Negotiating Your Compensation as a Software Engineer: A Prediction Problem Using Regression Techniques

Author: Simon Ayzman (sayzman@uchicago.edu)

Abstract/Executive Summary—Software engineers command high compensation for their skills. However, existing compensation transparency sites often do not provide enough personalization to help job seekers determine their market rate. This work examines granular Software Engineer compensation data from Levels.fyi using regression analysis techniques, such as ridge, lasso, and partial least squares. I develop models that accurately predict salary, stock, and bonus information with root mean square deviations of $23k, $59k, and $21k, respectively. An exploration of the models' regression coefficients reveals expected tech industry phenomena, such as compensation spikes (1) within tech hubs like California and New York, and (2) for particular skills like ML/AI, Distributed Systems, and Mobile Apps. It also uncovers less expected insights, such as (1) Netflix far outranking other companies, even in the FAANG cohort, on salary, and (2) fintech companies like Two Sigma and Citadel far outranking other companies on bonus.

1 Introduction

SOFTWARE engineering is known to be a highly lucrative field. Engineers navigating the job search process must clear a variety of hurdles, including curating the list of companies and teams they are interested in, networking with recruiters and other engineers, and preparing for technical and behavioral interviews. Indeed, one of the more fortunate challenges these job seekers will face is negotiating their offers.

It is best to go into these negotiations prepared and armed with knowledge. Inexperienced negotiators are prone to the well-known psychological phenomenon of anchoring, whereby too much weight is given to the first offer. One of the best counters to anchoring is to focus on target compensation, and one of the best ways to determine this target is to accurately know the "market rate" of labor: How much have others in a similar position been able to get? Thus, software engineers use a variety of providers to determine their potential value, including Glassdoor and Paysa.

In this analysis, I use data from one of the more popular compensation transparency sites used by Software Engineers called Levels.fyi, which crowdsources granular datapoints about individuals' pay and experience. While Levels.fyi already provides helpful analysis and graphs about its data, there is room for improvement. Each job seeker has their own unique background, and it might be difficult to understand how one fits into the constellation of other Software Engineers who landed the same roles. This can be solved as a prediction problem using regression techniques. Equipped with helpful point predictions and confidence intervals around those predictions, engineers can have the knowledge they need to maximize their compensation in future negotiations.

2 Related Work

There exists a small body of research around predicting salaries and a related body of research around building compensation structures (from the perspective of employers). I will focus on the former, as it more directly relates to this paper's analysis.

One paper in this category is "Salary Prediction in the IT Job Market with Few High-Dimensional Samples: A Spanish Case Study" by Martin, Mariello, Battiti, and Hernandez (MMBH). The researchers scraped data from an e-Recruitment website for IT jobs in Spain, which contains a large collection of job offers that included the salary, time dedication (full/part-time), requested skills, and many more pieces of information. Over five months around early 2016, they collected almost 4000 such job offers, with around 2,000 features each. They cleaned the data for missing values, dropped unstructured textual data (e.g., job title and description), converted certain textual field to numeric equivalents (e.g., "2 years" to "2"), merged semantically equivalent terms in certain features (e.g., "admin" and "administrativo" both belong under "administration"), removed keywords that appeared under a certain threshold of times, and rescaled all features to the interval [0, 1] based on normalization on their individual minima and maxima. This whole process managed to reduce the number of features from ≈2000 down to ≈200.

MMBH first did some analysis using a regularized linear regression model (ridge regression), an interpretable multiple linear model, and K-means clustering. To improve prediction, they then reframed the problem as a classification problem: they introduced four salary bands representing low, medium-low, medium-high, and high. After applying this labeling, they explored all of the

following classifiers: generalized linear model (GLM) via logistic regression (LR), K-nearest neighbors (KNN), multi-layer perceptron (MLP), support vector machines (SVM), random forests (RF), adaptive boosting with decision trees (AB), "Vote" (an ensemble voting classifier on all the previous models), and "Vote3" (an ensemble voting classifier on the best three previous models). In each of these models, the researchers optionally used the filter method X-MIFS for automatic feature selection, but they ultimately determined that this technique did not help much.

The researchers tuned their model configurations by performing a grid search on 90% of the data using 3-fold cross validation and picking the configuration with the best classification accuracy. In terms of classification accuracy on the testing data, the LR (58.6%), MLP (59.1%), and SVM (66.3%) models performed significantly worse than KNN (79%), RF (84.0%), and AB (83.6%). Moreover, they compared all the classifiers on the testing data on a number of other scores based on the confusion matrix: the precision, the recall, the F1 score, AUCPR, AUC-ROC. The Vote (84.4%) and Vote3 (83.7%) models performed similarly well to KNN, RF, and AB in terms of accuracy, and even better on the other relevant metrics (F1, AUC-PR, AUC-ROC).

