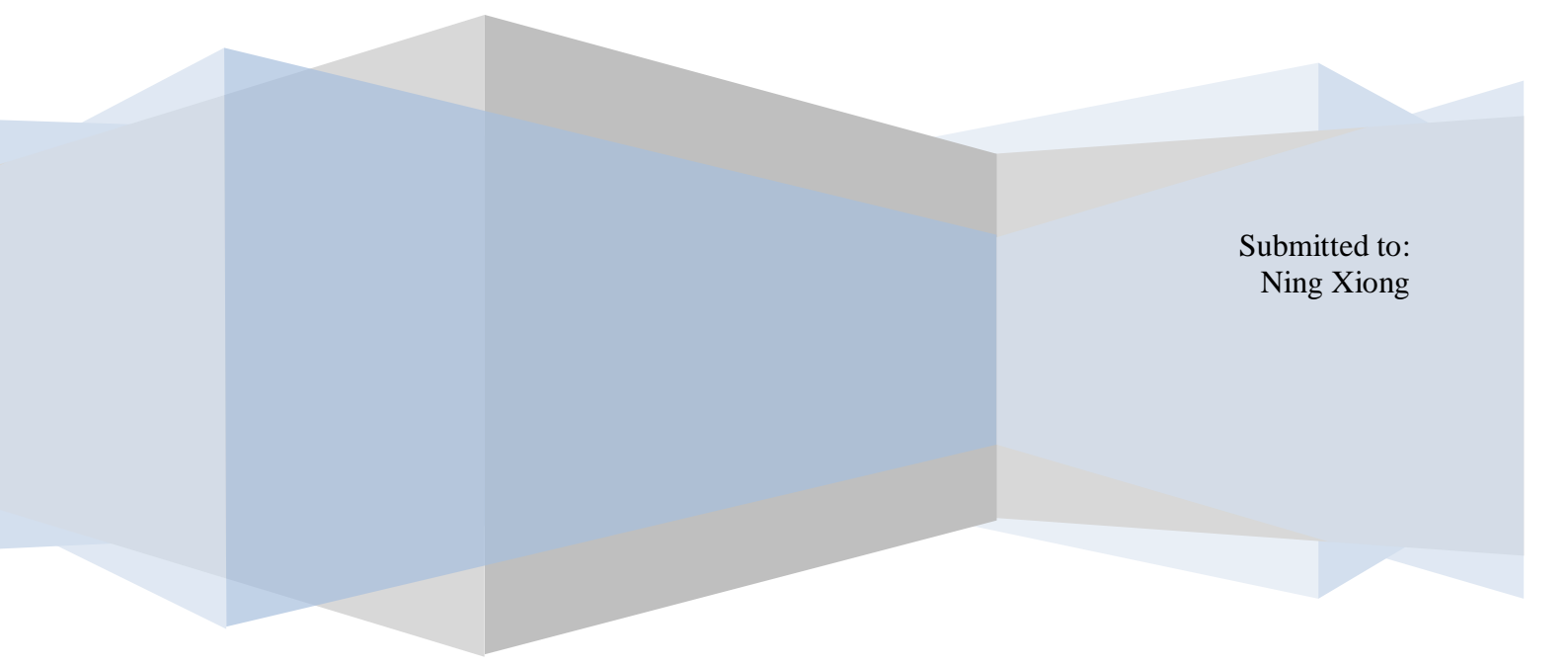


School of Innovation, Design and Technology,
Mälardalen University,
Sweden

Job Salary Prediction

A Project in Intelligent Systems



Submitted to:
Ning Xiong

Job Salary Prediction

A Project in Intelligent Systems

Debajyoti Nag
Computer Science Student,
IDT,
Mälardalens Högskola
Västerås, Sweden
dave0908@gmail.com

Tobias Larsen
Computer Science Student,
IDT,
Mälardalens Högskola
Västerås, Sweden.
tobias_larsen@hotmail.com

