# Salary Engine – A Job Recommender System and Salary Predictor

Lin Huang
Dept. of Electrical Engineering
Columbia University
lh2647@columbia.edu

Fan Ye
Dept. of Statistics
Columbia University
fy2188@columbia.edu

Mingrui Xu
Dept. of Electrical Engineering
Columbia University
mx2151@columbia.edu

Wei Cao
Dept. of Electrical Engineering
Columbia University
wc2467@columbia.edu

*Abstract*— **With the drop of the recent years' US unemployment rate, the job market has become more and more active. All members of our group are graduating students who are current finding jobs and we all notice the importance of a great job recommendation and information data analysis system for jobseekers and employers. So, our project, Salary Engine, is aimed to build a platform that can help both jobseekers and employers to share and receive job information. Also, based on the large dataset of job positions, we implemented the salary prediction system and the job recommendation system as well as a website to make the interaction between users and our system models. After several versions of update on our algorithms, the prediction and recommendation results have been very accurate.**

***Keywords- Salary; Job; Prediction; Recommendation; Data Analysis; Web***

## I. INTRODUCTION

Several years ago, due to the depression of US finance, the high US unemployment rate became a big concern in society. Many jobseekers, especially graduating college students, felt hard to find a job or could not find a job matching their backgrounds. Also, at that time, due to the lack of online career services and the Big Data concepts were not as popular as it is nowadays, jobseekers could not find an efficient way to search for the job information and they were also not able to have a sense of how the job market was with limited sources and data.

That time was also difficult for employers who wanted to recruit talented candidates. Because of the financial situation, many employers wanted to manage their budget to find talented people to lead the team. However, first, there were few platforms for them to post the job position information to public and second, they were not sure what kind of salary range could be both reasonable and competitive for that position. As a result, some of these companies went bankrupted.

Nowadays, with drop of the unemployment rate and the rapid development of Internet and Information Technology, much more career service websites such as Glassdoor.com, Indeed.com and Monster.com have grown to be the excellent platforms for jobseekers and employers. With the increasing popularity of Big Data analysis, a lot of job data are open sourced and there are much more statistics are on these data. Inspired by prosperity of job market and Big Data analysis concept, our project, Salary Engine, is a tool and website which helps employers to determine more reasonable salary range for a position with specific information and helps jobseekers to find more jobs matching their backgrounds.

Our project could be divided into three parts. The first is the salary prediction system. We used several classification and regression algorithms to build and test the prediction model. Then we compared the prediction results of different algorithm methods and picked Random Forest as the most accurate algorithm to make the prediction. The second is the job recommendation system, which uses the Item-based and User-based concepts to make the recommendations based on the similarity between the job positions and jobseekers' backgrounds. The third part is the website and the database where users can make the query and get the detailed information on job positions. Users can also see the introduction of the companies and make their comments on the information page of the company.

## II. RELATED WORKS

Many research teams had worked on the job recommender system design but most of them just focused on the theory and architecture[1][2]. In our project, the job recommendation system is a practical tool that uses Mahout as the analysis back-end and it is more efficient and user-friendly[3].

Also, Glassdoor.com is a great example for us when we construct our own website. Based on the instruction and tutorial of web development, we finished the interactions between frontend and backend and communication with our database.

## III. SYSTEM OVERVIEW

We planned to build a web that is very similar to a job-hunting web, focusing on recommendation and prediction. In this section, we will elaborate the dataset used to build the database, as well as the architecture of this web.

### *A. Dataset*

We take advantage of the public research dataset of Adzuna, which is a classified ads metasearch or vertical search

engine based in United Kingdom. The raw dataset includes more than 200 thousands posted job information, and for each row it has 12 fields, including description, location, contact type, salary etc.

To user the bulk load function provided by some database, we preprocessed the dataset, eliminated some useless field, converted the display format and generated a common .csv file.

*B. Architecture*

Our project architecture has been shown as Figure 1. The web is based on AWS (Amazon Web Service) platform. For the data side, we created some table like Users, Records, Comments using DynamoDB, and S3 is applied to store some picture like company's logo. Three cloud-based servers were created to connect data side and front end, and provide web service. The apache server aims to do basic queries for PHP, the tomcat server helps to call mahout to perform user-based and item-based recommendation; the django server enables the communication between front-end and python, which mainly does modeling and give the prediction result.

