Predicting Job Salaries from Text Descriptions

Shaun Jackman
sjackman@gmail.com
University of British Columbia

Graham Reid
graham.d.reid@gmail.com
University of British Columbia

Abstract

An online job listing web site has extensive data that is primarily unstructured text descriptions of the posted jobs. Many listings provide a salary, but as many as half do not. For those listings that do not provide a salary, it is useful to predict a salary based on the description of that job. We tested a variety of regression methods, including maximum-likelihood regression, lasso regression, artificial neural networks and random forests. We optimized the parameters of each of these methods, validated the performance of each model using cross validation and compared the performance of these methods on a withheld test data set.

1 Background

The data set is composed of 244,768 classified advertisements for jobs in the United Kingdom from the classified advertisement search web site, Adzuna. Each advertisement gives the title of the job, a plain text description of the job, the location, contract type (full-time or part-time), contract duration (permanent or contract), the name of the company, a category (such as customer service) and the annual salary. The majority of the data is found in the unstructured text description of the job. The challenge is to predict the salary of a job from its text description and other structured data. This data set and problem were obtained from a machine learning competition hosted by Kaggle [1], a web site dedicated to hosting machine learning competitions.

2 Methods

2.1 Data model

The data is composed of a job title and description, which is unstructured plain text, as well as additional structured data, the location, contract type and duration, company and category. The structured data were treated as categorical variables. The unstructured text features were modelled using a binary feature bag-of-words model.

The feature hashing trick [2] was used to limit the memory requirements to a fixed size. For each word in the description, the hash value of that word is calculated using MurmurHash3 [3] and the corresponding bit of the boolean feature vector is set to one. The size of the feature vector is thus fixed and does not depend on the variety of words present in the description. Some hash collisions are expected, but should be relatively rare with a sufficiently large feature vector. We used a feature vector of size 2^{24} bits, or 16 Mb.

We used a log-transform of the output variable, the salary. Without this transform, for a linear model each word of the description would contribute a fixed amount to the salary, such as £5,000 for the word "manager", or –£10,000 for the word "intern". This model does not make intuitive sense and could even result in predicting negative salaries. The effect of the log transform is that each word of the description instead contributes a multiplicative factor to the salary. For example, the word "manager" may cause the salary to be multiplied by 1.25, whereas the word "intern" may cause the salary to be multiplied by 0.50.

2.2 Comparison of regression methods

We trained and tested a variety of regression methods, maximum-likelihood regression, lasso regression, artificial neural networks, dropout neural networks and random forests. We used cross validation to optimize the parameters of the models. We then compared the performance of each optimized model by comparing the mean absolute error of each model's predictions on a withheld data set, which was not used for training the models. The software package Vowpal Wabbit 7.2.0 [4] was used for all of these regression methods except the random forest, for which we used the scikit-learn 0.13.1 RandomForestRegressor [5].

2.3 Maximum-likelihood regression

We used maximum-likelihood regression to predict salaries. We used a squared-error loss function and employed online stochastic gradient descent to minimize the cost function, $J(\boldsymbol{\theta})$ of equation 1.

$$J(\boldsymbol{\theta}) = (\boldsymbol{y} - \boldsymbol{X\theta})^T(\boldsymbol{y} - \boldsymbol{X\theta}) \tag{1}$$

2.4 Lasso regression

Since we have a very large number of features, namely the complete vocabulary of words used in the job descriptions, we used lasso regression to select informative features. We used a squared-error loss function with an L^1 regularizer and employed online stochastic gradient descent to minimize the cost function, $J(\boldsymbol{\theta})$ of equation 2. The optimal value for the regularization parameter, λ, was found using cross validation.

$$J(\boldsymbol{\theta}) = (\boldsymbol{y} - \boldsymbol{X\theta})^T(\boldsymbol{y} - \boldsymbol{X\theta}) + \lambda \left\|\boldsymbol{\theta}\right\|_1 \tag{2}$$

The learned feature weights of some common professions are shown in a word cloud [6] in Figure 1, where the size of the word is proportional to its weight.