Gerard Duff
Computer Science Student,
IDT,
Mälardalens Högskola
Västerås, Sweden
gerardduff@gmail.com

Omer Saleh Mohammed
Computer Science Student,
IDT,
Mälardalens Högskola
Västerås, Sweden.
osh12002@student.mdh.se

Abstract – This report documents the project completed in the Intelligent Systems course at Mälardalens Högskola in Västerås, Sweden. The project members are four computer science students studying at the university(3). In this project, various different artificial intelligence methods were used independently and in various combinations. To study the effectiveness of each method, a prediction competition was entered. The competition was to predict the salary of a job based on input attributes. The competition was hosted on the Kaggle website (1), and sponsored by an employment agency called Adzuna based in the UK (2).

I. INTRODUCTION

This report covers the project in the course *Intelligent Systems*, DVA406, at Mälardalens Högskola in the winter of 2013(4). The project group was an international group consisting of four members; computer science students studying at the school of Innovation, Design and Technology at Mälardalen University (5).

The goal of the project was to develop a prediction model, implementing various AI methods and also to study the effectiveness of each method. Results from every method implemented, combinations and individual, were submitted in a job salary prediction competition to observe the performance.

The four methods implemented were:

- Artificial Neural Networks
- Decision Trees
- Random Forest
- Hybrid Method

The competition entered was hosted by Kaggle(1) and organized/sponsored by an employment agency called Adzuna(2). The website provided a database of existing jobs, taken from various sources, to be used as training data. The website also provided a separate database to be used as the validation data.

Each method was trained using the training data and then tested against the validation data. The result produced for the validation data was uploaded to the website, which then calculated the error and relative position of the team on the leader board.

The following report contains the preprocessing steps undertaken to clean the data, a description of each method implemented, the post processing steps taken to improve the estimated salaries, and a brief overview of the results and observations made.

II. PREPROCESSING

The training data set provided by the website for this project was not perfect. A number of preprocessing techniques were used to clean the training data for general as well as for usage with each technique. These steps were meant to convert non-numerical, natural-language values to numerical (real or binary) inputs.

The provided data and cleaning techniques used were as follows:

1. **Id** - A unique identifier for each job ad, no changes were required.
2. **Title** - A free text field provided as the Title of the job ad. Normally this is a summary of the job description or role.

This data was split into tokens, and those tokens were selectively used for training. It worked surprisingly well for the Hybrid Method.

3. **FullDescription** – the full text ad of the job as provided by the advertiser. It was stripped of data, by Kaggle, to conceal salary informations, and hence resulted in a messy string.

An approach similar to Title attribute was tried, but it did not produce any significant improvements.

4. **LocationRaw** - The free text, unrefined location as provided by the job advertiser. This data was used where LocationNormalized was ambiguous or missing.
5. **LocationNormalized** – Free text, but refined location. Missing and ambiguous entries were replaced with keywords extracted from LocationRaw.
6. **ContractType** – natural language values, but of limited range: FullTime, PartTime or the value was missing. They were converted to numeric values, which is a bit nonsensical in case of ANN.
7. **ContractTime** – Natural language values, with limited variation. Permanent, Contract or the value was missing. Treated in a similar manner as ContractType.
8. **Company** – The name of the employer as supplied in the job ad. We did not use this attribute anywhere. But it might have some important role in the future scope of the project. Those remain to be explored.
9. **Category** – Natural language values with limited variation. Each job entry had a job category attribute which characterized the job into one of thirty general employment fields.
10. **SalaryRaw** - the free text, unrefined salary field we received in the job advert from the

advertiser.

Often mentions a range instead of a crisp value. This was used when SalaryNormalised was missing.

11. **SalaryNormalised** - the annual salary interpreted by website from the raw salary. This is the value we are trying to predict. In the training data, average of SalaryRaw was used when this field was missing.

