CS 229 Project | Final Report | Autumn 2016

Predicting Compensation for Job Seekers

Category: General Machine Learning
Team members: Megan Fazio (mefazio@), Jen Kilpatrick (jdki@), Darren Baker (drbaker@)

Problem background and related work

One of the top questions on most job seekers' minds is a simple one: "how much can I expect to be paid?" Unfortunately, the simplicity of the question can be deceptive: predicting compensation is a surprisingly difficult task, not only because most companies are reluctant to share details about how much they pay, but because wages and salaries can vary substantially based on factors like geography, industry, and employee experience.

In our CS229 project, **our team attempted to determine how accurately we could estimate base salary** (excluding bonuses or other forms of additional compensation) **for a particular job.** We obtained a large set of salary data from Glassdoor, a well-known site that collects information from its users about their employment experiences in terms of job satisfaction, compensation, and more. Since our desired output was a real-number value, we focused on exploring a variety of regression techniques to build our predictions.

Because detailed salary datasets spanning millions of jobs were largely unavailable before the advent of crowd-sourcing sites like Glassdoor, most previous efforts in salary prediction (including several previous CS229 projects) have applied similar regression techniques but focused on using more limited data to answer more targeted questions, such as how college choices affect individuals' future earnings ([1], [2], [3], [4]) or whether gender pay inequities exist for specific jobs ([5], [6]). The Glassdoor team has developed proprietary regression-based salary estimation methods for their more general dataset, but not published detailed results. Our goal in this project was to replicate Glassdoor's results and explore possible extensions to their methods to improve prediction accuracy.

Data source overview

The data science and economics research teams at Glassdoor graciously provided us with an anonymized dataset containing more than 2 million salary datapoints from the United States contributed by their users over the past several years. The main elements of the dataset are the various fields that are supplied by users when they enter a salary report through the Glassdoor survey. Here is an outline of the most important fields:

- **Job Title:** This is the user's role in their organization, such as "supply chain analyst", "account manager", "network administrator", or "manufacturing engineer". Users can supply any title they like, though they're encouraged to choose from a list of common titles (resulting in some degree of standardization).
- **GOC Attributes:** GOC stands for "Glassdoor Occupation Category," which is a set of multiple fields containing proprietary classification properties that the Glassdoor team developed to try to smooth out some of the noise in job titles across a range of industries and employers. (We did not use these features for our own predictions, but instead used them to construct "benchmark" models to evaluate the performance of our features against those of the Glassdoor data science team.)
- **Metro Area:** This is a field capturing the location of the user's job. Glassdoor offers users a custom list of metropolitan areas that corresponds roughly to the list used by the US Census Bureau.
- **Employer Name:** The name of the organization where the user works.
- **Employer Metadata:** There are several features here, representing facts like the size of the employer (in terms of employee count), the broad industry category (retail, restaurants, manufacturing, etc.), and the employer "type" (private company, public company, government agency, etc.). Some of this data is supplied by users, but a portion of it has also been associated with each employer by the Glassdoor team.
- **Employee Metadata:** This includes features giving more detail about the user's job status, such as employee "type" ("normal" vs. intern, contractor, etc.), years of (relevant) experience, and similar fields.
- **Base Salary:** This is the key field that we use for training and prediction. Users are given the option of submitting a value for hourly, monthly, or annual time periods, so there's also a field to encode this choice.

Structuring the learning problem: inputs/outputs and model specificity

The output value for our prediction task was always the base salary that we believed corresponded best to the input values. The trickier part of this problem was choosing the set of inputs to use in constructing a regression model. From the perspective of a person seeking a salary prediction -- whether a job seeker, employer, or researcher -- there are multiple plausible levels of specificity that might characterize a query. For example:

- A hiring manager or HR representative working for a specific company and seeking to set an appropriate salary for an open role would naturally care about getting a prediction that is as specific as possible to their

company. Therefore, this person might want to supply as much information as possible (including employer name, job title, and other relevant metadata) to get a customized, higher-precision estimate.
- Alternatively, a job seeker who has a particular job title in mind but is open to considering various industries and employers might only supply the job title and limited other data (such as their location and years of work experience) to understand average salary levels for that job across a range of employers.