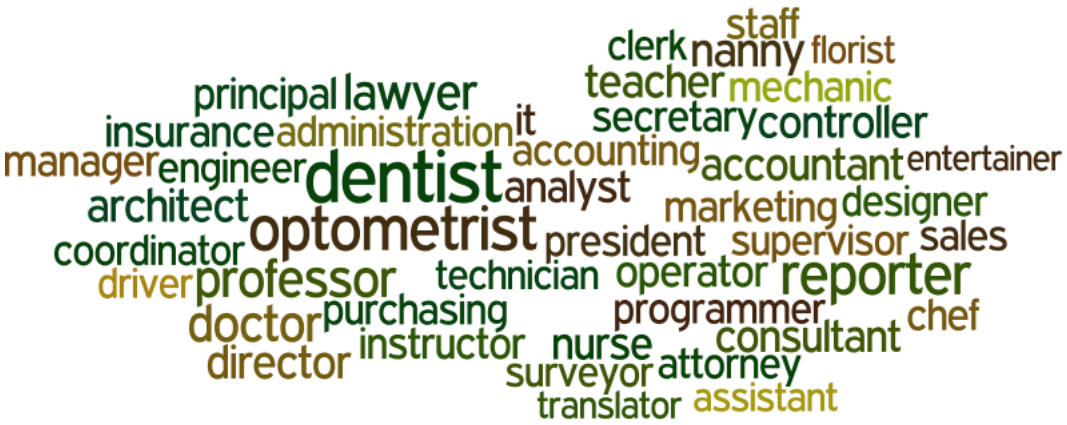

Figure 1: A word cloud showing the weights of common professions

2.5 Neural network regression

To test non-linear regression, we trained an artificial neural network with one hidden layer, a sigmoidal hidden-layer activation function and a linear output function. As with the previous regressors, we used a squared-error loss function, an L^1 regularizer and employed online stochastic gradient descent to minimize the cost function, $J(\boldsymbol{\theta})$. The variable $\boldsymbol{\alpha}_j$ of equation 3 is the vector of weights of the j^{th} hidden-layer neuron, and β is the weights of the output neuron.

$$J(\boldsymbol{\alpha}, \boldsymbol{\beta}) = (\boldsymbol{y} - \hat{\boldsymbol{y}})^T (\boldsymbol{y} - \hat{\boldsymbol{y}}) + \lambda \sum_j \|\boldsymbol{\alpha}_j\|_1 + \lambda \|\boldsymbol{\beta}\|_1 \tag{3}$$

$$\hat{y}_i = \boldsymbol{u}_i \cdot \boldsymbol{\beta}$$
$$u_{ij} = \tanh(\boldsymbol{X}_i \cdot \boldsymbol{\alpha}_j)$$

We alternated between optimizing the number of neurons and the regularization parameter, λ, using cross validation, until it converged. We note that this method may have converged on a local minimum, and it is possible that a better global minimum may exist.

2.6 Dropout neural network regression

We also trained a dropout neural network [8], which randomly omits each neuron with probability one half at each step of the stochastic gradient descent optimization. Randomly omitting neurons prevents overfitting the model by preventing neurons from co-adapting. The optimization of the number of neurons and regularization parameter was conducted as for the canonical neural network.

2.7 Random forest regression

We used random forest regression [9] to predict salaries. We used the scikit-learn [5] implementation, RandomForestRegressor, which does not support sparse feature vectors. For this reason, it was necessary to use a subset of features. We selected words from the lasso model that had the strongest weights. Additional informative words were selected by separating the jobs into five quantiles by salary and selecting words that gave the largest information gain (equation 4) [10], which is to say, those words that performed best at predicting the salary quintile of a job.

$$IG(T, a) = H(T) - H(T|a) = H(T) - \sum_{v=0}^{1} \frac{|\{x \in T | x_a = v\}|}{|T|} H(\{x \in T | x_a = v\}) \tag{4}$$

$$H(X) = E[I(X)] = E[-\ln(P(X))] = \sum_{i=1}^{5} P(x_i) \log P(x_i) = -\sum_{i=1}^{5} \frac{n_i}{N} \log \frac{n_i}{N}$$

The depth of each tree was unlimited and stopped when only two elements remained in each leaf node. Each tree was trained on all the data, but on a random subset of features. The number of features used in each tree is the square root of the number of available features.