Firstly, the attribute data was examined to find any pattern or relation among these features.

This heat-map shows variation in average SalaryNormalised with change in Category and ContractTime.

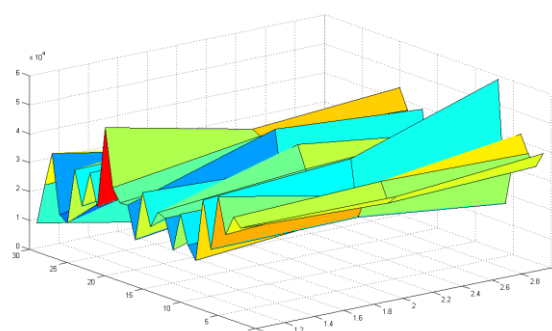


Fig. 1

This shows a similar relation for ContractType and Category.

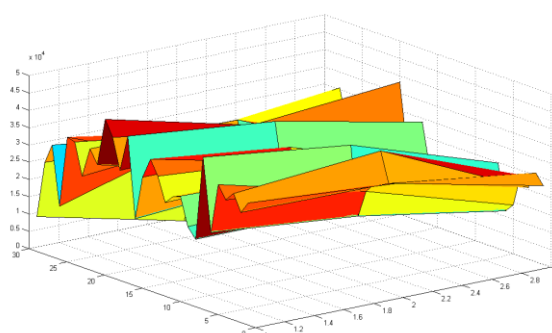


Fig. 2

Regional density for the average SalaryNormalised, using data in tree format provided by Kaggle, is shown in fig. 3

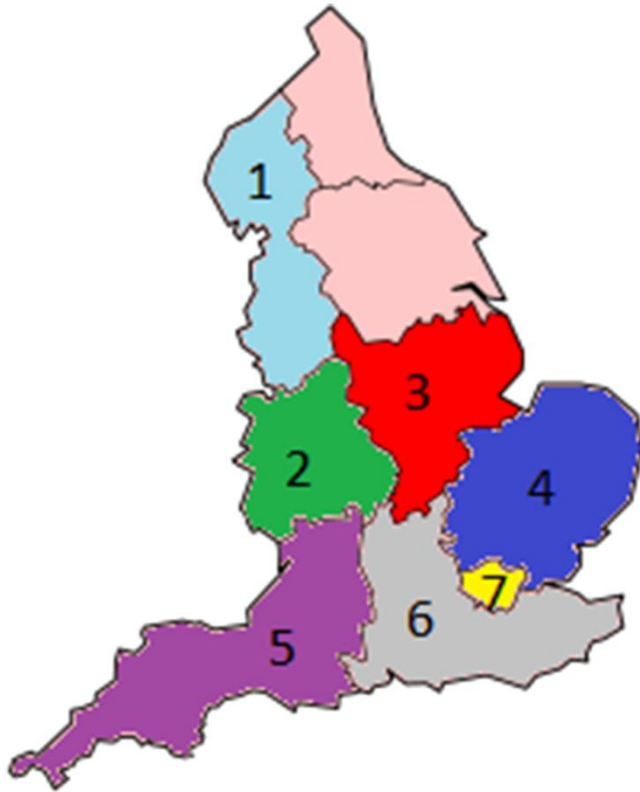


Fig. 3

| Key | Location | Avg. Salary | Visits |
|-----|--------------------|-------------|--------|
| 1 | North West England | 28559.18 | 15743 |
| 2 | West Midlands | 30490.21 | 15696 |
| 3 | East Midlands | 29007.87 | 13036 |
| 4 | Eastern England | 31602.33 | 12207 |
| 5 | South West England | 30645.48 | 17743 |
| 6 | South East England | 32408.29 | 40447 |
| 7 | London | 42116.62 | 60579 |
| 8 | South East London | 40871.72 | 12802 |
| 9 | UK | 34122.58 | 244768 |

A threshold of 10000 visits was required for a region to qualify for the above table.
Note that region 8 is too small to be shown on map.