In order to account for these possible variations, we decided to develop a series of models that ranged from relatively general to quite specific in terms of their input values. We ultimately settled on a collection of **6 different models,** with the first one serving as a general baseline, and each succeeding model adding a single (conceptual) feature to make the prediction more specific. Here is the list of models we created:

1. The baseline, which makes a prediction based on job title, metro, and the employee's years of experience.
2. Same features as model 1, plus the employee's "status" ("regular" vs. contractor, intern, etc.)
3. Same features as model 2, plus the employer type (private company, public company, gov't agency, etc.)
4. Same features as model 3, plus the employer's size (in terms of employee count)
5. Same features as model 4, plus the employer's "category" or industry (retail, technology, etc.)
6. Same features as model 5, plus the employer's name

We applied the approaches described in the next section to each of these 6 models to compare their performance. Unsurprisingly, the more specific models tended to produce salary estimates with lower error, as we show below.

Modeling and prediction approaches

We followed a multi-stage process of designing, implementing, and enhancing our regression models in order to maximize the accuracy of our predictions. These were the key steps in our process:

Data cleansing and normalization

The first step was to perform a "sanity check" on the data and perform several cleaning and transformation tasks to ensure that the data was in a form suitable for model building. Examples of actions we took at this stage were:

- Removing salary records that appeared invalid or suspicious -- e.g., those that reported annual salaries of zero or values that were less than a few thousand dollars per year.
- Normalizing text to remove special characters that interfered with matching job titles and other text fields.

We also considered at this stage whether to annualize salary figures in the dataset that were reported as monthly or hourly pay rates. However, in our exploration of the data and our discussions with the Glassdoor data team, it wasn't clear that annualized versions of these figures were perfectly comparable with salaries that had been reported on an annual basis in the first place. Since annual salary figures made up the large majority of the dataset, we ultimately decided to use only those values in our models and set aside the other figures for later investigation.

Feature engineering

One of the key challenges in working with the Glassdoor data was that several of the most important fields in the dataset -- including job title, metro name, and employer name -- were categorical, with a huge number of possible values. Although the metro field had "only" O(1000) values, the job title field contained nearly 100,000 unique values, and the dataset also referenced more than 200,000 distinct employers. (The text normalization described above provided some benefit in consolidating near-duplicate values, but the number of feature values remained massive.)

One computational challenge that arises with large numbers of categorical feature values is simply the time and memory burden of working with matrices with billions or even trillions of elements; we discuss this problem in the implementation section immediately below. More important for the prediction performance of our models was the issue of **sparsity** in fields like job title. A human may recognize that the job title strings "software engineering director" and "director of software engineering" are likely equivalent, or that "customer service associate" and "customer service representative" are at least linked by the fact that they involve customer service. However, a standard learning algorithm will treat these values as completely distinct (and therefore fail to capitalize on any underlying shared data between them) unless we introduce features to capture some aspects of their relationship.

Glassdoor created their proprietary "GOC" model to map job titles to standardized feature values representing certain job types, and we spent time developing our own feature engineering code in an attempt to produce features with predictive power similar to GOC classifications. We settled on a set of ~100 features using the following templates:

- Multiple features represented job title "prefixes" and "suffixes" that usually served as modifiers to express the seniority of a person's role. Prefix examples included terms like "junior," "senior," or "assistant," while suffix examples are numeric elements like "1," "2," "3" (or "I," "II," "III") and terms like "trainee."
- Many features attempted to separate the "role" portion of a person's title from the functional area also described in their title. (As a simple example, a "finance analyst" title indicates a person in an "analyst" role in the "finance" organization.) We separated dozens of the most common roles into separate features, leaving a (modified) job title that usually approximated the person's organizational/functional affiliation.