3 Results

3.1 Comparison of regression methods

We optimized the hyperparameters of each model using cross validation, where 70% of the data was used to train the model, and 15% of the data was withheld from training and used to validate the performance of the trained model. A final 15% of the data was used to test the generalized performance of the optimized models. The mean absolute error of each model is shown in Table 1.

Table 1: Mean absolute error (MAE) of various regression methods

Method	Training (£)	Validation (£)	Test (£)
Maximum likelihood	3404	7195	6376
Lasso	4124	6391	5945
Neural network	3912	6284	5868
Dropout NN	3947	6312	5876
Random forest	N/A[†]	N/A[†]	5000

[†]The random forest was run by my co-author, and the training and validation errors were not recorded.

3.2 Lasso regression

The L^1 regularization parameter λ of the lasso regressor was optimized using cross validation, the result of which is shown in Figure 2. The optimal value was found to be $\lambda = 1.4 \cdot 10^{-7}$. Of the 199,614 distinct words seen in the job descriptions, 88,087 had non-zero weights.

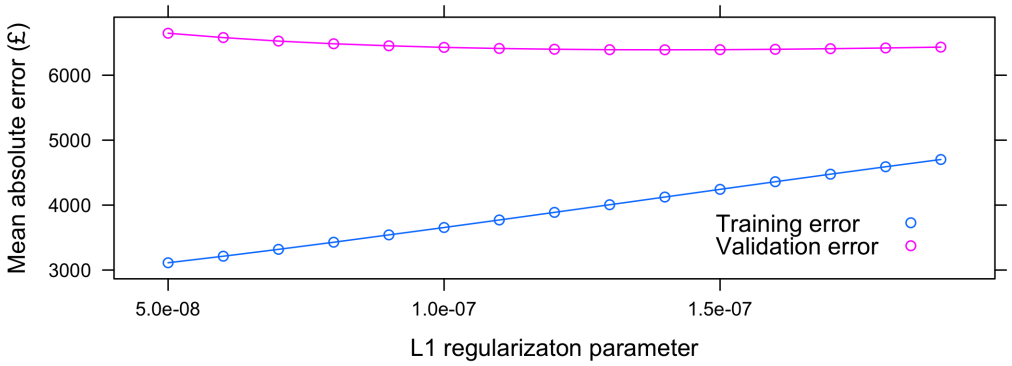

Figure 2: Optimizing λ, the L^1 regularization parameter of the lasso regressor

3.3 Neural network regression

The optimal number of neurons in the hidden layer of the artificial neural network was found using cross validation, the result of which is shown in Figure 3. The optimal number of neurons was found to be two. That the optimal number of neurons is so small is a surprising result. Also unexpected is that the training error increases with an increasing number of neurons. We would expect the training to continue to decrease with an increasing number of neurons, as the increasing complexity of the model becomes overfit to the training data.

