Avinash Tamby

Indeed Job Scraping Project

Executive Summary

The goal of this project was to scrape Indeed.com for Data Scientist job listings (and related job titles e.g. Data Analyst, Statistician, Machine Learning Engineer, etc.), save the job title, company name, location, summary and salary (if applicable) and use this information to try to predict whether a certain job will pay above or below the median salary for Data Scientists across the country. The vast majority of job listings on Indeed do not contain salary information (Less than 10% of job listings have an associated salary o salary range). Thus, the goal was to try to scrape as many relevant job listings as possible and build a strong model.

Introduction

Data science is a very popular field in computer science these days. It's been called the "sexiest job of the 21st century" and big tech companies including Google, Facebook, and Microsoft rely heavily on data to improve their products and stay relevant in a fast-changing industry. Many tech startups including Uber and other well-known companies also rely heavily on data science for product development and innovation. Thus, it would be interesting to see what the compensation looks like for this trending job. Of course, as with any job, more senior-level management positions undoubtedly pay higher than entry-level ones, but are there any other factors or skills that may be correlated with higher pay?

Methods

I used Python to do this project. First, I decided which cities I wanted to look for jobs in. I decided to use the 30 or so most populous cities in America (taken from Wikipedia) and a few other cities that I think might have a big tech scene (Palo Alto, CA, Boston, MA, Pittsburgh, PA among others). Once I had my list of cities, I build a scraper to scrape the Data Scientist job listings in each city and only changed the portion of the URL which corresponded to city name. I ended up with over 2500 different jobs; however, after discarding all of the jobs that had no salary listed and had hourly or monthly wages (which indicated that they might not be full-time jobs), I was left with about 150 job listings. The goal of this project was to determine whether a certain job listing would pay above or below the median salary for data scientists across the country, so of those 150 job listings, I first found the median, which was about $105,000. Then I ran several different models with different features to try and find the model that performed best. I used random forests and gradient boosting trees as classifiers. I used different combinations of job location, job title and a tf-idf vectorizer on job summary as feature subsets to put into my classifiers. Note: The job summaries were only snippets of the full job summaries that were listed in the job listing description. On Indeed, each job description is a link to the full job post which describes in more detail the qualifications needed for the job and a more complete job description. I only pulled a 2-line snippet of this full description, so each observation may not (and probably does not) have a very representative job description. I also built a function which takes in features, labels and a classifier and outputs an accuracy measure rather than running the same code with different features and classifiers. I also ran each model 50 times and averaged the accuracies to get a more stable accuracy measure for each model.

In my first model (1), I try a Random Forest with only location as a feature. In my second model (2), I try a Random Forest with location and seniority as my only 2 features. For seniority, I looked through each job title and checked if the words ‘Chief’, ‘Senior’, ‘Sr’, ‘Lead’, ‘Principal’, ‘Manager’, or ‘Director’ were in the job title and created a binary seniority feature with 1 if the job title included one of those words and 0 otherwise. In my third model (3), I fit a tf-idf vectorizer and use that as my feature set on a Random Forest Classifier. In my fourth model (4), I use the tf-idf feature set along with location and seniority with a Random Forest and in my fifth model (5), I use tf-idf, location and seniority on a Gradient Boosting Tree.

Results

Model (1) was one of my worst-performing models with an accuracy of about 57%. Model (2) performs slightly, although still barely better with an accuracy of .58. It is only when I add the tf-idf feature set that I see a significant boost in performance with Model (3) achieving a 76% accuracy. Model (4) achieves a 77% accuracy. Here, I decide to take a closer look at my results. The most important predictive features appear to be the words ‘looking’, ‘scientist’, ‘team’, and ‘experience’ along with the location of the job. While some of those words seem odd, I think it does make sense that ‘experience’ has some predictive value. The precision, recall and f1-scores for this model also perform pretty well achieving scores of .77, .77 and .74, respectively. The last model I ran, Model (5) also achieved relatively promising results with an accuracy of 0.77. This time, the most important features were the words: ‘looking’, ‘role’, ‘science’, ‘knowledgeable’ and the seniority feature: ‘senior’.

Discussion

It appears after talking to data scientists and browsing data scientist blogs and forums online that data scientists seem to swear by random forests and gradient boosting trees (although many prefer XGBoost over sci-kit learn’s GBT; I used the latter). And doing to natural language processing using a tf-idf vectorizer significantly improved my results over using my default feature set. Although in both models (4) and (5), at least one of location or seniority (from my default feature set) were designated in the top 5 most important features. I mentioned before that the job summaries were not complete and that this an important note to make. Many job listings have full job qualifications and descriptions in the link, and most of the time, not a lot of relevant information is provided in the snippet. A more thorough (although significantly more resource-intensive) approach would be to follow each link to the full job description page and scrape that information; however, with my limited computational resources and time constraints, this task would have been too resource intensive. Another caveat is that I have very few job listings in my dataset. I only have about 150 job listings for undoubtedly thousands of data science jobs available across the country. This undoubtedly leads to unstable and perhaps unreliable models with high fluctuations in terms of accuracy.

Conclusion

When trying to predict salary information, it seems that what ended up giving me the biggest boost in accuracy was the tf-idf vectorizer, and this sort of makes sense because it looks specifically at what words were actually important in the job summaries. Again, a more thorough approach would have been to scrape the entire summaries, but with an average (and thus somewhat stabilized) accuracy of 0.77 on my best model, it seems that just looking at job summary snippets performed fairly well.