To make our web more meaningful, users could register, make preview and the recruiter can also post new job info. Thus, our database will update everyday, which requires the everyday modeling to decide more accurate parameters. The details of algorithm analysis will be discussed in the following several section.

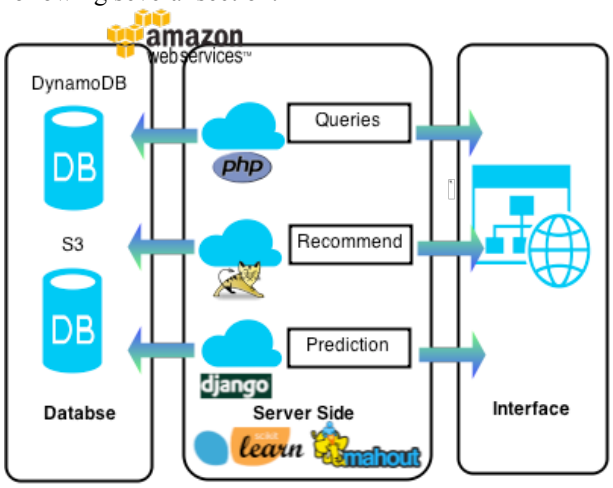



**Figure 1. Web Architecture**

## IV. ALGORITHM

A. Recommendation

We provide two kinds of recommendation to the job candidate; the first is trying to recommend the job that matches candidate's background, and another is based on the feeling given for certain companies.

We try to match the background from candidate's job category desired, location, expected minimal salary and contract type, etc. The similar result could be denoted as formula (1),

$$S = \alpha_1 \cdot loc + \alpha_2 \cdot sal + \alpha_3 \cdot con + \alpha_4 \cdot cat \quad (1)$$

The value for location, contract type and category are simply determined by the string matching, however the value for salary was calculated in a more complex way shown as formula (2),

$$sal = \begin{cases} 1, dis \in [0,5000] \\ 1 - 0.1 \cdot (dis - 5000)/5000, dis \in [5000,10000] \\ 0.9 - 0.2 \cdot (dis - 10000)/5000, dis \in [10000,15000] \\ 0.7 - 0.3 \cdot (dis - 15000)/5000, dis \in [15000,20000] \\ 0.3, others \end{cases} \quad (2)$$

where dis denotes the difference between your expected and real salary. For the condition dis<0, we adjust the coefficient less than above. During recommendation, we will choose the ten most similar jobs to the candidate.

For the company recommendation, we took advantage of mahout package in j2EE. We first took data from database, where for each row contains three fields as "userId", "companyId", and "rank". The rank should be computed like formula (3),

$$rank = \alpha_1 \cdot feeling + \alpha_2 \cdot sentiment(comments) \quad (3)$$

We are using amazon SQS to calculate the comments sentiment, however due to time limit, we have not finished that yet. At this moment, we only have the first part of rank. After finishing that, we generate the .csv file, and use Euclidean Distance to compute the similar, based on which we could get user and item-based recommendation.

B. Prediction

One of the most important goals of salary engine is to help companies set reasonable salaries for different positions they provide. Therefore, when giving certain information from the companies, we need to establish some models to give good prediction results of the salaries depending on the locations, job descriptions, categories, etc. In our project, we have used five different algorithms for modeling, which are regression tree, random forest, support vector machine and kth nearest neighbors.