Beyond the job title field, we also did additional feature engineering to reduce noise and dimensionality in the data:

- We took the integer-valued employee count field supplied by Glassdoor and transformed it into a discrete-valued field with ~10 different size buckets, since it seemed reasonable to believe that a company with 50,000 employees might pay differently than a company with 500, but less reasonable to assume that a company with 502 or 550 employees would pay very differently than a company with 500.
- We also identified the least-common values in the metro area field and consolidated them, on the assumption that the difference in pay levels between small and predominantly rural cities like Abilene, TX and Pocatello, ID would be much smaller than the difference between (say) Abilene and New York City.

Implementing basic linear regression

Throughout our project, we used the statistical programming language R to build our regression models. As mentioned above, **one early challenge with implementing linear regression on our large dataset was our limited memory and computation time.** The standard squared-loss (a.k.a. "least squares") linear regression code in R uses the normal equations and matrix factorization to compute a result, but this is problematic when our feature matrix has dimensions of O(1M) rows and O(250K) columns. We explored several ways to address this limitation:

- Thresholding the data to eliminate values that appear very infrequently (and therefore also eliminate their corresponding binary features)
- Running our models only on a randomly selected subset of the original dataset
- Implementing our own stochastic gradient descent code to see whether we could train our model faster if we relaxed the constraint of finding the true optimum given by the normal equations

Each of these approaches did partially alleviate the problem of memory limits and long training times, but at the expense of some predictive accuracy. Fortunately, we discovered later that Stanford statistics professors Jerome Friedman and Trevor Hastie have developed a much faster regression toolkit for R, called glmnet [7], which uses a coordinate descent algorithm that works well for sparse data like ours. Switching to this algorithm allowed us to use our full dataset for prediction without significant computational constraints.

Adding regularization and normalizing salaries with logarithms

Although our initial results seemed reasonable (see the results section for details), we still wanted to consider various methods for improving the accuracy of our predictions. We employed two strategies here:

- **Regularization:** We initially suspected that because our data was relatively noisy, our models might benefit from regularization. We decided to experiment with 3 different types: LASSO (which penalizes the L1 norm of the weight vector β), Ridge (which penalizes the L2 norm), and Elastic Net (which penalizes a linear combination of the L1 and L2 norms). Specifically, we used the α parameter in the glmnet package, which defines the linear combination of norms as follows [7]:

$$(1-\alpha)/2||\beta||_2^2 + \alpha||\beta||_1$$

  Setting α = 1 therefore gives LASSO regularization, while α = 0 gives Ridge and 0 < α < 1 gives a spectrum of Elastic Net configurations between these two extremes. (Note that the quantity above is multiplied by another parameter λ to determine the magnitude of the penalty added to the loss function.)

- **Log normalization of salaries:** Salary values have a naturally right-skewed distribution, as their values are rarely very small (and cannot be negative), but there are always some highly-paid individuals who have salaries substantially larger than most others -- say, $500K or more per year. Since linear regression uses squared loss, prediction errors on these outlying points create huge losses and therefore pull the entire weight vector toward them. We applied a log transform to all of the salary data to limit the effect of these large salary values.

Exploring alternate models

Finally, we experimented with **tree-based regression models** (decision trees and random forests) to determine whether they might deliver improved accuracy. Unfortunately, we weren't able to make much progress down this path due to **computational constraints.** Decision trees and their derivatives (including random forests) are constructed by recursively considering every possible "split" of the training data to find the one that best separates the training examples by the value of the dependent variable (in our case, salary). Because the number of possible splits over categorical predictors is exponential in the number of possible values, and our data included at least 3 predictors with huge numbers of values, we quickly concluded that building trees using any of these features was infeasible.

