Job Salary Prediction

Anonymous Author(s)
Affiliation
Address
email

Abstract

Kaggle is an online platform for machine learning and analytics competitions where companies post data and individuals post solutions to compete for monetary prizes and to hone their skills [1]. Recently Adzuna, a UK based classifieds company, hosted a Kaggle competition with the goal of improving job salary predictions for job postings in which the salary does not appear. Using various linear models, random forests and neural network implementations we were able to achieve an mean error of £4933 corresponding to the top 6% of the of the competitors. This paper describes the details of our feature selection and modeling methods.

1 Introduction

Often when job advertisements are posted online, the employer neglects to mention the salary. To an individual looking for a job, this poses a dilemma; do they risk wasting valuable time investigating a low paying job, or skip the advertisement and risk ignoring a great opportunity.

Adzuna, a UK based classifieds company hosts many job advertisements; approximately half of which do not list a salary. In order to provide better services to its user base, Adzuna wants to give accurate salary estimates for job postings for which the true salary is not provided. To this end, Adzuna hosted a Kaggle competition with the goal of improving job salary predictions [1].

This paper explores the performance of four different regression techniques applied to the Adzuna data. Section 5 deals with linear models while subsequent sections explore random forest regressors, gradient boosted tree regression and neural networks. With the exception of the neural networks which were implemented with Vowpal Wabbit [2], the remaining models were implemented with standard classes from the 0.13.1 scikit-learn Python distribution [3].

Unfortunately, we did not finish our work in time for the competition. However, our best models are very competitive placing us in the top 6% of the competition when run on the withheld testing data.

2 Data Structure

The training data for the competition consists of a .csv file of approximately 245 000 job postings with the information in each job posting arranged in 12 fields. These fields included free text title, description and location descriptors along with additional categorical fields such as contract type (permenant or contract), contract time (part-time or full-time), company, etc.

The goal of the competition is then to predict the salary given the non-salary fields. The loss function specified by the competition is the mean absolute error on a withheld testing set.

3 Benchmarks

Adzuna provided two benchmarks against which the efficacy of user submitted models could be tested [1]. The first of these simply output the mean salary of the training data. The second of benchmark was based on a random forest implementation. In this implementation the title, description, and location fields were vectorized, resulting in a vector consisting of the number of occurrences of the hundred most common unigrams (single words) in each field. This vector representation of the job postings was the used to train a scikit-learn random forest regressor. The benchmark performances are summarized below in Table 1.

Model	**Error**
Mean Benchmark	£13253
Random Forest Benchmark	£7633

Table 1: Adzuna benchmark performance

4 Feature Extraction

Due to the limited scope of this project, we decided to use bag-of-words models rather than more advanced feature extraction techniques. For the linear and forest models investigated in the later sections, the scikit-learn count vectorizer[1] implementation was used to extract unigram and bigram (collections of two words) features from the title, description and location fields while the remaining training fields were treated as categorical variables. To avoid memory issues and excessive training times, any features with relative frequency of less than 0.01% were ignored.

The implementation of the neural networks of section 8 was performed by my partner who took a slightly different approach to feature extraction. Rather than using a count vectorizer, my partner used a hashing vectorizer to extract features.[2] Although this form of feature extraction is much faster, it sometimes encounters problems when two or more features hash to the same index. This effect can be mitigated however, by choosing a hash function with a sufficiently large range.

For ease of notation, training sets consisting of categorical and unigram features will be referred to simply as unigram features. Likewise, training sets consisting of categorical, unigram and bigram features will be referred to as bigram features.

5 Linear Models

Kaggle user Foxtrot demonstrated that linear models could exceed the performance of the random forest benchmark provided by Adzuna, achieving an error of £6734 [4]. We reasoned that features with large weights in linear models would make ideal features for our random forest implementation.