*B.1 Text Mining*

Since there are several features in the dataset expressed by text, such as "Title", "FullDescription"…, the first thing we need to do is to quantize them because only numeral information can be dealt for modeling and through training process. The method we used is that convert a collection of text documents to a matrix of token counts. Each row is normalized to have unit Euclidean norm. The weights of each feature are computed by the fit method call. For simplification, here is an example shown below.

```
>>> measurements = [
...     {'city': 'Dubai', 'salary': 33000.},
...     {'city': 'London', 'temperature': 12000.},
...     {'city': 'San Fransisco', 'temperature': 18000.},
      ]
...
```

In the above, "city" is a text attribute while "salary" is a traditional numerical feature. After quantization, we change it into the following form by making a word dictionary.

```
array([[ 1.,  0.,  0., 33.],
       [ 0.,  1.,  0., 12.],
       [ 0.,  0.,  1., 18.]])
...
```

For a text composed by several words (strings), we need to further normalize it and use the fit model to calculate the weights. Here we use the function served by the tool called sk-learn[4]. A simple example is like the following. The first term is present 100% of the time hence not very interesting. The two other features are only in less than 50% of the time hence probably more representative of the content of the documents.

```
>>> counts = [[3, 0, 1],
...           [2, 0, 0],
...           [3, 0, 0],
...           [4, 0, 0],
...           [3, 2, 0],
...           [3, 0, 2]]
```

After fitting and normalization, the result is as follows:

```
array([[ 0.85...,  0.  ...,  0.52...],
       [ 1.  ...,  0.  ...,  0.  ...],
       [ 1.  ...,  0.  ...,  0.  ...],
       [ 1.  ...,  0.  ...,  0.  ...],
       [ 0.55...,  0.83...,  0.  ...],
       [ 0.63...,  0.  ...,  0.77...]])
```

By changing text into numbers successfully, we can do the next work properly.

### *B.2 Regression Tree*

The first algorithm coming to our mind is regression tree. Because tree-based methods involve stratifying or segmenting the predictor space into a number of simple regions, it's flexible and suitable for our salary data. Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as decision tree methods, in which the target variable can take continuous values (salaries) are called regression trees[5]. In the modeling process, we need to split the tree by selecting "best feature", which is the different of information gain from the two branches of the training data as in formula (4). We need to maximize the information gain by minimizing conditional entropy as formula (5).

$$I(Y, X_i) = H(Y) - H(Y \mid X_i) \tag{4}$$

$$X_i = \arg\min_{X_i} H(Y \mid X_i) \tag{5}$$

In our project, we use the function provided by scikit learn tool package to realize the training process and apply the regression criteria to minimize the Mean Squared Error (MSE) as formula (6). For the node m, we represent a region $R_m$ with $N_m$ observations.

$$H(X_m) = \frac{1}{N_m} \sum_{i \in N_m} (y_i - c_m)^2 \tag{6}$$

All the observations and features quantization will be discussed in the random forest algorithm.

### *B.3 Random Forest*

As the result of regression tree is not that accurate, we considered the random forest algorithm for further analysis. Random forests are an ensemble learning method for regression that operate by constructing a multitude of decision trees at training time and outputting the value that is the value output by individual trees[6]. It has many advantages than merely regression tree. It can scale well to large datasets ($O(10^7)$observations, thousands of covariates). It can estimate variable (features) importance automatically. And also overfitting has less effect on it. For our dataset, since it is a bit large (almost 2GB) with thousands of features generated from text mining, we need to prune some of the unimportant feature to handle the modeling process (CPU limitation). So we only choose 100 most important features for training. For random forest, the training algorithm applies the technique of random "features bagging" to tree learners as formula (7).

$$\hat{f} = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b(x') \tag{7}$$

The mean square error of test data is 7608, which is smaller than regression tree.

### *B.4 Support Vector Machine*

SVM is another algorithm that can regularize parameters to help avoid over-fitting. Meanwhile, with different Kernels, we can get different results with high flexibility[7]. The training result is more robust as well. The SVM loss function is shown as formula (8).

$$\arg\min_{\mathbf{w},b} \max_{\boldsymbol{\alpha} \geq 0} \left\{ \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i [y_i (\mathbf{w} \cdot \mathbf{x_i} - b) - 1] \right\} \tag{8}$$

We need to choose parameter to test the performance of different Kernels. The kernel we used in our datasets are linear, polynomial and RBF.

