# Mining Frequent Patterns and Knowledge Discovery from the Job Market Data

Peter Berta and Viktor Matovic

Slovak University of Technology in Bratislava, Bratislava 812 43, SK
Faculty of Informatics and Information Technologies Ilkovicova 2, 842 16 Bratislava 4

**Abstract.** Even though workforce is constantly growing, employers still need to lure the best applicants to work for them. Apart from advertisement and references, job applications are the most compelling ways of attracting candidates. While expected salary plays a big role in convincing possible applicants, it is difficult to set it appropriately. In this paper, we focus on predicting potential salary based on data of salaries in different positions in various fields and companies.

**Keywords:** data mining, salary prediction, job application, regression

## 1 Problem Analysis and Motivation

There is a great amount of highly educated people in the world, and the number is not decreasing. More and more people come to job market and look for the best opportunity for them. While job offers with high salary seem to be irresistible, applicants must consider other factors while picking up their upcoming employment. Great numbers on the paycheck do not always secure high interest in the position. A lot of aspects need to be considered. Work location, hiring company or expected work time. When some of the working conditions isn't the most suitable, the salary need to be appealing. Otherwise, no applicants will arrive for an interview. Employees therefore need to set the salary high enough to make people interested, but also low enough to not waste all of company resources. Without previous experience, setting this number can be challenging.

In this paper, we focus on predicting salary of job offers based on their description. As previously stated, employees need to set expected salary on the job applications appropriately. Even though final payroll can be individually discussed with an applicant, a rough estimation needs to be done. While company resources are more than often limited, employees unceasingly demand higher payment. These requests cannot be always approved. Thereupon compromise need to be made. Correctly determined salary can lure enough applicants and not ruin company's economics.

## 2 Data on job advertisements and its characteristics

Examined dataset is made up of job advertisements, with by employers have searched job seekers in a UK job market. These data samples are primarily organized by the numeric, nominal, numeric (ratio-scaled) and discrete attributes as stated in Table 1. Because we work with incomplete real-world data (e.g. attribute Contract Time within the aggregated advertisements is not always provided), we have a need to preprocess them. Furthermore, dataset we work with is not up to date, employer advertisements could be already removed or changed. Specified job positions could be already occupied. We can say data are affected by a timeliness. Data are characteristic by the high interpretability, for the most part because of raw text is at least 50 % of the data tuples content.

| Attribute | Type | Statistical description/description |
|---|---|---|
| ID | Numeric | None |
| Salary Normalized (year) | numeric/ratio-scaled | mean=34216.73, median=33033.64, midrange=74863.81, min=9492.956, max=140234.7 |
| Job Title | nominal/discrete | raw text / not normalized |
| Source Name | Nominal | raw text / not normalized |
| Category | nominal/discrete | raw text / not normalized |
| Company Name | Nominal | raw text / not normalized |
| Contract Type | nominal/discrete | raw text / not normalized |
| Contract Time | numeric/ratio-scaled | raw text / not normalized (e.g. permanent), 40 663 entries sometimes with missing values. Necessity preprocessing. |
| Location Normalized (extracted from raw text) | nominal/discrete | raw text / normalized |

Table 1: Data characteristics

In order to work with the data itself, they must be loaded and should be in appropriate format for the library functions to understand them. As we want to use all the columns from the required dataset and we want to minimize time to run the scripts, we pass an optional argument n = 200 (as 200 rows) to the following code during experimentation:

```
txt <- readLines(
  sprintf("%s/final/%s/%s.%s.csv",
    data.raw.dir, locale, locale, src),
  n=200, skipNul = T, encoding="UTF-8")
saveRDS(txt, sprintf("%s/%s.rds", data.raw.dir, src))
```

Reader should take an attention to bypassing the need of passing an exact filename. Function *saveRDS* will save RDS file in the data/raw directory, with which we will later manipulate as with single R object.