As our primary reason for investigating linear models was to find a small number of highly predictive features, we opted to use lasso regression rather than ridge regression in fitting the models. As shown in Equations 1 and 2, the loss function of lasso regression differs from ridge regression by the replacement of the $\mathbf{L^2}$ regularizer with an $\mathbf{L^1}$ regularizer. With ridge regression, the coefficients $\boldsymbol{\theta}$ go slowly to 0 as the regularization coefficient is increased. With lasso regression, on the other hand, the regularization parameter serves to set the weights of weakly correlated features identically to zero. Additionally, if two features are highly correlated with one another, lasso regression will select one of them and set the coefficient of the other to zero. As such, lasso regression is ideally suited as a method of feature selection; with an appropriate choice of the regularization parameter we will be able to select predictive, near orthogonal features.

[1]The scikit-learn class CountVectorizer [3] constructs a dictionary of unique tokens from the training corpus. Each time a token is encountered in the fitting procedure, the value corresponding to that token in the document's feature vector is incremented. Preliminary investigation showed that the linear models gave the best predictions when a feature was simply given as present or absent as opposed to the total number of occurrences. For this reason, the results of the subsequent sections are based on binary feature vectors.

[2]Instead of using a dictionary to record unique tokens, a hashing vectorizer uses a hash function to map features directly onto indices.

$$J_{ridge}(\mathbf{y}) = (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^T (\mathbf{X}\boldsymbol{\theta} - \mathbf{y}) + \alpha\boldsymbol{\theta}^T\boldsymbol{\theta} \tag{1}$$

$$J_{lasso}(\mathbf{y}) = (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^T (\mathbf{X}\boldsymbol{\theta} - \mathbf{y}) + \alpha \left\|\boldsymbol{\theta}\right\|_1 \tag{2}$$

5.1 Results

Figure 1 summarizes the performance of our linear models for various choices of the regularization parameter, while Table 2 lists the performance of the two best linear models on the test set. Note that we were able to exceed the performance of the benchmark as well as Foxtrot's model.

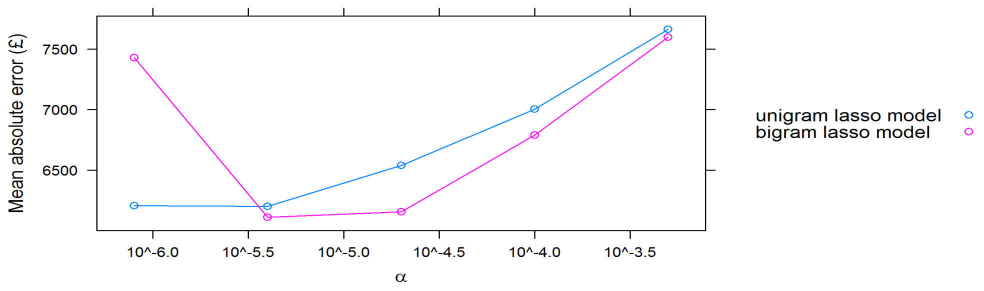

Figure 1: Performance of the linear models on the withheld test set for a range of regularization parameters.

Model	Error
Unigram Lasso, $\alpha = 4 \cdot 10^{-6}$	£6201
Bigram Lasso, $\alpha = 4 \cdot 10^{-6}$	£6112

Table 2: Linear model performance

6 Random Forests

6.1 Feature Selection

The features used in the random forest benchmark are obviously less than ideal; if one simply selects the most common unigrams from each field, the resulting feature vector will be dominated with words such as "and", "the" and "it". These words will appear in nearly every job posting and their ability to distinguish between job postings will be limited.

In an effort to avoid this issue, two different techniques were used to select relevant features from the corpus. As shown in the previous section, properly optimized linear models were able to beat the random forest benchmark by sound margins. By using the features with the largest weights in the linear models, it should be possible to duplicate this success with random forest models.