Results and conclusions

For each of our 6 models and each modeling approach (regularized vs. not, log salaries vs. original), we split our dataset into training and test sets using proportions of 80% and 20%, respectively. As noted above, we also constructed a simple benchmark for each of the 6 models by using Glassdoor's "GOC" attributes in place of the original job title features. Our best results (with corresponding λ value) for each model variation are summarized in the tables on the following page: we've reported average training and test error, but also the median *percentage* error between our predictions and the true salary values because this is Glassdoor's preferred accuracy metric.
Key take-aways include the following:

- **Our feature engineering appears to do a reasonable job replicating Glassdoor's GOC model,** as our prediction error on non-regularized models is fairly close to that observed in the benchmark models (between 15 and 19%).
    - As an additional point of validation, we observed that **the majority of our engineered features had coefficients that were highly statistically significant,** suggesting that these features are highly predictive of salary levels. (Due to space constraints, we could not include these coefficients here.)
- **Training error is only slightly smaller than test error in most cases,** suggesting (unsurprisingly) that our **linear regression model has relatively high bias, but also that overfitting is not a major problem.**
    - This observation is reinforced by noting that **regularization with small values of λ delivers some small gains in predictive accuracy, but no major boosts.**
    - We believe that overfitting is lessened not only by the relatively low variance of the linear model, but also by our very large volume of training data and by the partial reduction in sparsity that we were able to achieve through feature engineering.
- As mentioned previously, **the more specific models (i.e. those including more features) generally deliver lower error levels than more general models;** model 1 is the most general and has the lowest accuracy, while model 6 is the most specific and has the best accuracy (though also the most overfitting).
- It is clear that **predicting the log of salary values is substantially more accurate than predicting the original value,** which offers support to our hypothesis that non-log predictions are at least somewhat skewed by the minority of high-salary outliers in the dataset.
- Finally: **salaries in the real world have a lot of variation that can't be fully accounted for by only the features in this dataset,** as we see in the relatively low R^2 values for even the best regression models.

Future work

The Glassdoor data science team noted in our discussions that they have been able to achieve median test error as low as ~10-11% on certain subsets of their data. We believe that an attractive avenue for future work would involve trying to match or exceed this level of performance by:

- **Adjusting salaries for inflation:** We did not adjust salary values for inflation or wage growth over the past several years. Although inflation has been relatively low in the overall economy, some industries (such as technology) have experienced average wage growth of at least a few percent per year. Accounting for these kinds of changes might easily deliver several percentage points of accuracy gains in our predictions.
- **Adding interaction terms:** Our feature engineering does give up some "implicit" interaction when categorical variables are binarized. We would experiment with re-adding explicit interaction terms to capture non-additive relationships in our current set of predictors.
- **Collecting more detailed data from users:** This is not something we could do on our own, but it's a strategy that Glassdoor has begun piloting in a new personalized salary estimate tool. Since salaries are likely affected by a range of employee attributes that aren't captured in the current dataset -- for example, education level or the nature of previous work experience -- collecting more detailed data might allow further improvements to our predictions.

	Original salary predictions				
Model #	**Regularization Type**	R^2 value	Mean absolute training error	Mean absolute test error	**Median percent test error**
1	***Glassdoor benchmark***	*0.6104*	*$24,614*	*$24,608*	*18.7*
	No regularization	0.6196	$23,853	$24,373	18.6
	Lasso (λ = 1)	0.6196	$23,854	$24,370	18.58
	Ridge (λ = 1000)	0.6191	$23,869	$24,380	18.58
	Elastic Net (λ = 10)	0.6191	$23,868	$24,366	**18.57**
2	***Glassdoor benchmark***	*0.6104*	*$24,614*	*$24,608*	*18.7*
	No regularization	0.6206	$23,824	$24,461	18.60
	Lasso (λ = 1)	0.6205	$23,824	$24,458	**18.58**
	Ridge (λ = 1000)	0.6200	$23,840	$24,469	18.60
	Elastic Net (λ = 10)	0.6201	$23,839	$24,453	**18.58**
3	***Glassdoor benchmark***	*0.6283*	*$24,330*	*$24,305*	*18.49*
	No regularization	0.6287	$23,567	$24,202	18.47
	Lasso (λ = 1)	0.6287	$23,568	$24,199	18.46
	Ridge (λ = 1000)	0.6278	$23,594	$24,219	**18.45**
	Elastic Net (λ = 10)	0.6282	$23,583	$24,194	**18.45**
4	***Glassdoor benchmark***	*0.6341*	*$24,142*	*$24,147*	*18.4*
	No regularization	0.6336	$23,412	$24,042	18.49
	Lasso (λ = 1)	0.6335	$23,412	$24,039	18.47
	Ridge (λ = 1000)	0.6324	$23,449	$24,069	**18.43**
	Elastic Net (λ = 10)	0.6331	$23,428	$24,034	18.47
5	***Glassdoor benchmark***	*0.6450*	*$23,701*	*$23,753*	*18.28*
	No regularization	0.6464	$22,999	$23,634	18.30
	Lasso (λ = 1)	0.6464	$22,999	$23,631	18.29
	Ridge (λ = 1000)	0.6456	$23,023	$23,647	**18.26**
	Elastic Net (λ = 10)	0.6459	$23,015	$23,624	18.27
6	***Glassdoor benchmark***	*0.7466*	*$20,890*	*$22,453*	*16.78*
	No regularization	0.7488	$19,386	$22,578	16.92
	Lasso (λ = 1)	0.7486	$19,391	$22,527	16.87
	Ridge (λ = 1000)	0.7482	$19,409	$22,575	16.90
	Elastic Net (λ = 10)	0.7463	$19,480	$22,388	**16.73**

	Log salary predictions *(some models omitted for brevity)*				
Model #	**Regularization Type**	R^2 value	Mean absolute training error	Mean absolute test error	**Median percent test error**
1	No regularization	0.6450	$24,151	$24,805	17.5
	Lasso (λ = 0.00001)	0.6450	$24,154	$24,804	**17.48**
	Ridge (λ = 0.01)	0.6446	$24,185	$24,825	17.53
	Elastic Net (λ = 0.0001)	0.6447	$24,174	$24,807	**17.48**
6	No regularization	0.7833	$19,329	$22,296	15.21
	Lasso (λ = 0.0001)	0.7777	$19,610	$22,149	15.16
	Ridge (λ = 1000)	0.7829	$19,390	$22,284	15.28
	Elastic Net (λ = 0.0001)	0.7817	$19,431	$22,190	**15.14**

Tables 1-2: Prediction errors for multiple models using both original salary values and log salary values.
We considered many values of lambda for each type of regularization;
the best value we found for each type is listed in parentheses.
The best overall values of median test error for each of models 1-6 are highlighted in yellow.

References

[1] James, Estelle, et al. "College quality and future earnings: where should you send your child to college?." *The American Economic Review* 79.2 (1989): 247-252.

[2] Rumberger, Russell W., and Scott L. Thomas. "The economic returns to college major, quality and performance: A multilevel analysis of recent graduates." *Economics of Education Review* 12.1 (1993): 1-19.

[3] Agrawal, Monica et al. “Prediction of Post-Collegiate Earnings and Debt.” CS 229 project, autumn 2015: http://cs229.stanford.edu/proj2015/212_report.pdf.

[4] Strand, Miranda and Tommy Truong. “Predicting Student Earnings After College.” CS 229 project, autumn 2015: http://cs229.stanford.edu/proj2015/209_report.pdf

[5] Becker, William E., and Robert K. Toutkoushian. "Measuring gender bias in the salaries of tenured faculty members." *New Directions for Institutional Research* 2003.117 (2003): 5-20.

[6] Wood, Robert G., Mary E. Corcoran, and Paul N. Courant. "Pay differences among the highly paid: The male-female earnings gap in lawyers' salaries." *Journal of Labor Economics* (1993): 417-441.

[7] Friedman, Jerome H. et al. “Package ‘glmnet’: Lasso and Elastic-Net Regularized Generalized Linear Models.” R project documentation, 2016: https://cran.r-project.org/web/packages/glmnet/glmnet.pdf