MMBH's paper represents quite a comprehensive method and analysis of salary prediction using regression and classification techniques. I found another, more limited paper about salary prediction, "Random Forest for Salary Prediction System to Improve Students' Motivation" by Khongchai and Songmuang (KS). KS used a subset of techniques used by MMBH and had a significantly smaller feature-set. They obtained 13,541 previous graduate student records to train on and evaluated their models on 50 student samples using confusion matrix values.

KS went in depth on using the decision tree and random forest techniques across four salary bands, and provided insights into model improvement based on parameter tuning. The best random forest classifier had the higher accuracy (90.5%), as compared to the best decision tree classifier (61.4%). Ultimately, MMBH's detailed process and survey of models are good examples to learn from for this analysis, and KS's deeper look into a smaller set of techniques provides a more niche direction for improving individual models.

3 METHODOLOGY

3.1 Initial Development Process

From the outset of this project, I obtained a small dataset from Levels.fyi for academic use that contained 500 individual compensation entries. The feature-set included base salary, stock grant, bonus, total compensation (the sum of the previous three), company name, job level, location, gender, years of experience, years at the company, and the specific software engineering field (labelled as "tag"). In addition, it contained optional free form text that included sign-on bonus information, higher degree information, and more.

I initially planned to make classification-based predictions on four variables: salary, stock, bonus, and total compensation. I was going to create bands/ranges to facilitate the classification, using one of three mechanisms: (1) buckets based on static, manually chosen values (e.g., $25k buckets up to the maximum dataset value), (2) buckets based on dynamic, heuristically chosen values (e.g., buckets that increase progressively from a $20k range up to a $50k range), and (3) buckets based on percentiles (e.g., 5% percentile buckets). The benefit to these buckets is that they provide a psychological benefit beyond a simple point prediction: your compensation negotiations become more effective because your target value has some flexibility to it.

However, I ultimately decided to stick to only regression analysis. The complexity surrounding what bucketing mechanism to use and the compensation data's bias towards round numbers made the classification framing imperfect by design. The same sort of compensation "range" effect can be achieved by using regression analysis to make a point prediction and then building a confidence interval around it with the calculated root mean square error (RMSE).

3.2 Data Cleaning and Preparation

After deciding on the regression analysis direction, I found the full dataset on Levels.fyi by inspecting the web page traffic. It was available as a JSON file and contained almost 25k individual compensation entries. However, unlike my initial 500 entry dataset, this one was much less clean.

This dataset contained compensation entries for all kinds of roles, such as product managers and data scientists, not just software engineers. It contained inconsistent values for salary, stock, and bonus, where some entries contained the full numbers and others had it in thousands (e.g., "125,000" vs "125"). Some didn't even add up correctly to the provided total compensation. The locations were clean and consistent, but the company names, levels, and tags were not. They were frought with spelling, spacing, and capitalization differences (e.g., "Dropbox" vs. "DropBox" vs. "dropbox") and semantic equivalencies (e.g., "Operating Systems" vs. "OS"). The free-form text field with other details was difficult to parse and only contained non-empty, valuable information in less than 20% of entries. It contained information like whether the entry was a masters/PhD degree holder, but this was not consistently available and was heavily biased towards more recent entries, indicating evolution over time of the input mechanism, rather than an influx of higher-degree holders.

My first step was to filter the entries to only those who were Software Engineers, as this was the focus of my analysis. The total compensation field was always present and consistently formatted as a number in thousands (e.g., 125 instead of 125,000). I left this field as-is to maintain the ease of readability. I removed any entries that were missing salary data because this was a required field from my perspective. If stock and bonus information

were missing, I simply assumed them to be 0. I then scaled all the compensation-related fields to numbers in thousands. Sometimes, the provided stock values were the total over a four-year period, so I had to check if that value divided by four could work. Ultimately, the compensation scaling step required meticulous analysis of the dataset's ranges for salary, stock, and bonus. To ensure correctness, I added a safeguard that the resulting salary plus stock plus bonus had to correctly add up to the total compensation (within $20k on either side). Otherwise, I dropped those entries. All this filtering yielded 14,557 entries.

Next, I tackled the inconsistent company names and tags. Using a combination of fuzzy string matching via Levenshtein distance and some special casing, I was able to create mappings between datapoints that were similar. Whatever was the most popular name for a particular grouping became the new applied version. For example, "Cisco", "CISCO", and "Cisco Systems" would all be changed to the most popular version, which was "Cisco". This step helped provide unity to entries that the machine would think refers to different companies or tags, but were actually referring to the same thing.