Although linear models are able to select features with strong correlation to salary, they are unable to select descriptive features with net zero correlation (perhaps due to nonlinearities) or distinguish between features which have very high correlation to salary but which occur infrequently throughout the corpus. Taking the deficiencies of the linear model selection into account leads us to consider information gain (Equation 3) as a method of determining the relevance of features.

$$G(f) = \sum_{i=1}^{M} P_r(c_i) \log P_r(c_i) + P_r(f) \sum_{i=1}^{M} P_r(c_i|f) \log P_r(c_i|f)$$
$$+ P_r(\bar{f}) \sum_{i=1}^{M} P_r(c_i|\bar{f}) \log P_r(c_i|\bar{f}) \quad (3)$$

By dividing the training data into n groups based on salary, it is possible to use information gain as defined in Equation 3 [5] to select features, f, which discriminate between different salary ranges, c_i, (while having near zero overall correlation). For the purposes of this project, the advertisements were divided into five equal groups corresponding to jobs with salaries in the 0^{th} to 20^{th} percentile, 20^{th} to 40^{th} percentile, etc.

6.2 Optimization

The random forests used here are slightly unusual in that each forest considers only the square root of the number of features at each node and that the forests are grown until no additional information can be extracted from the leaves (i.e. the maximum depth parameter is set to some very large number). The first of these conditions may be understood as a method of simultaneously increasing variance between trees while reducing the training time of the regressor.

The second condition essentially enforces the idea that when dealing with documents it is advantageous to consider a wide variety of features before making a decision. If the maximum depth were set to some small number n, each tree would consider the presence or absence of only a small number of features before regressing the document. Figure 2 demonstrates that when choosing between models of equal training complexity it is advantageous to favor tree depth over the number of trees in the forest.

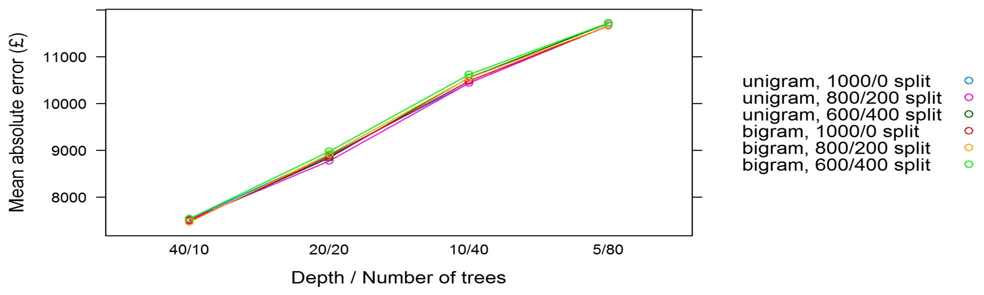

Figure 2: Random forest performance at constant training complexity. All models were trained on a 20% / 80% split of the training data for 10 trees with the linear features selected from the corresponding linear model with $\alpha = 10^{-4}$. Had the models been trained on an 80% /20% split, the training time would have been prohibitively large. Note that the errors given here are given for the 80% validation data, not the withheld testing set. We found that there was very little difference in performance between forests with a maximum depth of 40 and unlimited depth forests.

To find the optimal set of features, many small (10 tree) random forests were trained on 20% of the data using various combinations of features selected from the linear models of the previous section and the information gain metric defined above. The results of this search are shown below in Figure 3 for models with a total of 1000 features. Note that the best results are not obtained by selecting features from the best linear models, but rather from models with a much smaller number of non-zero weights (larger regularization parameters).