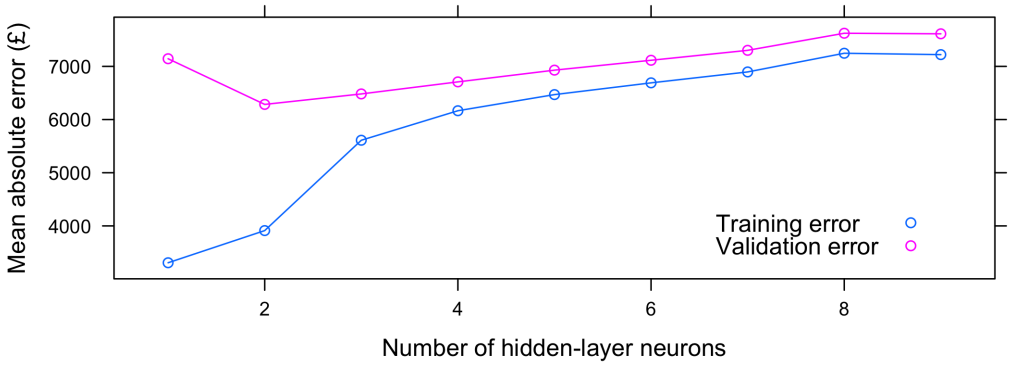

Figure 3: Optimizing the number of hidden-layer neurons of the neural network ($\lambda = 1.2 \cdot 10^{-7}$)

The L^1 regularization parameter λ of the artificial neural network was optimized using cross validation, the result of which is shown in Figure 4. The optimal value was found to be $\lambda = 1.2 \cdot 10^{-7}$.

3.4 Dropout neural network regression

We similarly optimized the number of neurons of a dropout neural network and the L^1 regularization parameter λ using cross validation and found the optimal number of neurons to be 26, as shown in Figure 5, and the optimal L^1 regularization parameter to be $\lambda = 1.4 \cdot 10^{-9}$.

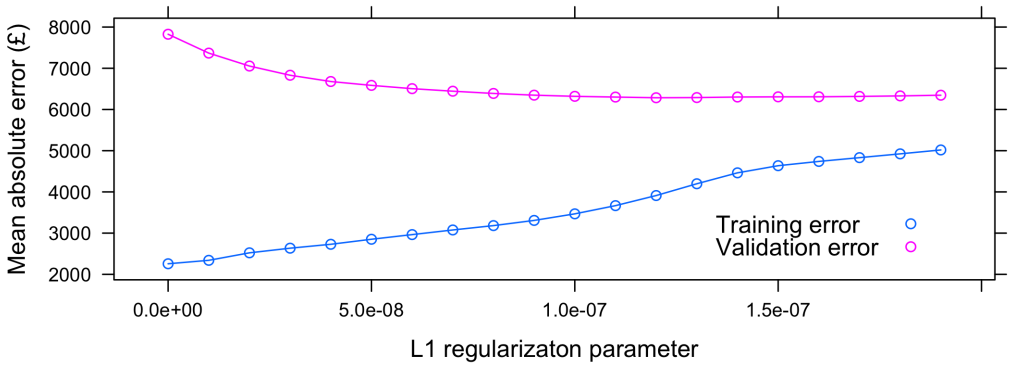

Figure 4: Optimizing λ, the L^1 regularization parameter of the neural network with 2 hidden-layer neurons

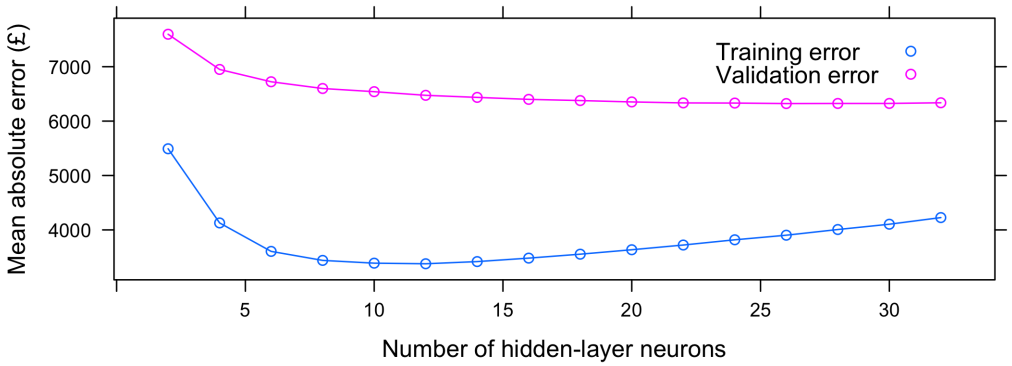

Figure 5: Optimizing the number of hidden-layer neurons of the dropout neural network ($\lambda = 1.4 \cdot 10^{-9}$)

Randomly omitting neurons while training effectively randomly trains 2^n different models, where n is the number of neurons. This ensemble of neural networks acts as a form of regularization, and so it is not surprising to see that the optimal value of the L^1 regularization parameter λ is 100-fold smaller for the dropout neural network than the canonical neural network.