Then, I set out to create dummy variables for each of the categorical columns. In order to prevent feature crowding, I only created dummies for those categorical values that appeared above a certain threshold. Locations and companies needed to appear at least 15 times, and tags needed to appear at least 25 times. These numbers were chosen as heuristics based on relative appearances. Using my domain knowledge, I also decided to create interaction term dummy variables for (1) company and level, and (2) company and location. The company x level interaction term is necessary because companies inherently pay differently based on level, but there are wildly different level names between companies (e.g., Google's "L3" vs. Facebook's "E3" vs. Microsoft's "59"). Given this inconsistency, it wouldn't make sense to include the levels as categorical features to analyze on their own. The company x location interaction term is necessary because it is based on a hunch that, while locations themselves might provide some premium to compensation because of the relative costs of living, this exact premium may differ based on company. I chose not to include a company x level x location interaction term because this seemed excessive and would balloon the feature count, even if it was plausible that compensation for company levels may differ by location.

Finally, I had to transform some last few columns. The "years of experience" and "years at the company" features needed to be normalized using a standard normal scaling. I performed a dry run training/test split of the dataset (the same one I would use during model selection) to calculate these numbers correctly. Moreover, for salary, stock, bonus, and total compensation, some boxplots and 2-D comparison plots revealed "trumpet shapes" of these target values, indicative of non-constant variance across their domains (see Figure 1). A common solution for this is to log transform these values, which is what I ultimately did. My final dataset contained log salary, log stock, log bonus, and log total compensation as targets. As features, it contained scaled years of experience, scaled years at the company, company dummies, location dummies, tag dummies, company x level interaction term dummies, and company x location interaction term dummies. The total feature count was 615.

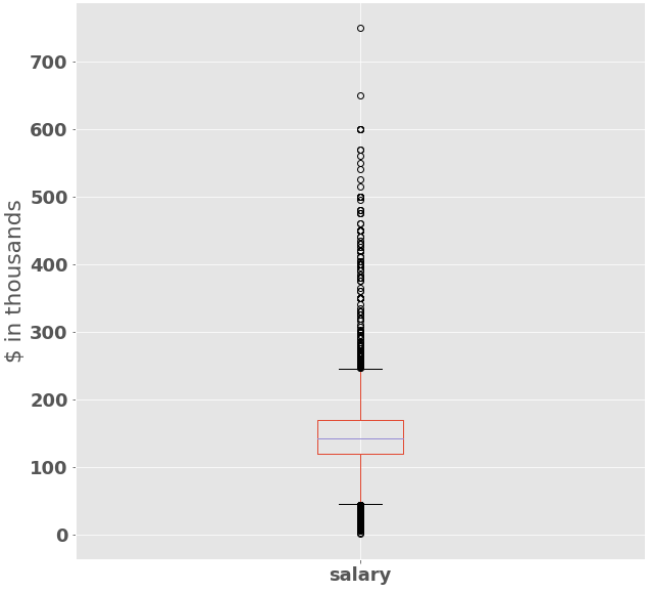

Fig. 1. Boxplot of the salary values. Most of the values center between $100k and $200k, but can go as high as $750k. There is a noticeable sparcity of values the higher you go. This means that the salary domain exhibits non-constant variance and may need to be transformed.

3.3 Model Selection

From the outset, I thought I would predict all four targets of total compensation, salary, stock, and bonus. However, as I started model selection, I thought it would make more sense to not directly predict total compensation, given how its individual components might vary wildly. I decided to just focus individually on salary, stock, and bonus. I could sum the predictions at the end and compare those values against the actual total compensation.

Since I was performing separate regression analyses on three targets, I decided to limit the types of regression model types I would try. I settled on using ridge, lasso, and partial least squares regression. Ridge and lasso would likely yield similar results, but I wanted to see if lasso would zero out certain coefficients that yielded little value, whereas ridge regression would keep them. Since partial least squares (PLS) regression and principle component analysis (PCA) usually yield similar results, and because there is some complexity around training and testing the PCA model, I decided to only use PLS. See the **Future Work** section for an explanation as to why the multilayer perceptron (MLP) regressor and K-nearest neighbors (KNN) approaches weren't used.

My first tests relied on the log transformed targets. I split the data 90% for training and 10% for testing. For

each of my models, I used 5-fold cross validation on the training set using negative mean squared error as a scoring mechanism. For the ridge and lasso models, I tested alpha values in the linear space between 10^{-2} and 10^{100}. In the log transformed case, the ridge regression model outperformed the others in all three cases: salary, stock, and bonus. This made sense to me, as I figured that each feature likely added some difference to the model, even if it was small. However, after performing prediction on the test set, building a confidence interval with one RMSE on either side, and untransforming the values, I noticed that the predictions on the lower and upper ends of the interval were very wide if the RMSE was high. The salary training RMSE (for its best model) was 0.1882, so this wasn't a big issue with the prediction range using the test RMSE. However, the stock and bonsus training RMSE's were 2.2599 and 2.7218, respectively. For context, an RMSE of 2.4789 on a re-transformed prediction of $50.2 would yield $598.8 on the upper end of the range. This is a function of multiplying the predicted value by $e^{\pm RMSE}$ when re-transforming from log.

In order to receive more reasonable values for my ranges, I decided to undo my log transformation of the target values. I brought them back to their direct values as numbers in thousands. I re-performed the exact model selection steps as above and yielded better training RMSE's and tighter ranges on the predictions. The table below shows the training RMSE's for each of the models on each of the targets, with the winning model bolded.

	SALARY	STOCK	BONUS
Ridge RMSE	**22.492**	56.564	**20.935**
Lasso RMSE	23.372	**56.529**	21.362
PLS RMSE	27.762	65.916	22.609

Notice how similarly the ridge and lasso perform against each other in almost every case, with the lasso model barely winning out in the stock case. The regularization/alpha values for the winning salary, stock, and bonus models were 0.8697, 0.01, and 2.0092, respectively.

4 Results & Analysis

In order to test our winning models, I performed predictions on the test set, i.e., the remaining 10% of data, or about 1,456 entries. The resulting test RMSE's for salary, stock, and bonus were 23.463, 59.156, and 20.933, respectively. They measured up against the training RMSE's quite well, indicating that the models weren't overfitting to the training data. Another interesting value is the test RMSE of total compensation (where the "predicted" values are actually the sums of the individual predictions for salary, stock, and bonus): 77.564.

Keeping in mind that these represent numbers in the thousands, we can build ranges around around our point predictions. One RMSE deviation in either direction yields a confidence interval of about 68% and two RMSE deviations yield a confidence interval of about 95%. For negotiation purposes, we are more likely to care about what most other people like us are making, rather than what the overwhelming majority is making. Thus, it may be sufficient to provide a negotiation range with one RMSE, rather than two RMSE's.

The salary RMSE itself is quite tight, relatively speaking. A point prediction within a range of $50k can provide a good idea about salary expectations. On its face, the stock RMSE seems to have some room for improvement. However, stock is also known to be more flexible than other forms of compensation, and is often used as a way to sweeten total compensation, when salary is difficult to negotiate on. Nevertheless, the wide range provides some uncertainty. The bonus RMSE seems to be low and on par with the salary RMSE. This is misleading, however, because bonuses tend to be significantly smaller than salaries, perhaps on the order of 5 – 15% of salary. Thus, a $21k bonus RMSE also leaves some room for improvement.

Performing a coefficient analysis on the winning models can be useful for understanding the effects of various features. Note that, while the coefficients may seem to correlate directly to increases/decreases in the compensation targets, this is misleading. The alpha value used for regularization affects the value of these coefficients. Thus, while they shouldn't be taken at face value, the coefficients provide an important look into their relative effects. The results show some well-known phenomena in the tech industry. For instance, California and New York are major tech hubs, and this is reflected in the coefficient analysis (see Figure 2). Moreover, for particular "hot" skills like ML/AI, Distributed Systems, and Mobile Apps, there is known to be a market premium, whereas less desireable skills like Testing (SDET) have market discounts (see Figure 3). The results also show some less expected insights in the data. For example, Netflix dispenses far higher salaries than any other company, including those in its Facebook, Amazon, Apple, Netflix, Google (FAANG) cohort (see Figure 4). Further research, beyond this paper, suggests that this is because of Netflix's hiring practices. Unlike its peer institutions, Netlifx only hires senior software engineers; by not hiring new grads / junior engineers, its salaries tend to be higher overall. Another unexpected insight is that fintech companies tend to provide larger bonuses than other companies (see Figure 5). Further research suggests that this is a holdover from the finance industry overall, where high bonuses are commonplace.

These are all interesting insights to observe from the models, but this coefficient analysis does come with some risks. Multicollinearity might be an issue in the feature-set, but it is unclear how it affects results in this case. This is exacerbated by the presence of interaction terms. It's useful to measure the individual effects of each feature, holding all others constant, but the way the models were built shows that there is a relationship between the interaction terms and the features they are built from. We may indeed want to look at the effect of something like location and the additional effect of company location,

but there may be some obscurity in how to interpret results. For example, one consequence of this is how the coefficients "bounce" the ultimate point prediction back and forth. For instance, a company like Google has a premium reflected in its salary model coefficient (i.e., it has a positive coefficient), and the Google levels individually bounce the final salary up or down (i.e., Google L3 coefficient is negative, whereas the Google L6 coefficient is positive). This makes coefficient analysis more complex because each one needs to be taken in context with its related coefficients.