III.LEARNING

ARTIFICIAL NEURAL NETWORKS

Introduction and Implementation

The Project was started with simple, back propagation Neural Networks with one hidden layer.

Two ANNs were implemented to study the effect of converting discontinuous, non-numeric data to continuous, numeric form.

One ANN was implemented in Python, to process real-valued inputs, and the other in Java, to process binary inputs. The choice of programming language was based on the preference and experience of the programmers, and did not effect the results as the same algorithm was used in both implementations. The call graph for the implemented algorithm is shown in fig. 4.

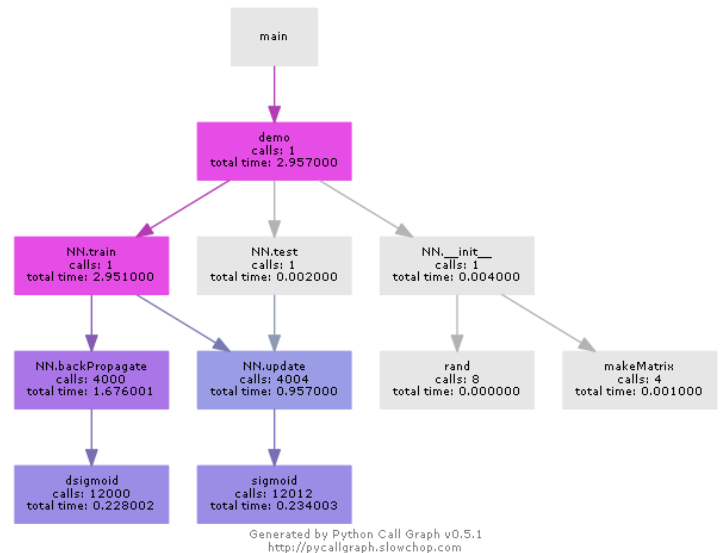


Fig. 4

Attributes

The first key attributes used were ContractType, ContractTime and Category.

The Python ANN had 3-input neurons in the input layer while the Java ANN had 35-input neurons.

Result

As a result, the Binary valued ANN was found to perform better than the Real valued ANN.

ValidationError(Real-valued ANN) = 11995

ValidationError(Binary valued ANN)=10091

The suspected reason for this is the conversion of discontinuous data to numeric form, as the non-implied distortion in relation among data points, introduced in the data during the conversion, is less in case of binary inputs.

The LocationNormalised feature was added as an input, resulting in 56 binary inputs in the input layer, but any improvements were not achieved.

DECISION TREES

Introduction

One of the methods implemented during this project was constructing a decision tree. As this project was based on attributes with discrete values it was projected that implementing a decision tree would yield the best results.

The programming language chosen to implement the decision tree was python. Python was chosen as the programming language has it has powerful tools to read, write and manipulate comma separated value (CSV) files. It also has useful features for dynamically creating lists in its dynamic type system and automatic memory management, which made implementing the decision tree simpler. (6)

Attributes

Firstly, four key attributes were chosen to implement a basic decision tree decision model. The attributes chosen were location normalized, job category, contract type and contract time.

The location normalized attribute was the city nearest to where the job was located, i.e. London, Manchester, Glasgow. However in some cases the location was simply normalized to "UK". As this value was not sufficiently specific, these entries had to be cleaned.

A list of locations in the UK, a location tree, was compiled. The "location raw" attribute, which was a description string of the location, was then analyzed and a new normalized location was found by splitting the location raw attribute into a list of words, and searching that list of words for the name of a city included in the location tree. When the new normalized location was found, it was extracted and replaced "UK" in the location normalized field.

The contract type attribute had three possible values: permanent, contract, or undefined. The contract time also had three possible values: full time, part time, and undefined. The job category attribute had 29 distinct values, as can be seen in the results section.