### *B.5 Linear Regression*

As one of the most common algorithms for data analysis, linear regression is very simple to be handled and can be used for prediction in our project. At the beginning, we come to use it as a benchmark for our test results. The training process is to minimize the RSS as formula (9).

$$\arg\min \sum_{i=1}^{n} y_i^2 - 2\mathbf{w}^\top \mathbf{X}^\top \mathbf{Y} + \mathbf{w}^\top \mathbf{X}^\top \mathbf{X} \mathbf{w} \tag{9}$$

When we were training the data, we found that unlike random forest or SVM, we could use full features we got

from text mining (1,000+) to establish the model, which made something different. The model itself is not that bad comparing with other complex algorithms. Results and details are shown in the next part.

*B.6 K-Nearest Neighbors*

KNN is the last algorithm we tested. Unlike all the above algorithms, it is a non-parametric method used for regression. We just take k closest training examples in the feature space, the output is the salary value for the company. It is something like random forest, both of which can be viewed as so-called weighted neighborhoods schemes. The results of kNN also shows this conclusion[8]. More details will be shown in experimental results.

## V. SOFTWARE PACKAGE DESCRIPTION

Our software package can be roughly divided into three parts: database and recommendation, prediction and front-end, which are written by Java, Python, PHP respectively.

*A. Database Management and Recommendation*

Since, we need to use AWS's database, this part is based on a J2EE version Eclipse installed with Amazon toolkit. There is a class called RdsLoader, which contains some basic query such like create or drop a table, select or delete one record. This class provides the way for you to check and do changes on database.

Another import part written in Java is the recommendation. There are two types of recommendation here. The first is trying to calculate the similarity between one job and user's background. Another is based on mahout to perform user-based and item-based recommendation. To realize to communication between front-end and server side, two Java servlet were written, which need you to specify current user's ID.

*B. Data Modeling and Prediction*

In this part, we used Python to build the model. Firstly, we need to preprocess the data.

What we got is a large dataset. The main dataset consists of a large number of rows representing individual job ads, and a series of fields about each job ad. These fields are as follows:
Id - A unique identifier for each job ad
Title - A free-text field supplied to us by the job advertiser as the Title of the job ad. Normally this is a summary of the job title or role.
FullDescription - The full text of the job ad as provided by the job advertiser. Where you see ***s, we have stripped values from the description in order to ensure that no salary information appears within the descriptions. There may be some collateral damage here where we have also removed other numerics.

LocationRaw - The freetext location as provided by the job advertiser.
LocationNormalized - normalised location based on the raw location.
ContractType - full_time or part_time, specific additional field we received from the advertiser.
ContractTime - permanent or contract.
Company - the name of the employer as supplied to us by the job advertiser.
Category - which of 30 standard job categories this ad fits into, inferred in a very messy way based on the source the ad came from. We know there is a lot of noise and error in this field.
SalaryRaw - the freetext salary field we received in the job advert from the advertiser.
SalaryNormalised - the annualised salary. Note that this is always a single value based on the midpoint of any range found in the raw salary. This is the value we are trying to predict.
SourceName - the name of the website or advertiser from whom we received the job advert.

All of the data is real, live data used in job ads so is clearly subject to lots of real world noise. Thus our first challenge about building model is preprocess the data. We used the singular value decomposition method to make the data smooth, and then discard the missing values.

And secondly, by using sklearn.base Python package, we did the text mining to extract the information from the variables showed as text, like full job description variable.

Also we used the scikit-learn package which is a package for machine learning in Python. By using this package, we did all the model building, like random forest, support vector machine, and k nearest neighbors.

Finally, we output the results, and calculated the mean square error for different methods, compared the results.

*C. PHP Based Interface*

For more efficiency, we chose PHP instead of Java servlet. PHP was written together with some front-end language like HTML and JavaScript. The names of all pages are ended with .php. And the style of our front-end is actually Bootstrap.

There are four basic pages in out interface. "index.php" is our front page, which provides the entries for functions such like log in, register, and keyword search. "search.php" is pages with search results listed on it. "res.php" displays a company in more details including its basic information and

some statistics data. “pos.php” displays the complete information for a certain job. Besides, the list of the recommended company and job are shown in the page “recommend.php” and “rec_com.php” respectively.