5 Future Work

There exist a number of untaken paths in this project that can yield further interesting results. For instance, more regression-based approaches can be attempted to yield better results. I considered using an MLP regressor model, going so far as to perform some light experiments. However, early tests showed that training such a model would be slow, and that a grid search would be necessary to adequately tune the model hyper-parameters. I chose to avoid this approach given the speed and relative accuracy of the ridge, lasso, and PLS regression models. Moreover, I valued the ability to interpret my results, not just achieve effective prediction. Coefficient analysis can yield some interesting insights that wouldn't be possible with MLP. This was also the reason why I nixed the KNN approach for this paper's purposes.

One obvious addition, based on earlier discussion, is to introduce classification models into the mix. The related works show that success can be achieved with a variety of classification-based models. One advantage this dataset has over MMBH, for instance, is that it is three times larger. Application of MMBH's models might yield better results here. However, the features in that analysis and this one differ. Nevertheless, the issue of how to bucket compensation ranges in each of the three categories of salary, stock, and bonus persists. Attempting various experiments on this point would be critical.

Lastly, one interesting, unused source of information is the free-form "other details" in the original data. It contains useful information about sign-on bonuses and degrees, but ultimately proved non-existent in much of the entries. With time, as the dataset from Levels.fyi increases in size and aggregates more of this kind of data, it would be interesting to re-run this paper's analysis. Domain knowledge suggests that a PhD greatly affects total compensation on the order of tens of thousands of dollars. It would be interesting to see how something like a Master's degree affects compensation, if at all.

6 Reflection

A majority of this analysis' effort went into data cleaning and preparation. Given the need for clean data for effective prediction, the amount of time spent here was worth it. While the code for regression model fitting and prediction is not very complicated, a fair amount of work was put into writing reuseable software. Since there were three targets in this analysis, rather than one, there is value in writing functions. It reduces copy-paste bugs and helps ensure consistency. Yet further effort was exerted in analyzing the resulting regression models' coefficients to glean interesting insights.

I already had knowledge about regressions, including more advanced regression models that use regularization to penalize more features. This analysis allowed me to formalize that learning in an interesting problem. One thing I learned from this project was a number of classification models. While I didn't ultimately apply them in this project, I still performed some research. MMBH was a great guide in this regard because it laid out a nice roadmap for techniques I briefly read about and techniques that were entirely new, like ensemble voting classifiers. Another thing I learned was how to use a library called patsy, which provides a nice interface for creating dummy variables, even with interaction terms. I also learned how to use a fuzzy string-matching library called fuzzy-wuzzy to help with standardizing the input data. I had to play around with the tolerance levels, so as to not accidently combine datapoints that weren't actually referring to the same thing.

Overall, this was a very interesting problem for me to tackle and related deeply to my own personal interests as a software engineer. I hope some of these insights and models can help others as well.

7 Codebase

The project codebase can be found here: https://mit.cs.uchicago.edu/mpcs53120-spr-20/sayzman

References

[1] Khongchai, Pornthep, and Pokpong Songmuang. "Random Forest for Salary Prediction System to Improve Students' Motivation." *2016 12th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*, 2016, doi:10.1109/sitis.2016.106.

[2] Kolakowski, Mark. "Compensation in Financial Services." *The Balance Careers*, The Balance Careers, 1 Apr. 2019, www.thebalancecareers.com/compensation-in-financial-services-1286874.

[3] Le, Alexandria. "Why Doesn't Netflix Hire New Graduate Engineers?" *Quora*, 2019, www.quora.com/Why-doesn-t-Netflix-hire-new-graduate-engineers.

[4] Martin, Ignacio, et al. "Salary Prediction in the IT Job Market with Few High-Dimensional Samples: A Spanish Case Study." *International Journal of Computational Intelligence Systems*, vol. 11, no. 1, 2018, p. 1192., doi:10.2991/ijcis.11.1.90.

[5] Mohiuddin, Zaheer, and Zuhayeer Musa. *Levels.fyi*, 2020, www.levels.fyi/.

[6] "Stack Overflow Developer Survey 2019." *Stack Overflow*, 2019, https://insights.stackoverflow.com/survey/2019.