Surprisingly, the best model results from selecting 1000 unigram features with an 800/200 split of linear model features to information gain features. This result is rather counterintuitive as you might expect the bigram models to be better at discriminating between documents. Examining the density of the training matrix sheds some light on the situation. As shown in Table 3, the training data of

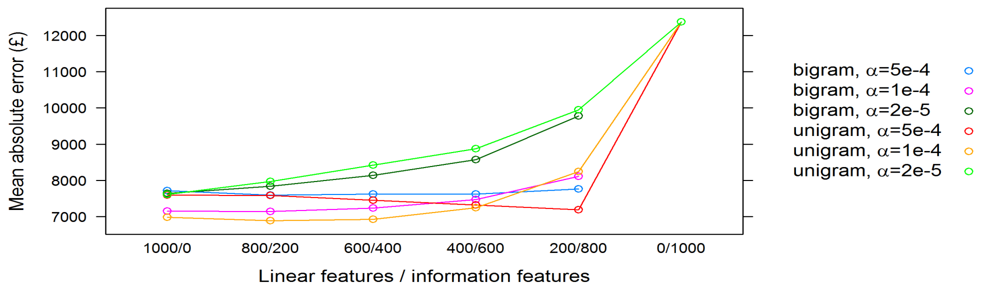

Figure 3: Random forest error at constant training complexity with 1000 features. All models were trained on a 20% / 80% split of the training data for 10 trees with the linear features selected from the linear model indicated in the legend. The x-axis denotes the split between features selected from the linear models and features selected by the information gain criteria.

the best unigram models is denser than that of the corresponding bigram models. This indicates that we are selecting bigram features that discriminate very well between income levels but that are not present in the majority of the samples. As such, at least for small numbers of trees, we find that our feature selection process works best for selecting the more common unigram features.

Model	**Density**
Unigram RFR, 800/200, $\alpha = 1 \cdot 10^{-4}$	0.0191
Unigram RFR, 800/1200, $\alpha = 1 \cdot 10^{-4}$	0.0113
Bigram RFR, 800/200, $\alpha = 1 \cdot 10^{-4}$	0.0137
Bigram RFR, 800/1200 $\alpha = 1 \cdot 10^{-4}$	0.0073

Table 3: Density of random forest features

6.3 Results

Using the results of the previous section, it can be seen that the best models used linear models with the weight of the $\mathbf{L^1}$ regularizer set to $\alpha = 1 \cdot 10^{-4}$. Rebuilding these models with 50 trees and training on the entire data set gives the results summarized in Table 4. Note that the best of these models surpasses the random forest benchmark by over £2600. All of the models in Table 4 would have placed in the top 10% of the competition.

Model	**Error**
Unigram RFR, 800/200, $\alpha = 1 \cdot 10^{-4}$	£5000
Unigram RFR, 800/1200, $\alpha = 1 \cdot 10^{-4}$	£5003
Bigram RFR, 800/200, $\alpha = 1 \cdot 10^{-4}$	£5138
Bigram RFR, 800/1200, $\alpha = 1 \cdot 10^{-4}$	£5094

Table 4: Performance of the four best random forest models

7 Stochastic Gradient Boosting

Stochastic gradient boosting [6] is a powerful technique for greedily training regression models. Developed in 1999 by Jerome Friedman [7], gradient boosting works by greedily building a function which minimizes some specified loss function. Specifically, let $L(\mathbf{f})$ be our loss function where we treat $\mathbf{f}$ as a function of the samples, $\mathbf{f} = (f(\mathbf{x_1}), ..., f(\mathbf{x_N}))$. We then solve for $\mathbf{f}$ stepwise using gradient descent. At the m^{th} iteration we find $\mathbf{g}_m$, the gradient of $L(\mathbf{f})$:

$$g_{im} = \left[\frac{\partial L(y_i, f(\mathbf{x_i}))}{\partial f(\mathbf{x_i})} \right]_{f(\mathbf{x_i}) = f_{m-1}(\mathbf{x_i})} \tag{4}$$

The estimate of $\mathbf{f}$ at the m^{th} iteration is then:

$$\mathbf{f}_m = \mathbf{f}_{m-1} - \lambda \mathbf{g}_m \tag{5}$$

Where λ is known as the learning rate and must be chosen appropriately for optimal convergence. This can be made into a useful algorithm by fitting a weak learner, such as a decision tree, to the residuals to approximate the gradient function [6, 7].