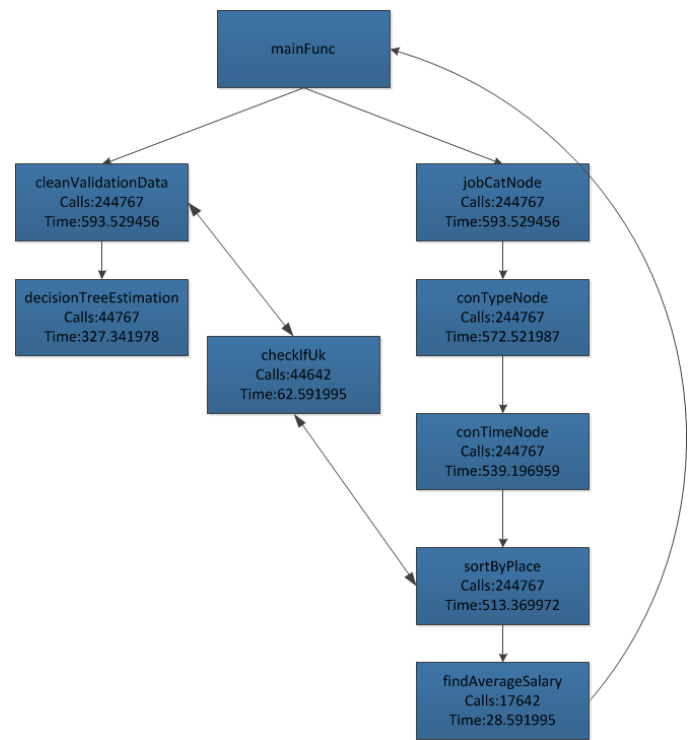


Fig. 5

Implementation

After the key attributes were chosen, an algorithm was designed and implemented. This algorithm evolved throughout the project to try and find the optimum algorithm that gave the best and most accurate results. The call graph generated with the assistance of pygraph can be seen in figure 4.

For normal classification problems, entropy is used to decide which attribute to use at each node. (7) However this project was not a classification problem as the exact salary was needed so another approach was needed.

The first approach taken was arbitrarily choosing which attributes to use at each node. The reasoning behind this approach was to get used to using the new programming language python, and learn how to implement a decision tree.

The next approach undertaken was to use the standard deviation from the average to choose which attribute to use. So the training data was split in half, and the first half was used as training data, and the second half was used as validation data. The average salary was found for each attribute value and the range of salaries for each attribute. The attribute which had the least outliers was chosen at each node.

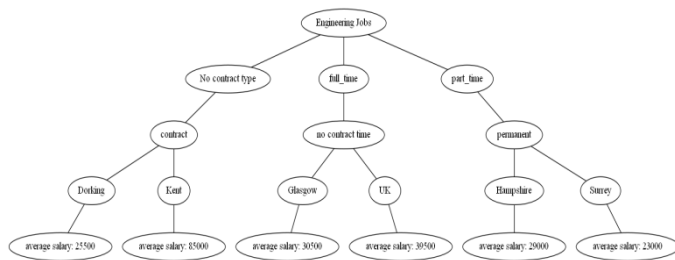


Fig. 6

Another approach that was taken was to decide which attribute yielded the most information at each node.

After this tree was built, it was evolved to include another attribute job title. The job title information was cleaned by removing unnecessary text. For example, for the entry “Systems Analyst Engineer in London”, the location London, and the word “in” was removed as the location data was already included in the location node. This process was repeated for each entry, and the exact job title was abstracted for each entry.

To estimate the salary the average of the predicted salaries was used. So when building the tree, if there were two or more entries with the same attributes yet different salaries, the average of the two salaries was used as the salary. If there were several salaries, the average and standard deviation was calculated. Each of the salaries were checked to see if any of the salaries were greater than the average plus the standard deviation, and if there were they were marked as executive salaries.

When a validation entry was looking for an estimated salary and the leaf yielded several salaries, the validation entry was checked to see if the job description contained a keyword such as “director”, “manager”, “head”, and “competitive salary”. If the entry contained any of these words then the entry was given the executive salary, and if not then it was given the average salary.

Result

A sample extract from the final Decision Tree was plotted and can be seen in figure 5. The best result found for the decision tree was:

ValidationError = 9676

Introduction

Random forests is learning ensemble method for classification and regression that fits a number of decision trees classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over fitting .It is better to think of a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest.(8)

The constructional model for the Random forest is:

1. The shape of the decision to use in each node.
2. The type of predictor to use in each leaf.
3. The splitting objective to optimize in each node.
4. The method for injecting randomness into trees.

In this case, the simplest type of decision was used by using the growing of ensemble of decision trees, the random vectors are generated that govern the growth of each tree in the ensemble, after that each tree votes for the most popular class.

The generalization error of a forest of tree classifiers depends on the strength of the individual trees in the forest and the correlation between them. Python language was used which lets us work more quickly and integrate our system to process all of the data that was taken from the computation site(1) but the problem was that the data was so big, it crashed the computers .

The RF package was found to optionally produce two additional information attributes:

1. Measure of the importance of the predictor variables.
2. Measure of internal structure of the data.

The figure 7 below shows how the data was dealt with, if form of the model for the random forest:

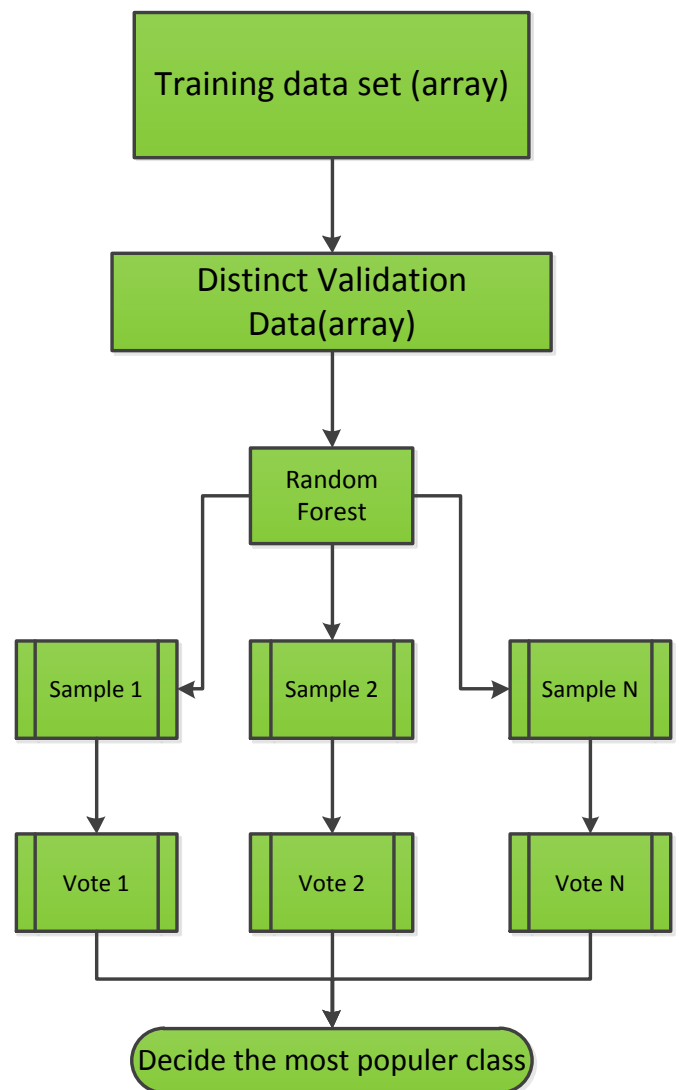


Fig. 7

Introduction

A *genetic algorithm* (GA) is an optimization and search algorithm which doesn't require a data model. Its evaluation metric relies solely on the input data and an evaluation function (in GA terms; *fitness function*). A genetic algorithm, as the name suggests, mimics real world evolution in its mission to gain progress.