*D. User Interface Description*

Our interface is shown as Figure 2. There are three kinds of user rights, visitor, candidate and recruiter. In order to become a candidate or recruiter, you may need to take advantage of our register form and provide some info of your background. The permission for each role of user has been shown in Table 1.

| | Visitor | Candidate | Recruiter |
|---|---|---|---|
| View info | √ | × | √ |
| Search | √ | × | √ |
| Comments | × | √ | √ |
| Rank | × | √ | √ |
| Recommendation | × | √ | × |
| Post Job | × | × | √ |
| Prediction | × | × | √ |

**Table 1. User Permission**

(1) Search: We provide two kinds of search, for the simple search, you can input the keyword searching by position, salary or location. Using advance search, you could specify these info in one time.

(2) Comments: You can give the preview about a certain country. Meanwhile, you need to give your feeling, which will automatically converted into an integer noted as rank for it. This rank would be the very important basis for recommendation. Also, if you show a good feeling, then your comments will be marked as “recommend”, otherwise “not recommend”. Figure 3 has shown an example of this comments function.

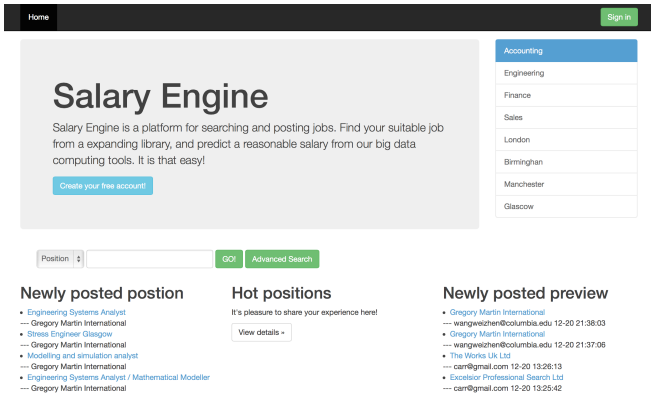



**Figure 2. Interface of Salary Engine**

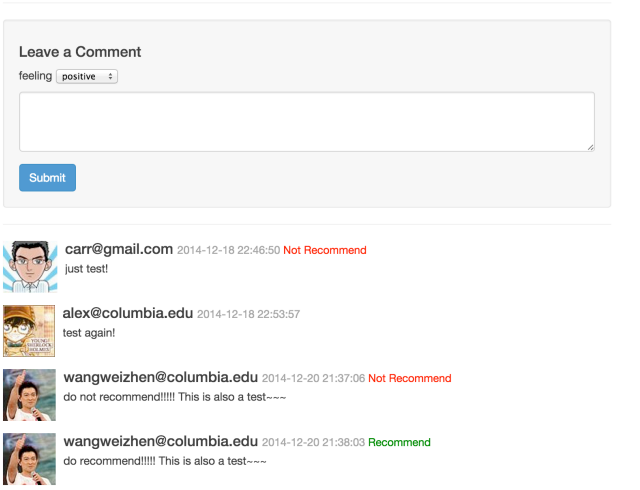



**Figure 3. Comments function**

(3) Recommendation.
This is only for candidate, as mentioned above the two kinds of recommendation provide ideal job and company respectively. Just need to click on the “Recommend” and “Company” button on the left top of the page, you could see the results like Figure 4, and Figure 5.

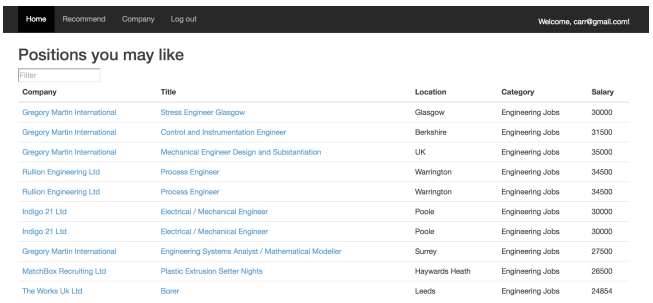



**Figure 4. Job Recommendation**


**Figure 5. Company Recommendation**

(4) Prediction
This is only for company recruiter, before posting a new job you could use this function to get a rough standard. Just need to click on “Predict” button on the left top of the page.