In the case of stochastic gradient boosting regression, the algorithm above is simply modified so that the residual is computed for some sub-sampling of the total data. According to Friedman, this serves to both act as a form of regularization which prevents overfitting and reduce the computation time [7]. As such, all of the base learners for the models discussed in the subsequent sections were trained on random samplings of 50% of the data with a least squares loss function.

7.1 Optimization

Unfortunately we did not have sufficient time or computational resources to fully optimize the gradient boosting implementation (In total this project consumed over 1000 processor hours on personal computers). Rather than optimize the split between information gain and linear features, the number of features per split, the maximum tree depth and the learning rate, we took a simpler though certainly less optimal approach.

Firstly, because the unigram random forests consistently outperformed the bigram random forests, the gradient boosted tree regressors (GBRs) were trained only on unigram data. Since the algorthim uses decision trees as its base estimators (like the random forests previously discussed) we made the simplifying assumption that the 800/200 feature split of the previous section would be an appropriate split. Next, we noted that the training time of the model is approximately linear in both the maximum tree depth and number of features to consider per split (fps). We then tested four different configurations of constant product between maximum tree depth and number of features for a range of the learning rate parameter λ. These models were trained with a total of 20 trees on 20% of the training data. The results can be seen in Figure 4.

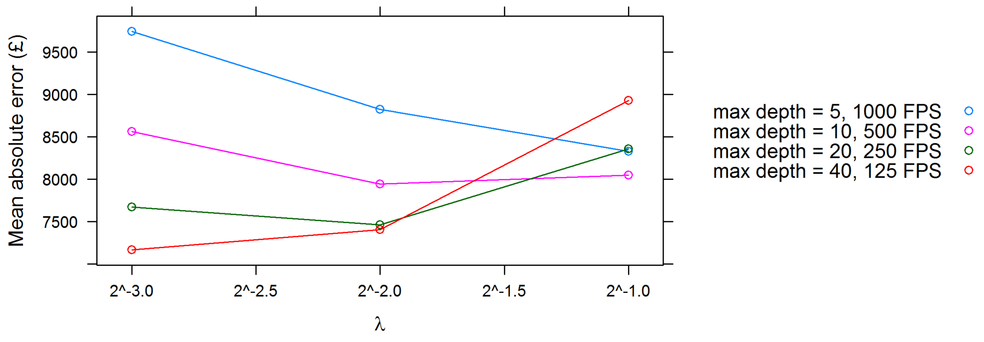

Figure 4: GBR performance at constant training complexity. All models were trained on a 80% / 20% split of the training data for 20 trees with the linear features selected from the best unigram linear model of section 5 with the learning rate parameter λ as given on the x-axis.

7.2 Results

From Figure 4 the fastest converging model was the model with a maximum tree depth of 40 and which considered 125 features per split (note that this is very similar to the results of the previous section). This model was retrained on the entire training set with 100 trees for a range of λ. The results are summarized below in Table 5.

Model	Error
Unigram GBR, $\lambda = 1.25 \cdot 10^{-1}$	£5362
Unigram GBR, $\lambda = 6.25 \cdot 10^{-2}$	£5326
Unigram GBR, $\lambda = 3.13 \cdot 10^{-2}$	£5640

Table 5: Performance of the three best gradient boosted models

8 Neural Networks

Like random forests, neural networks can be effective non-linear regressors. We considered a neural network consisting of an input layer, a single non-linear ($\tanh$) hidden layer and a linear output layer with an $\mathbf{L^1}$ regularizer. The network was implemented with Vowpal Wabbit [2] and was trained on unigram data vectorized by a hashing vectorizer as discussed in section 4.

