumni have reason to create meaningful connections. This is because each is actively looking to either find a job or fill a job. This means that there are no extraneous details or processes they have

[1]https://www.grandviewresearch.com/industry-analysis/recruitment-process-outsourcing-rpo-market#:~:text=The%20global%20recruitment%20process%20outsourcing,factors%20driving%20the%20market%20growth.

to go through in order to get what they are looking for. Second, our platform matches students to jobs they are most qualified for and likely to be successful at. This saves alumni time, so their referrals are based on merit and fit instead of arbitrary measures as is true of the status quo.

Stakeholders

Our primary stakeholders are the users of our platform: students and alumni. Our students are those college students, initially in STEM majors, who are looking for jobs and internships. Many of these jobs require an explicit referral before a student's profile is even considered. This disadvantages many capable candidates that may not be as well connected. To this end, the student is getting the added value of being able to network directly with alumni that can present opportunities that they may not otherwise get.

For alumni, the value is twofold. First, many alumni are incentivized to refer candidates to their firms. If the candidate is offered a position, the alumni may be offered some sort of financial reward. Our platform allows alumni to find candidates that are most aligned with their companies, and our ML model helps predict which candidates are best fit to certain position openings. Second, our platform is an easy way for alumni to help out candidates from their own schools, instead of having to scour through email listservs, on campus recruiting sessions, and LinkedIn. This makes the alumni more in control of who gets to join their firms.

Finally, additional stakeholders include school administrations, many of whom actively look to find their students jobs, and company HR teams. Both of these types of stakeholders are incentivized to encourage (and even pay for) the services that our platform can offer.

Competition

Our key competitors fall into two categories: direct and indirect. We summarize both below, though we note that there are no other tools that are geared directly towards enabling referrals for technology positions.

Our direct competitors are platforms that allow connections between alumni, companies, and students. Though the market is relatively fragmented, LinkedIn and TalentNest hold the highest shares, with 18.5% and 22.5% respectively[2]. At the University of Pennsylvania, we primarily use Handshake, which is a smaller player that is commonly used among Ivy league institutions. These platforms largely function in similar ways. Companies post jobs, and students are able to apply through the platform or they are redirected to a portal. However, these sites focus on the application process rather than the connections aspect. While LinkedIn allows students to connect with other alumni, this is not a standardized process and job postings often hold limited information for which alumni they should get in touch with. For this reason, our platform serves as a good precursor in the recruiting pipeline. We allow students to get in touch with alumni from their universities before

[2] https://www.datanyze.com/market-share/ats--28/handshake-market-share

they go through the actual application process on one of these sites, improving their odds for success.

Our indirect competitors largely come from the status quo. While the firms we mentioned above are primarily geared towards applying to jobs, the firms we consider indirect competitors are other ways students can connect with alumni and companies. These include informational sessions and email. These are not truly competitors in the conventional sense, but they can steal away students and alumni that would otherwise have used our services. We are confident, however, that the value we add both through our matching process and simple UI makes NetWorked a far more seamless user experience than attempting to find arbitrary employees of a company. We also believe that our platform gives a much higher success rate at facilitating meaningful connections, since all our users are on the platform for the same, specific purpose.

Revenue Model

Our revenue model would be subscription-based. Once at scale, we would introduce multiple subscription lengths for students and recruiters, e.g., monthly / weekly / annual subscription rates. The value added for students is potential referrals to companies of their choice, and more attention from recruiters on the platform. The value added for recruiters is being able to use our data analytics engine, and interact with applicants on-platform, knowing which alumni referred them.

We could also sell ad space to companies looking for applicants. A lot of the traffic on our platform will be students looking for jobs, so our web pages are perfect marketing space for companies with spots to fill.

Our Technology and Further Research

Creating a more scalable database

Description: Our current database makes use of a single Mongo Cluster and a single S3 bucket. However, to scale our application, we would likely need to horizontally scale our database implementation to store a larger amount of data and allow for a higher number of concurrent requests

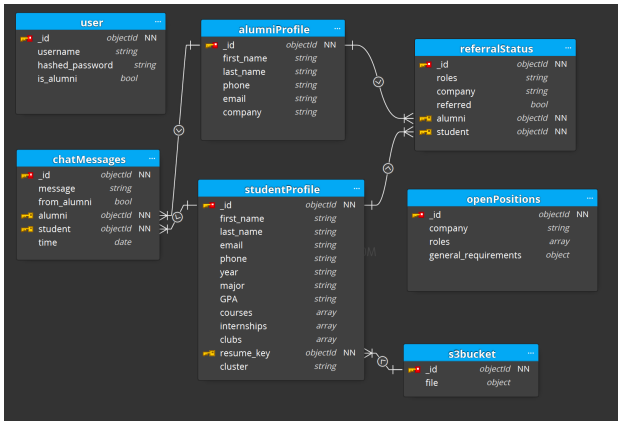
Analysis:
Currently, we are using a single Mongo Atlas Cluster to store all of our application data. Our database consists of six collections (user, studentProfile, alumniProfile, chatMessages, referralStatus, and openPositions). The alumniProfile and studentProfile consists of the data for each individual alum and student respectively, the referralStatus collection contains the referral status between every student and the companies they are interested in, and the chatMessages collection contains every chat message between any pair of students and alumni. Below, we have included a diagram of the current schema.