## VI. EXPERIMENT RESULTS

A. Recommendation

The recommendation results have already been shown in Figure 4 and Figure, however due to lack of users' dataset we cannot provide some algorithm analysis here. The thing need to be mentioned is that we adjust the coefficients in section IV based on our team member's experience during using this web's recommendation function.

## B. Prediction

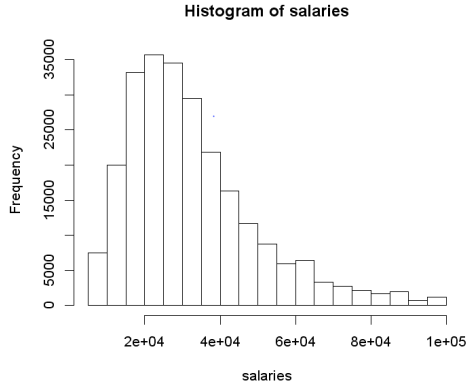
### *B.1 Linear Regression*

Linear regression model seeks to find estimates of our predictors' parameters so that sum of squared errors is minimized.

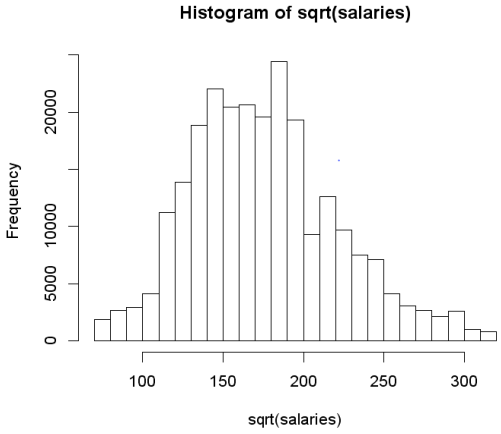
As Dang Ha The Hien points out in the comments, linear regression actually assumes the errors (residuals) are normally distributed. The transformation of the dependent variable might help achieve normality of the residuals:

*I believe that for dependent variables which are restricted to be positive, e.g. salary, interest, rate of return…, we usually need some kind of transformation (usually log) because the variance of errors will increase when the value of dependent variable increases. For example in your ads salary problem, if your model predicts an ads's salary is 100k$, the 20k$ error is more likely to happen than when you predict an ads's salary is 30k$. Generally, in these cases, the standard deviations of the residual distributions increase when the dependent variables increase.*

So being a linear learner, probably expects the target variable residuals to have a normal distribution. We checked empirically Therefore, it is important to log transform salaries, because the original distribution is skewed:








The result of linear model is not bad, especially in terms of MSE. In this method, we used the job title, the information we extracted from the full job description, normalized location, job type, contract time, company, category, and the response variable is normalized salary. We got the mean square error of the model with these features is 4863.75.

### *B.2 Regression Tree*

Basic regression trees partition the data into smaller groups that are more homogenous with respect to the response. To achieve outcome homogeneity, regression trees determine:

- The predictor to split on and value of the split
- The depth or complexity of the tree
- The prediction equation in the terminal nodes

While trees are highly interpretable and easy to compute, they do have some noteworthy disadvantages. First, single regression trees are more likely to have sub-optimal predictive performance compared to other modeling approaches. An additional disadvantage is that an individual tree tends to be unstable.

In this case, we use regression tree to predict the salary for each position and also we use 10 folds cross validation to resampling.

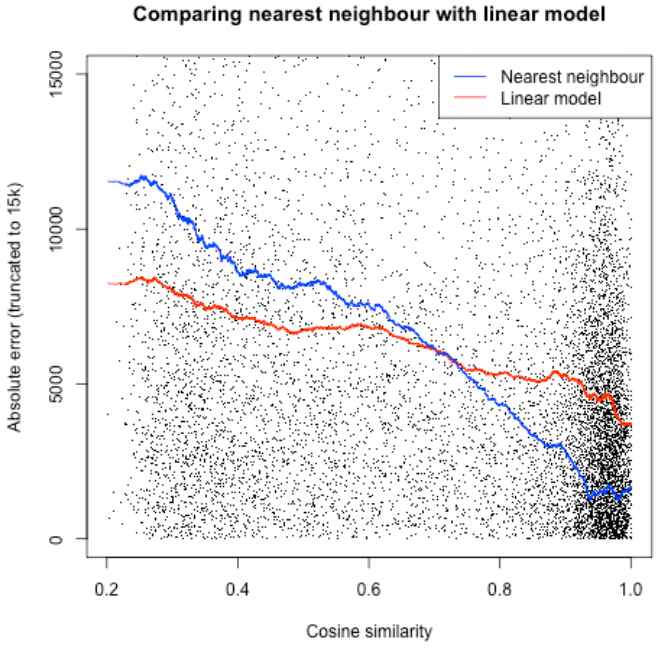
The results of Regression Tree are not very good. The mean square error of the model is 9653.21.