8.1 Results

To achieve optimal performance, we performed an axis-aligned grid search (alternating between optimizing the regularization parameter and the number of neurons in the hidden layer), until convergence was achieved. Below, Figure 5 plots the performance of the neural network as a function of the number of neurons in the hidden layer at the optimal value of the regularization parameter α. Additionally, the performance of the best neural network is given in Table 6

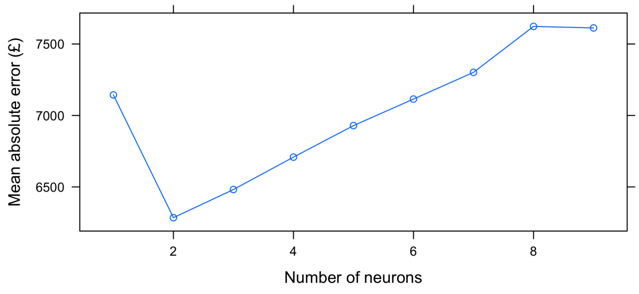

Figure 5: Performance of the neural networks as a function of the number of neurons in the hidden layer at the optimal value of the regularization parameter. All models were trained on a 80% / 20% split of the training data.

Model	Error
Unigram Neural Network, $\alpha = 1.2 \cdot 10^{-7}$	£5817

Table 6: Performance of the best neural network model

9 Conclusions

Although we did not achieve the best results in the competition (the top performing method scored an average error of £3465 [1]), many of our methods scored in the top 10% of the competition with our best method corresponding to the 6th percentile. These results are summarized in Table 7 which lists the performance of the best model for each regression technique.

Most importantly, we demonstrated that linear models and information gain can be used to extract predictive features from text documents. Models based on these features consistently outperformed the random forest benchmark provided by Adzuna. Additionally, we showed that while the inclusion of bigram features improved the linear models, it did not necessarily improve the predictive capabilities of our random forests. With our current feature selection mechanism, the bigram features were simply too sparse to impart any advantage over the unigram features.

Model	**Error**
Bigram Lasso, $\alpha = 4 \cdot 10^{-6}$	£6112
Unigram RFR, 800/200, $\alpha = 1 \cdot 10^{-4}$	£5000
Unigram GBR, 800/200, $\lambda = 6.25 \cdot 10^{-2}$	£5326
Unigram Neural Network, $\alpha = 1.2 \cdot 10^{-7}$	£5817
Aggregate RFR	£4933

Table 7: Performance of the best models for each regression technique. The "Aggregate RFR" model is a linear combination of the four best random forest models of section 6. This model was able to outperform all of our other models and achieved a ranking of 16th out of 294 competitors.

Due to time and computation constraints, the GBR models were not fully optimized. Additional work could concentrate on further improving these models, perhaps using gaussian process optimization to explore the high dimensional parameter space.

Additionally we would like to explore the possibility of further optimizing our feature extraction and selection techniques. In future work we would like to include features based on document statistics (document length, average number of words, Flesch-Kincaid readability statistics, etc) as well as features based on similarities between documents (perhaps using latent dirichlet allocation, an unsupervised learning technique which assumes that the distribution of words in a document is indicative of underlying categories [8]).

References

[1] Kaggle. *Job Salary Prediction*. `https://www.kaggle.com/c/job-salary-prediction`.

[2] John Langford. *Vowpal Wabbit (Fast Learning)*. `http://hunch.net/~vw/`. 2013.

[3] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python ". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[4] Foxtrot. *Predicting advertised salaries*. `http://fastml.com/predicting-advertised-salaries/`. 2013.

[5] Yiming Yang and Jan O Pedersen. "A comparative study on feature selection in text categorization". In: *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*. MORGAN KAUFMANN PUBLISHERS, INC. 1997, pp. 412–420.

[6] Kevin Murphy. "Machine learning: a probabilistic perspective". In: MIT Press, 2012. Chap. 16, pp. 554–563.

[7] Jerome H Friedman. "Stochastic gradient boosting". In: *Computational Statistics & Data Analysis* 38.4 (2002), pp. 367–378.

[8] David M Blei, Andrew Y Ng, and Michael I Jordan. "Latent dirichlet allocation". In: *the Journal of machine Learning research* 3 (2003), pp. 993–1022.