## 3 Related Work

Chunmian Ge et al. have created an extensive analysis on factor that influence starting salary of IT graduates [2]. Even though their paper was published in 2015, they also worked with paper as old as 20 years, which improves their objectivity. While Chunmian Ge et al. do not work with data mining methods, their area of interest is close to ours. Utilizing certain results of their work, we might improve our model at later stage. One of the main results of their work is correlation table, which directly shows valuable attributes of individuals. However, some of these attributes are irrelevant to our work, since we do not posses that kind of information. For example, gender or employee's foreigner status are unreachable to us.

Apart from job salary prediction, there are papers concerning data mining social media articles or other textual posts. These two papers specifically use 'tweets' from social network Twitter[1] to determine occurrences of influenza[2] [3,4]. In addition to tweets, Aron Culotta in his paper also takes into consideration public information about the person who posted the tweet [3]. Some of this information includes gender, age, location, which might improve predict spread of the illness. In his paper, he considers two regression models, Simple Linear Regression and Multiple Linear Regression. Even though great number of improvements to the method are suggested, he concludes that Multiple Linear Regression outperforms Simple Linear Regression.

Todd Bodnar et al. also used mainly keywords to identify possible influenza characteristics in public tweets [4]. However, they used much greater number of tweets and experimented with other regression models. At first, they used Linear Regression and Multivariable Regression. After modifying their model to prevent overfitting by selecting best keywords, they considered SVM Regression (Support Vector Machine). They concluded, that all their regression models performed comparatively well. However, SVM Regression model is intuitively better in some instances, Todd Bodnar et al. explains why it does not mean that it is heavily superior [4].

While Logistic Regression and Naïve Bayes present reliable methods widely used in data analysis, Ming-wei Chang et al. propose combined method called Partitioned Logistic Regression (PLR) [5]. According to their results, PLR outperforms Logistic Regression and Naïve Bayes, when using the same training examples. Ming-wei Chang et al. explain PLR as a set of Logistic Regression models combined using Naïve Bayes principles. This combined regression model is tested and evaluated on several different datasets, to prove its effectiveness. The task at hand consists of spam filtering.

[1] Twitter web page: https://twitter.com/
[2] More information about influenza disease: https://www.rivm.nl/en/Topics/I/Influenza

# 4 Feature Engineering

## 4.1 Sampling Data

To work with some specific functions we will later use, user can separate the functions file in a separate place and load it simply by a *source()* function passing it a path to the file. We will show how we store the data, it's more helpful for us to store them as a training object, devtesting object and testing object. All named objects store their data which we will be later manipulate with. We remind the reader that we will use a slightly changed terminology for these objects, training, devtesting and testing objects represent the folders we call *branches*.

Training branch contains 5 % of 200 loaded rows from a dataset. Knowing that we can use higher percentage, we later realize, that this higher percentage will lead to consuming more memory and time because of loading them more often. Reader should take an attention to be in a right folder, we assume that he/she has a separate folder to work with and which will be structured in a way our functions understand. We create the three called branches as shown in a next code excerpt:

```
#if we use more than one dataset we want to split, it's more than
#recommended to have some naming conventions. Next code will walk
#through the files named version.first, version.second and version.third
#assuming some variable definitions previously
#data.raw.dir stands for the name of our directory
for (pick in c('first', 'second', 'third')) {
toTrain <- readRDS(sprintf("%s/%s.rds", data.raw.dir, pick))
lines_of_Train <- length(toTrain)
trainSampleSize<-assign the calculated training branch size (by a rule)
sampledIds <- assign ID attributes, but the way will be changed soon#and
#finaly store a branch, pick arg is optional if having one file
saveRDS(chosenLines, sprintf("%s/%s.train.rds", data.raw.dir, pick))
```

Same approach is applied to the second and third branch (devtesting and testing). If we would call the training branch a corpus, we can state that corpus file is next to the above code splitted to data samples each to the separate branch.

## 4.2 Preprocessing

Preprocessing is a process, we made up this process from the smaller tasks. These tasks do the normalization, remove non-ASCII characters, introduce end of sentence marks and replace undesired elements from a text. Because we will use only one training branch, we do not need to walk through the *for* cycle and we just pass the code, that will store the desired variables in a list. These variables stand for a corpus filename,

size of the branch and as a signal for skipping a stemming process. Function *clean* is responsible for a cleaning a training branch. This function stands on the top of other useful methods required and called from them to do a specific work. After this work is finished, sub-functions return their results back to the *clean* function. These functions are:

- *clean*: which is a function that takes a training branch as an argument, loads the compressed file if is not already loaded, but does a more important task too. It does by calling a function *cleanCorpus* which splits the training branch into smaller parts we call *data chunks*. These data chunks are next to the splitting process cleaned and undesired text elements are replaced. After running all the steps, it will save its result in a RDS format by the use of *saveRDS()* function. File will be stored at a *data clean directory*. Feature engineering principles are shown in a function code, that use specific variables and pointers.

- *cleanCorpus*: is a function that is called with a promise to split a training branch and clean their data chunks later. Because she works with the passed variables, stemming process is prohibited yet. This function creates parts of each corpus is called with, we call *chunks* by itself, assigning them ID attributes from a matrix that store the function called *initIds*. Function *cleanCorpus* requires the massive work of a function called *cleanText*, which except the massive cleaning process is responsible for a stemming process too.

- *cleanText*: is a function which loads and use stemmed text lines in a data chunks for its own later used mechanisms. Because we do not want to replace undesired symbols in a loaded rows (such as smileys), we remove them by the massive use of *gsub()* function so often called in *cleanText()*. If user wants to use these functions without the need of use regular expressions, we recommend passing an optional argument *fixed*.

```
gsub("\\s([A-Z])\\.\\s", "\\1", x, ignore.case = FALSE,
perl = FALSE, fixed = FALSE, useBytes = FALSE)
```

### 4.3 Full Description Preprocessing

Since the data we work on mostly contain textual data, we need to preprocess them slightly. The first attribute we adjust to our needs is *FullDescription*. This attribute mostly consists of several sentences, which altogether explain the job offer itself. Preprocessing sequence on *FullDescription* follows:

1. Remove punctuation
2. Remove numbers
3. Remove stop words
4. Remove unnecessary spaces

For example, consider sample description sentence (the stars are part of the sentence):

```
Registered Nurse Jobs Edinburgh required urgently H****
Healthcare have Registered Nurse assignments throughout
Edinburgh and Lothian. You should have a minimum of ****
year post registration experience and be enthusiastic and
flexible in your approach to work.
```

After the preprocessing process, this sentence would get edited to sentence:

```
registered nurse jobs edinburgh required urgently h
healthcare registered nurse assignments throughout edin-
burgh lothian minimum of year post registration experi-
ence enthusiastic flexible approach work
```

### 4.4 Creating N-grams

After cleaning the training branch in a data chunks, we merge them back and use them as a one branch/corpus. This corpus gets back assigned ID attributes distributed by a use of *initIds* function. As previously stated this function does it by a use of previously called ID-attributes matrix. As was the *clean* function on the top of the calls in a cleaning process, purpose of the function *nGramization* is very similar. This function calls the three massive processes and get back results from them. These processes are managed in a way, each process belongs to one method. Next processes are called for a purpose of creating unigrams, bigrams and trigrams:

- tokenizer() – this function gets an argument to know, which type of n-grams we want to make. Then loads a function from a RWeka package and returns it to the caller. We will later use this to create a vector containing tokenized words from a loaded lines.
- buildNGramChunks() – this function gets the training branch and splits it as a help to the *nGramization* function. We split the branch for a purpose of speeding up tokenizing the words, we watch the running time as well. Building up the data chunks for a later tokenizing is a two-part process. For a purpose of merging the chunks back to the previous state, there are *mergeChunks* function.
- nGramizeChunk – stands above another function called *splitNGrams*. These functions are required to do the splitting process, then tokenizing a training branch will cost less resources.
- splitNGrams – function is a helper function for a *nGramizeChunk*. It does so by splitting up already splitted. This function is responsible for a final tokenizing.
- mergeChunks – helper function that is called after all of above to merge data chunks back to the previous state (to a unite training branch).

.

# 5 Data Mining Methods

To determine the most optimal regression method to predict job salary, we considered recommendations by Vlado Boža from his presentation[3]. The base of his method suggests creating n-grams, specifically bigrams, and using Linear Regression on top of that data.

## 5.1 Linear Regression on top of n-grams

Linear regression is one of the basic data analysis methods, which is widely used even today. While result of classification is identifying the most suitable class for piece of data, regression is estimating numeral values from set of real numbers. The key aspect of this estimation includes identification of the most suitable attributes, which correlate with result value at desirable rate.
To differentiate between existing variables and value which is to be estimated, naming conventions are used. Attributes which contribute to the estimation are called independent variables. The result value of regression is called dependent variable. Note, that various literatures can use different naming conventions.
Based on the number of independent variables, we acknowledge two major types of linear regression:

1. Simple Linear Regression – 1 independent variable
2. Multiple Linear Regression – >2 independent variables

In both cases, there is only one dependent variable. In case of Simple Linear Regression, model function usually consists of $x$ and $y$ coordinates as follows:

$$y = \alpha + \beta x$$

The goal is to find α and β so that the linear function would depict the best fit for majority of data in dataset. Utilizing this function, one can predict values of opposite axis, when only one value is accessible.

Considering the nature of linear regression, independent variables need to be in numerical form. This does not correspond to our data, which is mostly textual. In order to convert textual attributes to numerical values, one can extract certain keywords into attributes and assign them values 0 or 1, depending on their presence in the data. Textual attributes which include more than one word can be similarly extracted in form on n-grams, specifically bigrams. N-grams are defined as consequent expressions from a sequence of expressions. Bigrams therefore consist of two words, which are consequent in the attribute. The goal is to find the most influencing bigrams out of the whole dataset.

[3] Presentation link:http://www.mlmu.sk/wp-content/uploads/2017/01/mlmu_01_vlado_boza.pdf

# 6 Method of Salary Prediction and its Problems

Let us suppose, that we want to learn about job and salary preferences of the job seekers within the condition – mining them from the excessive data we dispose. Because we want to gain job market insight for the employers, they should be also given information from the other side of the river bank. Suppose they will want to gain an insight, what salary they should apply in a HR processes. Let us define that S = $\{S_1 \dots S_n\}$ are the job salaries, where each indexed S is a salary (e.g. integer, float integer) Salaries are assigned indexes by the following rule:

If there are computed weights of the job advertisements and that weights represent number of occurrences of a sector a job is offered, then assign them with a salary letter in a set S. Letter S would be given an index by a rule a higher number a weight is, lesser index will be assigned.

Job seekers want to find out the most likely salary they would get after they finish with degree.

## 6.1 Tasks and Conclusions

Dataset file Train_rev1.csv is as a whole set of vectors. Each of the dataset vectors are nonempty. Let us assume that exists at least one vector such as vector V, where one his dimension is equal that in S. It's defined as is, because having one record saying that there's a job with the offered wage $S_1$ = 1000 €, then we know we have some columns job_title1 and job_wage2=1000€ belonging to S. Let us define, that each of these vectors have assigned unique identifier. Our task is to find out the most demanded salaries Q as a subset of set called S. We can compute a subset Q as stated by the holding a following approach:

Let us define following set $\{(S_1 \dots S_{100})(S_1 \dots S_{50})\}$ containing two vectors representing the most demanded salaries. We define the minimum support count threshold equal to 1 as defined in[1.] In our example it's the number of rows in a file containing {(job_title1, job_wage2=1000€)}. We want to find two closed frequent salary vectors defined in[1]. They are $\{(S_1 \dots S_{100}):1(S_1 \dots S_{50}):2\}$. Their support counts are predefined as stated in[1] to numbers 1 and 2. Then we have maximal frequent salary vector $(S_1 \dots S_{100}):1$.

It's because the following rules:

- two closed frequent salary vectors are closed if and only if, there's some salary vector in the dataset containing one of the previous, but at least one different indexed S,
- two closed frequent salary vectors are frequent if and only if, they occurrences are greater than minimum vector occurrence or equal to. Minimum vector occurrence is equal to 1,

- we define a salary vector as maximum frequent salary vector if and only if, is it frequent and there exist no other maximum frequent salary vector S that would have at least one of the dimensions different than previous. Dimensions of the vector S represent salaries.

A problem we see, is to find and use an effective and scalable replacement for the Apriori Algorithm with a goal to find out the most demanded salaries as well as other frequent patterns stated above from the real-market job-oriented data we dispose.

# 7 Experiments

## 7.1 Word Cloud

For further data analysis, we created several Word Clouds. Word Cloud depicts most used words extracted from certain text or document. Size of the word in the visualization expresses its frequency in the given document. In Fig. 1, most frequent cities in which job is being offered are depicted. One can see, that London, Yorkshire, Hamshire and Surrey are the most common locations.

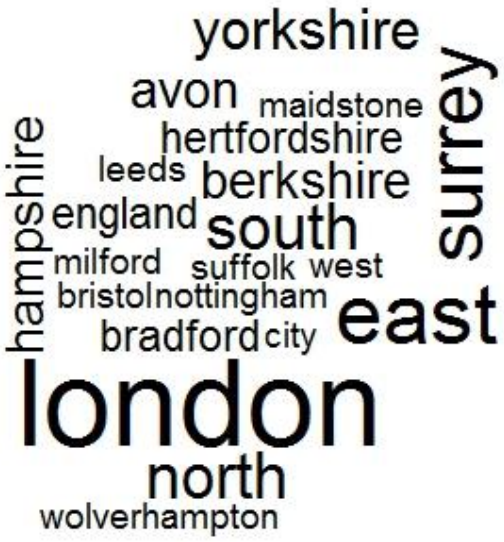


*Figure 1: Word Cloud of Cities*

# 8 Evaluation

After getting all the ngrams, training branch becomes the devtest branch by a simple command renaming the corpus attribute of the data variable. To evaluate model, we must build one and we call it ngrams.file. We will generate this file later and this file falls to the folder devtest, but because we changed terminology we call it a devtest branch.

## 8.1 Evaluation Method

As a method of evaluation, we are planning to use Mean Absolute Error (MAE), mainly because of its straightforwardness. MAE is an average absolute difference between predicted and real value. Many researchers demand this statistic to be made, since it is easy to interpret. Considering x being real value and y being the predicted one, MAE could be expressed by:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{\mathrm{n}}$$

# References


1. Han, J., Kamber, M., Pei, J.: Data Mining: Concepts and Techniques. 3rd edn. Morgan Kaufmann, Elsevier(2012).
2. Chunmian Ge, Atreyi Kankanhalli, and Ke-Wei Huang. 2015. Investigating the Determinants of Starting Salary of IT Graduates. *SIGMIS Database* 46, 4 (November 2015), 9-25. DOI: https://doi.org/10.1145/2843824.2843826
3. Aron Culotta. 2010. Towards detecting influenza epidemics by analyzing Twitter messages. In *Proceedings of the First Workshop on Social Media Analytics* (SOMA '10). ACM, New York, NY, USA, 115-122. DOI=http://dx.doi.org/10.1145/1964858.1964874
4. Todd Bodnar and Marcel Salathé. 2013. Validating models for disease detection using twitter. In *Proceedings of the 22nd International Conference on World Wide Web* (WWW '13 Companion). ACM, New York, NY, USA, 699-702. DOI: http://dx.doi.org/10.1145/2487788.2488027
5. Ming-wei Chang, Wen-tau Yih, and Christopher Meek. 2008. Partitioned logistic regression for spam filtering. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (KDD '08). ACM, New York, NY, USA, 97-105. DOI: https://doi.org/10.1145/1401890.1401907
6. https://www.kaggle.com/c/job-salary-prediction/