We believe that the current implementations of the chatMessages and referralStatus collections (primarily the former) will run into a number of performance issues as the number of users for our platform increases. Although we could vertically scale our database to deal with some of these issues, by using a cluster with higher size, capacity and CPU, the benefits of scaling on a single machine are bounded.
Since we are storing all of the chat messages in one collection and chat is a feature that multiple users may use concurrently, the volume of concurrent requests may overburden a single server. Thus, we think horizontally scaling might be more effective for our purposes. Splitting up our dataset over multiple different servers will allow us to split up the workload and more effectively handle concurrent requests. Since MongoDB is relatively easy to scale compared to many relational database systems, we think using this database system still makes sense if we were to scale our application. MongoDB provides functionality to easily replicate and shard, which could help with the horizontal scaling of our database.

One way to shard our collections would be by using the company the alumni works for. Assuming we have about an equal number of alumni across many different companies, sharding on this key would help distribute the data pretty evenly. Currently, our chatMessages collection does not have this field, so sharding by this key would require us to add this field to the collection. For this case, hashed sharding would probably be a better choice than ranged sharding, especially since our sharding key does not have any meaningful ordering.
In addition, since the alumniProfile and studentProfile collections are likely to be much smaller than the other chatMessages and the referralStatus collections, it may make sense to not shard those two collections to avoid adding too much unnecessary complexity.

MongoDB's "sharded cluster"[3] also makes it easy to combine sharding and replication. In fact, each shard is deployed as a replica set. A replica set in mongodb is basically a group of processes that maintain the same dataset. Thus, this sharded cluster makes it possible to handle "Automatic Failover", which elects one of the secondary nodes (in a replica set) to become a primary node if the original primary node disconnects.

Current Schema



[3] https://docs.mongodb.com/manual/sharding/, https://docs.mongodb.com/manual/replication/

Improving performance of our data analytics

Description: Our current database uses a simple K-means clustering algorithm to classify students into a certain number of buckets. These include buckets for high academic achievers, high experienced candidates, and more. Then, when an alumni selects students, we can learn which clusters they favor the most and recommend students that fit their requirements. As more students join the platform, the clustering becomes more accurate and buckets can be added.

Analysis:
To improve performance of our platform, we'd primarily target the clustering algorithm. Firstly, we could run the clustering algorithm not on every new addition of a student, but rather every-time the database changes in a significant enough way for the clusters to change. We could define this significance via a certain number of students added (e.g. only rerun the clustering everytime the database size increases by 20%), or via a certain number of changes to student profiles (e.g. only rerun the clustering everytime 200 student attributes are created / updated). Since we're not rerunning clustering everytime a new student is added, new students would be assigned a cluster based on what cluster centroid they are closest to. Note that this improvement in speed comes at the expense of some accuracy, because the optimal clusters may change when a new student is added. In this setting however, students generally fall into 1 cluster quite neatly so we don't envision this being a problem.

We are using K-Means as our clustering algorithm, which means that initialization will have drastic impacts on the end resulting clusters. Thus, our initialization techniques were very important to us. We wanted the resulting clusters to not only be optimal, but also intuitive. A good non-intuitive solution wouldn't be helpful to us, nor would it be helpful to recruiters looking for particular types of students. To keep the resulting clusters intuitive, we initialized the K-means with what we thought was a student representative of the groups we thought we should end up with. For example, an academic but not experienced student would have a high GPA but low number of internships. A student great across the board would have a high GPA and a lot of internships as well.

We could improve accuracy of the algorithm by manually weighting features according to their importance. For example, we could have alumni rank the characteristics they find most valuable in candidates. This could initially translate into a scoring system that serves as a weighting for each characteristic. This could then help us determine not only which candidates are most similar to their preferences, but also which clusters they would find most qualified.

In general, there are many other recommender systems that we looked into. For example, Netflix's[4] famous matrix based algorithm that relies on collaborative filtering. Our current system relies on content filtering, as we store user (in this case alumni) attributes. However, we could instead rely solely on alumni ratings. This would require each alumni to rate candidates based on how qualified they believe they are. Using these ratings, much like the Netflix system, we could create a matrix of

[4] https://towardsdatascience.com/tensorflow-for-recommendation-model-part-1-19f6b6dc207d

alumni and students, with cells corresponding to how each alumni rated a student. Through a combination of matrix factorization and some form of gradient descent, then, we could calculate the expected rating for an arbitrary alumni on an arbitrary student. This circumvents any issues related to manual labelling, but requires both more computation and more initial data for alumni. We see this as a worthwhile future addition, but ultimately decided against this as our initial model because it would require too much data at the onset to be useful in any way. That is, if the matrix mentioned above is too sparse, then the model provides very poor results. As data is built up, however, we may be able to see increasing returns to this type of model and it could complement our current K-means approach.