3.5 Random forest regression

To optimize the random forest, we tried limiting the depth of the tree and increasing the number of trees and found that fewer trees with unlimited depth performed better. Our final random forest regressor was composed of 50 trees. The number of trees was limited by available computing power, and using more trees should improve performance.

We used 1,000 features and varied the split between the number of features selected by the largest absolute lasso weights and by the largest information gain in distinguishing the salary quintiles. We found that a split of 800 features selected by lasso and 200 features selected by information gain performed best. We also tested 2,000 features and found, somewhat unexpectedly, that 1,000 features performed better. Since each tree is trained on a random subset of features, it becomes important that each feature be informative.

4 Conclusions

The random forest outperformed the neural network, which outperformed the dropout neural network, which outperformed the lasso regression, which outperformed the maximum-likelihood regression. The only surprise here is that the dropout neural network did not perform better than the canonical artificial neural network. A dropout neural network is able to use more neurons than a canonical neural network while still avoiding overfitting, and because it is effectively an ensemble method, it should reduce error due to variance, just as a random forest does.

We are able to predict the salary of a job using a textual description of that job to within a mean absolute error of £5,000. A human tasked with the same problem would not, we expect, perform any better at predicting salaries. We find these results quite satisfactory. That being said, there is almost certainly room for improvement.

The bag-of-words model used is the simplest possible feature space. Other natural language processing techniques could help, such as removing stop words and using longer strings of words as features such as bigrams and trigrams. Another appealing option would be to use simple syntatic analysis to extract noun phrases such as "heavy machinery technician" from the description.

An additional machine learning technique to consider would be k-nearest-neighbours adapted for regression, where the prediction is a weighted average of the salaries of the k most similar job descriptions, and closer neighbours are weighted more heavily than distant neighbours.

We expect that the data would have underlying structure. For example, a number of job postings may be summarized as "A senior programming job in London", and we expect these jobs to have similar salaries. It would be valuable to explore machine learning techniques that attempt to capitalize on this underlying structure, such as a deep-learning neural network, or latent Dirichlet allocation [11], which assumes that the distribution of words for each job depends on a latent category of that job, such as "service industry", and attempts to learn those categories from the data.

References

[1] Kaggle Inc. (2013). Job Salary Prediction.
http://www.kaggle.com/c/job-salary-prediction

[2] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola and Josh Attenberg (2009). Feature Hashing for Large Scale Multitask Learning. Proc. ICML.
http://alex.smola.org/papers/2009/Weinbergeretal09.pdf

[3] Austin Appleby (2011). MurmurHash.
https://code.google.com/p/smhasher

[4] John Langford (2007). Vowpal Wabbit – a fast online learning algorithm.
https://github.com/JohnLangford/vowpal_wabbit

[5] Pedregosa, Fabian, et al. (2011). Scikit-learn: Machine learning in Python. The Journal of Machine Learning Research, 12, 2825-2830.
http://scikit-learn.org

[6] Jonathan Feinberg (2013). Wordle – Beautiful Word Clouds.
http://www.wordle.net

[7] Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society, Series B 58 (1): 267–288.
http://www.jstor.org/stable/10.2307/2346178

[8] Hinton, Geoffrey E., et al. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580.
http://arxiv.org/abs/1207.0580

[9] Breiman, L. (2001). Random forests. Machine learning, 45(1), 5-32.
http://link.springer.com/article/10.1023/A:1010933404324

[10] Wikipedia authors (2013). Information gain in decision trees.
http://en.wikipedia.org/wiki/Information_gain_in_decision_trees

[11] Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. The Journal of machine Learning research, 3, 993–1022.
http://dl.acm.org/citation.cfm?id=944937