*B.3 K Nearest Neighbors*

The KNN approach simply predicts a new sample using the K-closest samples from the training set. To predict a new sample for regression, KNN identifies that sample's KNNs in the predictor space. KNN works well if the decision boundary is non-linear and flexible.

In this particular case, we help the company to predict the salary for each position. Upon pre-processing the data and selecting the distance metric, the next step is to find the optimal number of neighbors. Like tuning parameters from other models, K can be determined by cross validation. For our model, 20 values of K ranging between 1 and 20 were evaluated. As illustrated in plot 10, a K of approximately 20 yields the lowest MSE (Using 10-fold Cross-Validation).

And in this method, finally we got the MSE is 8215.2.




*B.4 Random Forest*

In particular, trees that are grown very deep tend to learn highly irregular patterns: they overfit their training sets, because they have low bias, but very high variance. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance. This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance of the final model.

Thus we also used random forest method after the regression tree method.

In this method, we tried 50 trees each time:

```
features = feature_extractor()
steps = [("extract_features", features),
         ("classify", RandomForestRegressor(n_estimators=50,
                                            verbose=2,
                                            n_jobs=1,
                                            min_samples_split=30,
                                            random_state=3465343))]
```

The results of our random forest are 7608.13.

*B.5 Support Vector Machine*

Support Vector Machine can be applied not only to classification problems but also to the case of regression. Still it contains all the main features that characterize maximum margin algorithm: a non-linear function is leaned by linear learning machine mapping into high dimensional kernel induced feature space. The capacity of the system is controlled by parameters that do not depend on the dimensionality of feature space.

In this case, we also tried support vector machine with kernels. Since we are not sure which kernel method should be used, thus we tried linear kernel, polynomial kernel, RBF kernel.

Finally we used the linear kernel and got the MSE 8741.32.

## CONCLUSION

After all the implementation, construction and testing, whole system of Salary Engine runs well and all the recommendation and prediction results are acceptable and reasonable.

So far, we mainly use the given database for the processing. But in the future, with the increase of users of Salary Engine, we will make web database to be dynamic so that more newly-post job info from our users could be loaded in as part of the dataset. Moreover, we want to make Salary Engine to be not only a career service website, but also a social platform for jobseekers and employers to be friends to have a better communication experience.

This project was conducted by Lin Huang, Fan Ye, Mingrui Xu and Wei Cao. We came up with ideas and composed final report together. Lin Huang and Wei Cao mainly focused on the construction of the website and database. Fan Ye and Mingrui Xu focused on the algorithm models design, implementation and experiment. All authors contributed about equally to the work.

## REFERENCES


[1] Yao Lu, Sandy El Helou and Denis Gillet, "A Recommender System for Job Seeking and Recruiting Website".

[2] R Lee, ER Wilbur, "Age, education, job tenure, salary, job characteristics, and job satisfaction: A multivariate analysis", Human Relations, 1985

[3] Ronald C Taylor, "An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics", BMC bioinformatics, 2010

[4] F Pedregosa, G Varoquaux, A Gramfort, "Scikit-learn: Machine Learning in Python", The Journal of Machine, 2011

[5] X Su, M Wang, J Fan, "Maximum likelihood regression trees", Journal of Computational and Graphical, 2004

[6] A Liaw, M Wiener, "Classification and Regression by randomForest", R news, 2002

[7] Z Xuegong, "Introduction to statistical learning theory and support vector machines", Acta Automatica Sinica, 2000

[8] Wikipedia, "k-nearest neighbors algorithm", http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm