Salary Prediction Using Job Descriptions

By: Max Breslauer-Friedman, Ryan Looney, Qixiang Jiang

Introduction

It has become more common for companies and recruiters to include salary information in job listings, even with some states mandating it by law. However, an immense majority have no salary information, or merely a very broad range. As students who are in the process of applying to jobs, we wanted to tackle this issue by trying to predict salaries based solely on the job description we were reading. This problem is particularly interesting, because it doesn't consider things like job title and experience level which can all be very misleading as many companies use different systems and conventions to determine these labels. Using the job description, we are given a large amount of context into the hard and soft skills required by the employee as well as things like perks and team culture, which all can give insight into how much the salary might be. Ultimately, in this project we study two different featurization techniques (Term Frequency-Inverse Document Frequency and Bag of Words) as input to three different models (Linear Regression, Support Vector Regression and Feed Forward Neural Network) and analyze the Mean Absolute Error to determine which approach is most accurate in predicting salaries. We also allow users to find job descriptions on their own, and have a model predict the salary of this job.

Data

Linkedin Job Postings 2023 from Kaggle

The LinkedIn Job Postings 2023 dataset from Kaggle provides around 14,000 job postings with job title, description, max salary, min salary, pay period, location and many more data points. For the sake of this evaluation project, we will only need the job description and pay related columns. As there is a max and min salary for most listings, while some have a median salary, an “expected” salary can be calculated for all listings by using the average of max and min, or simply using the median if provided. Additionally, the pay period is either YEARLY, HOURLY, MONTHLY, so the pay listed will be calculated for all to determine the annual salary. It appears from Kaggle that around only 40% of the data has salary information, reducing the size of the data set to about 5,000 entries, but still enough data to train and test various models. This also ensures there is not too much data to the point where our laptops would struggle to train more complicated models.

Models

Linear Regression

For this project we used the Linear Regression model from sklearn to run many experiments and better understand our data, as we figured it would have the fastest train time. We would then reduce the number of data points and features to allow for faster training for the more heavy duty models like SVR and the NN.

The first set of experiments were run using TF-IDF as the featurization technique. Inherently, TF-IDF accounts for unknown words by scoring them as zero in the matrix. We ran tests using the resulting matrix on linear, lasso, and ridge regression, to study how overfitting would affect our results. We used the best of these three models for all the following linear regression experiments, which was Lasso regression. Although it seemed more overfit than Ridge regression, it was selected as it typically is known to perform better at feature selection, which was important when we were trying to reduce the number of features. To try and improve the results of our model we downsampled the data, reducing the dataset by almost 50%. Using this downsampled dataset, we ran tests on the size of the training set used, which was 80%. Finally, we used all the previous optimal settings to find the optimal max_df value, which is a parameter in the tfidfVectorizer. This value removes any tokens in the corpus that appear in more than max_df-percent of the documents. We found that 0.25 (25%) was the optimal value here.

Fewer experiments were run using BoW featurization, as it was immediately clear that the results were less accurate using this technique on linear regression, but we didn't eliminate this option for future models.

Support Vector Regression

For support vector regression, we utilized the downsampled data gathered from the linear regression experiments. This data was featurized using Bag-of-Words (BOW) featurization (binary) and Tf-Idf featurization.

For the SVR models, we wanted to test out different values of the 'c' hyperparameter, which specifies a level of tolerance of missclassification for the hyperplane that is used to discriminate between classes in SVM [1]. For each model run, we tested the following values of c: 1, 10, 100, 250, 500, 750, 1000, 2000, 3000, 4000. We also tested different kernel functions; 'linear', 'poly' and 'rbf', creating 2 models for each kernel function, one for BoW and one for TF-ID. This lead to 6 total models to train and test, for all the list values of c. We noticed the epsilon hyperperamater had little effect in our experiments, thus used the default value for all models.

Neural Network

For our Neural Network model we constructed it using keras. We used the downsampled data set that was constructed during the experiments with the Linear Regression models. For featurization, we used TF-IDF vectorization, with a max_df of 0.25, a parameter that was also

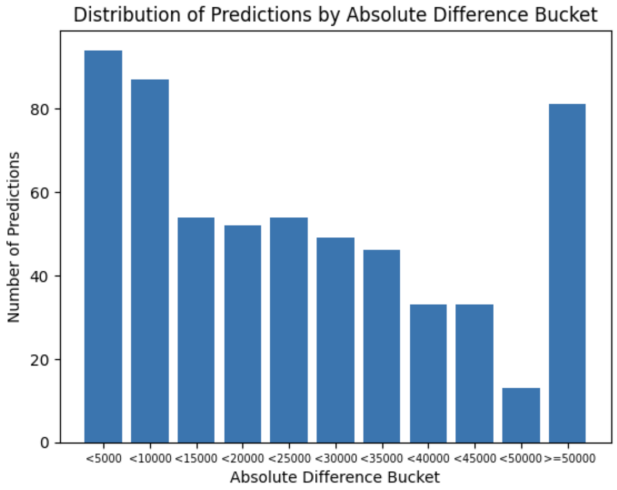
tuned during the experimentation earlier on. We split the data using 80% of the training set. We experimented with different configurations of a neural network by changing the number of hidden units in each of the hidden layers. We also experimented with a different number of epochs when fitting the training data, to see how that would affect our metrics.

Results

All of our results were analyzed using Mean Absolute Error (MAE). This metric is the measure of the average absolute differences between predicted and actual values in a set of observations

Linear Regression

The linear regression model performed the worst of the three models, which makes sense, as it is unlikely that lengthy text descriptions of jobs can be linearly separated. We tested three different types of linear models, regular Linear Regression, Lasso regression and Ridge regression. The regular Linear Regression model was incredibly overfit, giving an MAE of $27 on the training data and $37800 on the testing data, prior to any other hyperparameter tuning. Lasso and Ridge regression both helped to reduce overfitting, and had MAEs of $29,603 and $26,201 respectively. The further models were trained on Lasso regression. After all the experimentation we determined that 80% train/test split, with TF-IDF vectorization with a max_df of 0.25 was the most optimal configuration, providing an MAE of $25,264.

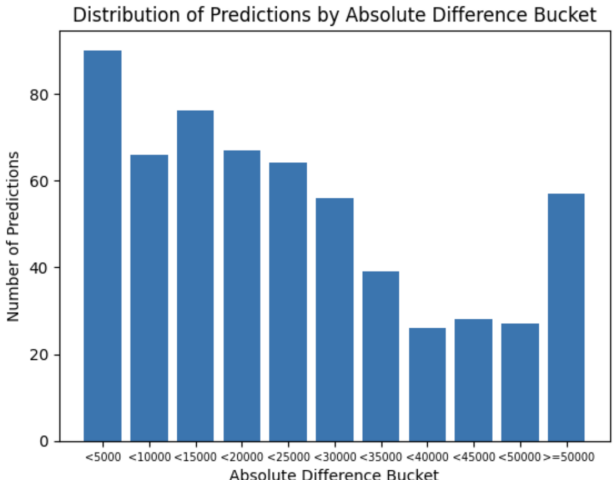


The graph above shows the number of data points in the testing dataset that fell within certain ranges in respect to their target salaries. We can see as the delta gets larger, the number of predictions in that bucket decreases, which is promising. But there are still significant amount of datapoints to the right of the graph, especially in the greater than $50,000 bucket.

Experiments were held using BoW featurization, but that MAE would only get as low as around $32,000, significantly higher than when using TF-IDF, so no further models were trained using Bow for linear regression.

Neural Network

After lots of experimentation we determined that the optimal configuration of our neural network was with four hidden layers, of sizes 128, 64, 128, and 32, and the relu activation function. We also determined that 10 epochs gave optimal results when fitting the data. When experimenting with more epochs, we noticed immense overfitting, where the training MAE was in the low thousands, while the testing MAE was still in the mid 20,000s. The optimal configuration resulted in an MAE of $23,890. This result is marginally better than that of linear regression, and significantly worse than that of support vector regression.



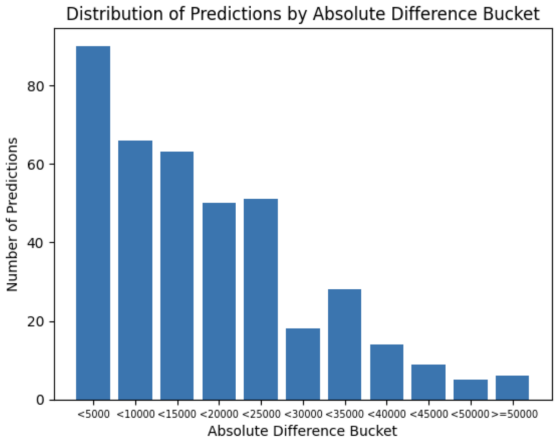
Referencing the graph above we can see that more predictions were within $5,000 of the actual salary than any other bucket. Between $10,000-$15,000 was the second highest. These are promising results, as we see a rapid decrease in the count of predictions as the delta increases.

Support Vector Regression

The Support Vector Regression models performed relatively well on the data, particularly the linear kernel trained on the Bag-of-Words. This model was very accurate, (for certain values of c) with 50%, 40%, and 20% of datapoints falling within $15,000, $10,000, and $5,000, respectively. The MAE for this model was the lowest of all modelsl. When training with a c-value of 100 the MAE was just under $16,000.

The linear-kernel TF-IDF model performed similarly to this model, but only had about 40% of predictions falling within $15,000 of the target value.

Across the models, increases in c-values tended to result in increased accuracy, but this was not always the case.



The graph above shows the results of training an SVR model on the optimal parameters determined above. About half of the predictions fell within $15,000 of the actual salaries, which was much better than both the neural network and linear regressions.

Future Work

Given more time there are several aspects of our models that could be worthy to change:

Different Methods of Featurization

BoW and TF-IDF feautirzation showed varying results with different models, so we might benefit from experimenting with more techniques, such as Skip-Gram based word embeddings, and pooling those embeddings together for reach job description. While this would increase the complexity and size of the input data, perhaps the embeddings would capture more meaning in each word in the job description than other forms of featurization.

Train different types of neural networks (RNN or BERT) that might better capture semantic meaning.

The models we trained for this project were largely regression-based. Investigating models that capture more semantic meaning in text, such as BERT.

Try direct feature engineering (location, industry, job title keyword, etc.)

Although we have discussed many models that seek to bring us away from feature engineering in NLP, it might make sense to have a salary prediction model that has at least some human-engineered features. We know as humans that salary varies (probably most significantly) along the dimensions of **location, industry, seniority, and job title**. It is easy to see how a model that only knows about job descriptions and not locations could easily mis-label a description from a metropolitan location if it was trained on a similar description from a more rural location. Having the model understand these dimensions for each job posting might give it a significant increase in performance.

Citations and Consultations

[1] Wikipedia Contributors. “Support-Vector Machine.” *Wikipedia*, Wikimedia Foundation, 20 Apr. 2019, en.wikipedia.org/wiki/Support_vector_machine.

[2] “Meaning of Epsilon in SVM Regression.” *Cross Validated*, stats.stackexchange.com/questions/259018/meaning-of-epsilon-in-svm-regression. Accessed 6 Dec. 2023.

[3] “Machine Learning - What Is the Influence of c in SVMs with Linear Kernel?” *Cross Validated*, stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-kernel.

[4] Sethi, Alakh. “Support Vector Regression in Machine Learning.” *Analytics Vidhya*, 27 Mar. 2020, www.analyticsvidhya.com/blog/2020/03/support-vector-regression-tutorial-for-machine-learning/.