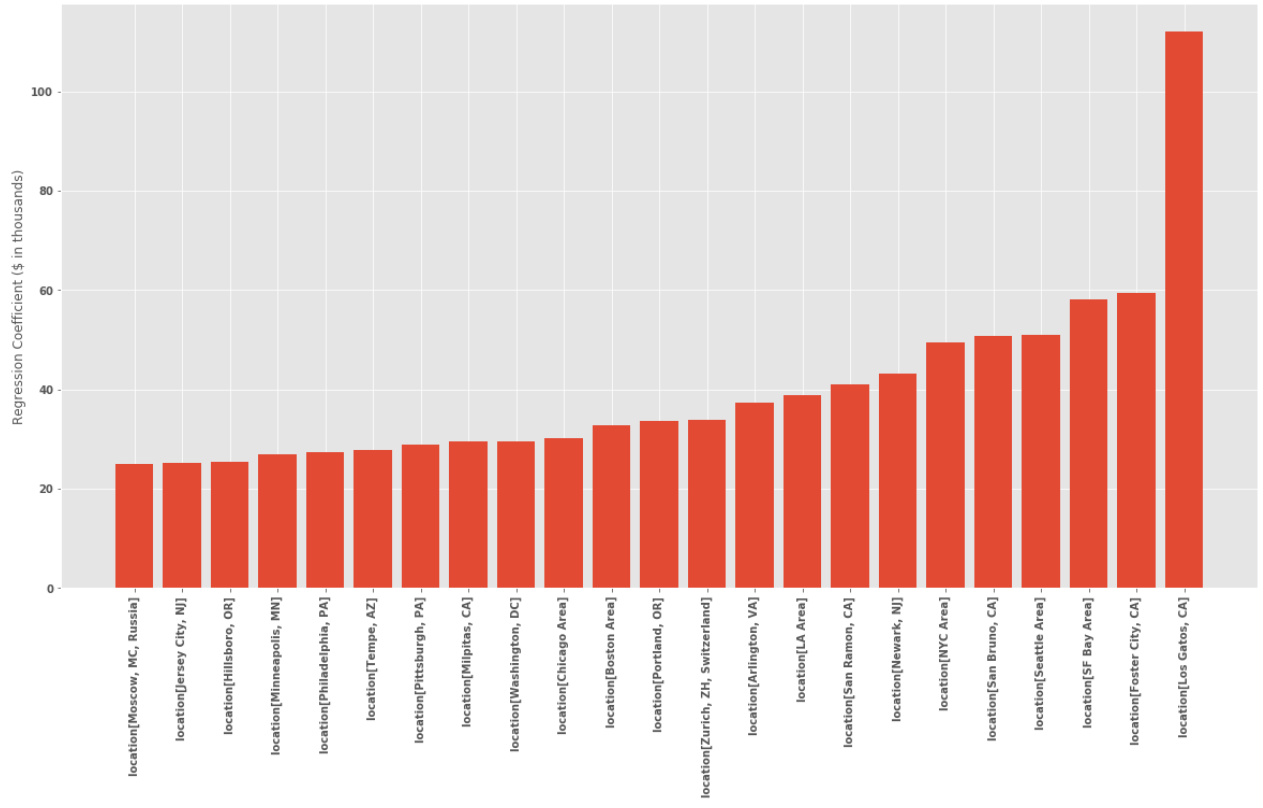

Fig. 2. The top coefficients in the best salary model based on location. Known tech hubs in California and New York rise to the top.

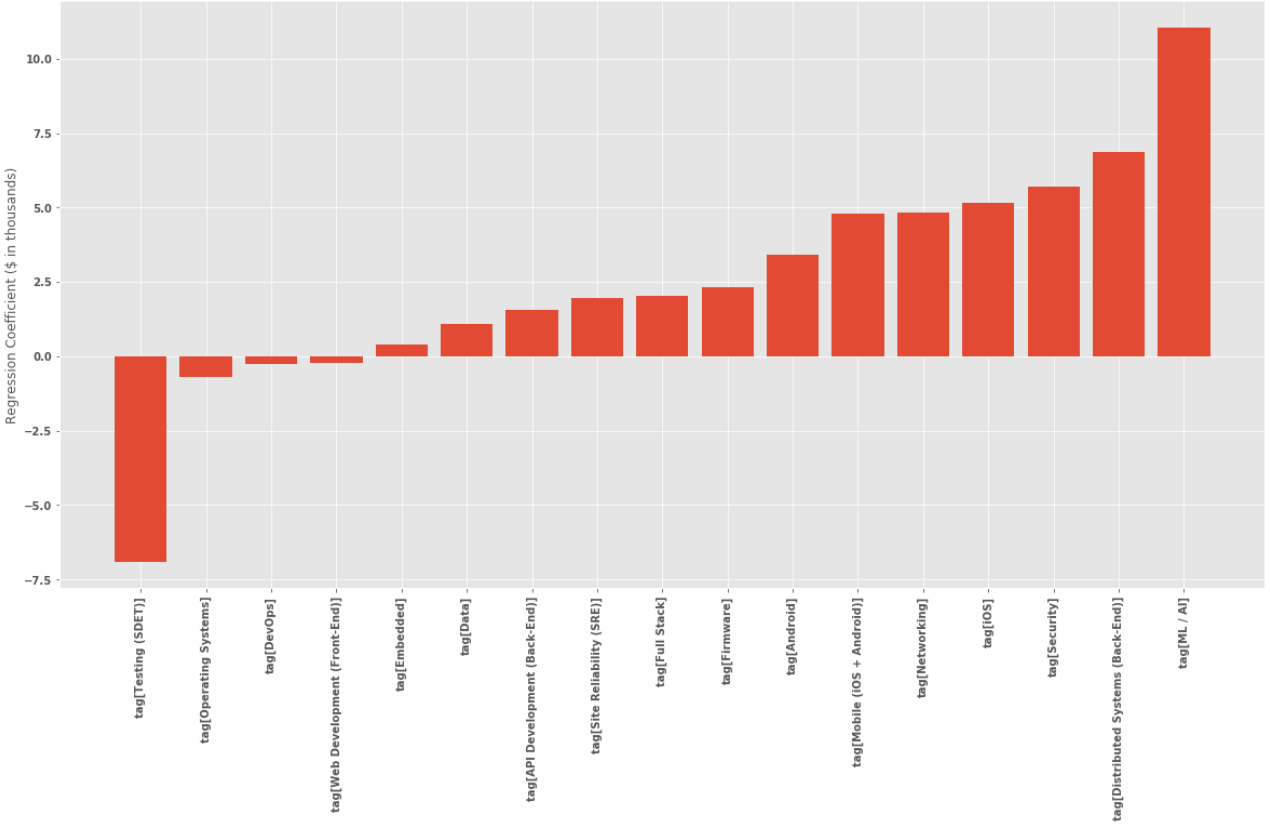

Fig. 3. The coefficients in the best salary model based on tag. Known “hot” skills like ML/AI, Distributed Systems, and Mobile Apps rise to the top. Less desireable skills like Testing fall to the bottom.

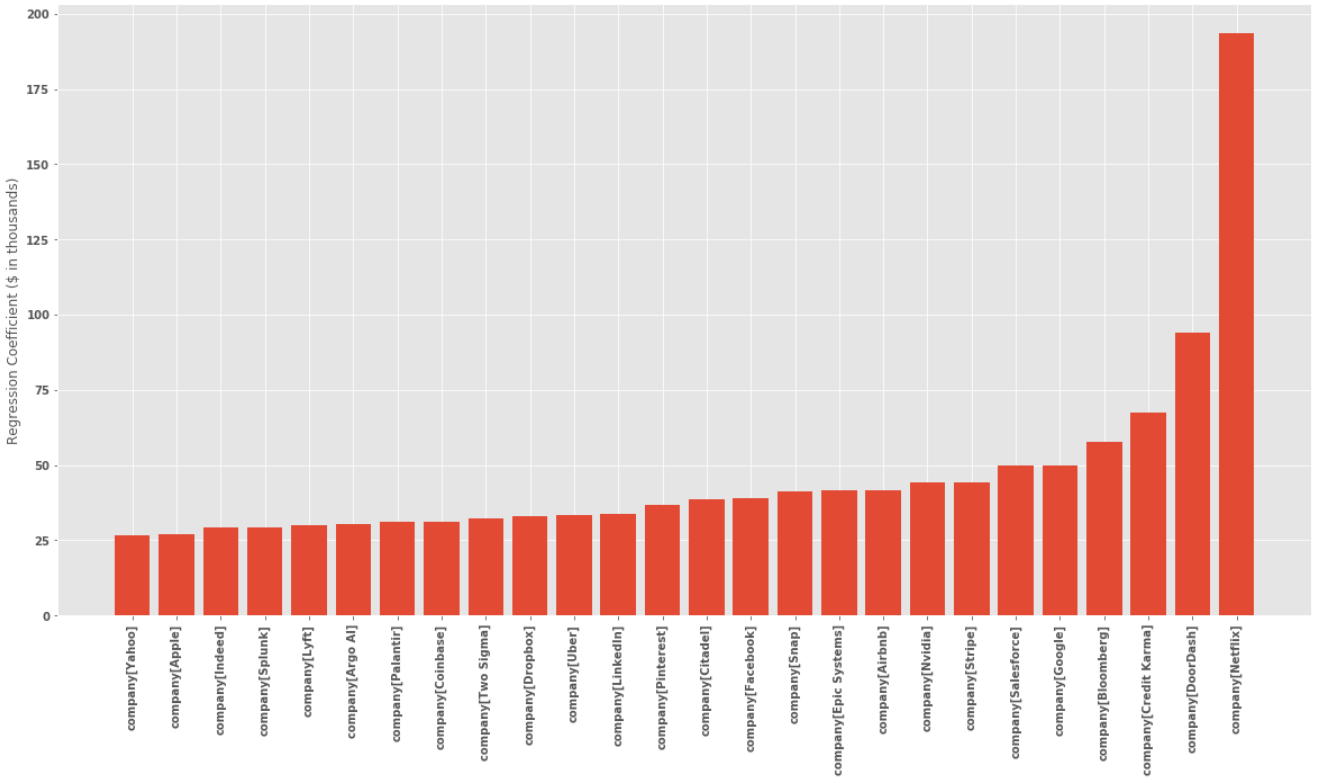

Fig. 4. The top coefficients in the best salary model based on company. Netflix rises noticeably above the rest.

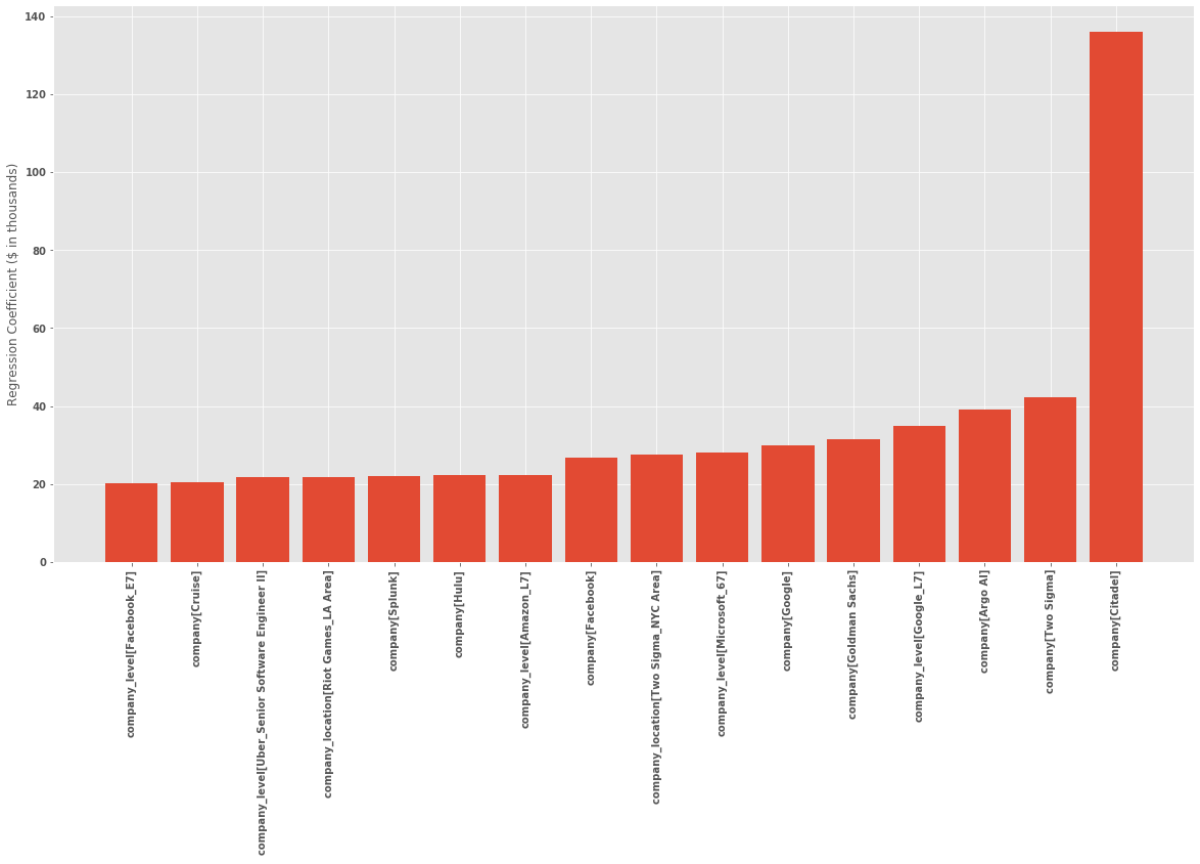

Fig. 5. The top coefficients in the best bonus model. Fintech companies like Citadel and Two Sigma make noticeable appearances.