The *population size* was chosen to consist of 16 *individuals*. The *genes* for each individual were set to be a bit array of a fixed length of 300. Each bit had a 30% chance of initially being set.

No explicit *termination criteria* based on the achieved result was used. The GA executed for 30 generations since, in the typical case, there was no significant improvements after the 15-20 generation.

Fitness function

The fitness was, obviously, the reversed salary error. We used a quarter of the training data as validation data to calculate the salary error. A more proper error calculation, like the *K-fold cross validation* algorithm could have been used. However, when the salary error was calculated for the validation set and submitted the actual salary error, as calculated by Kaggle, it surprisingly differed by less than a percent

Selection

A simple strategy was used, *pure elitist selection*, meaning that the two fittest individuals were chosen for the crossover. The strongest individual was also retained in the population so no generation slump could transpire.

Crossover

A uniformly random *one point crossover* was used as depicted in the Fig. 8.

| | |
|-----------|---------------|
| Mama | 1011001110011 |
| Papa | 0100110010010 |
| Offspring | 1011001010010 |

One point crossover index

Fig. 8

Mutation

The number of bits was related to the generation using this formula:

$$2 + R \left(\frac{50}{\sqrt{g + 3}} \right)$$

R: Uniform random function
g: zero based generation index

The following graph (Fig. 9) better highlight the characteristics of the formula. It is a common strategy to use a higher mutation ratio to broaden the search space for earlier generations.

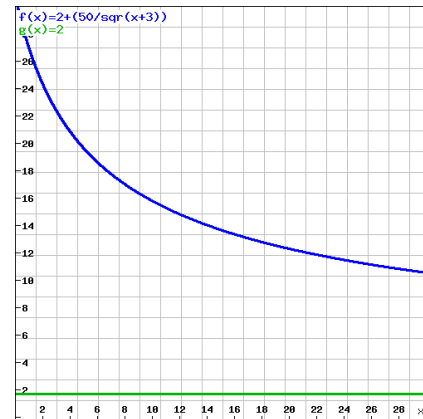


Fig. 9

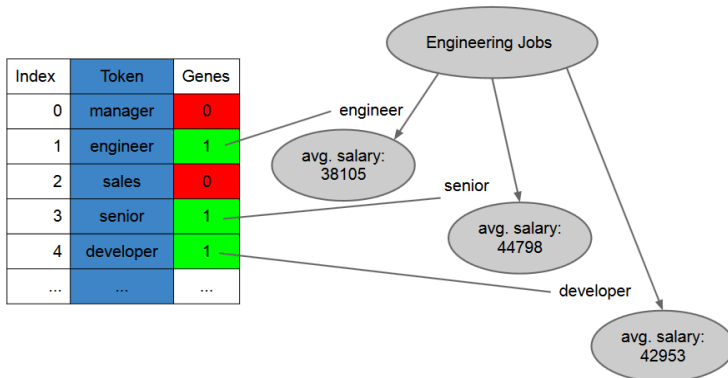
The *blue* line is the maximum random value for number of bits mutated and the *green* line is the offset used to guarantee that at least 2 bits gets mutated.

Note that a mutated bit is flipped, i.e., it changes value, hence is the number of set bits modified. Another strategy would be to randomly select two indexes, one false bit and one true bit and interchange their values thus keeping the number of bits set consistent.

Parallel execution

It took approximately 110 seconds for each generation to be processed when it was evaluated and modified by a single thread. However, the fitness calculation was performed by a thread pool with a maximum degree of parallelism of four since the processor for the GA environment had four cores (a higher degree actually caused an out of memory exception since each thread allocates a lot of memory). The average time for each generation to be processed when executing in parallel mode was slightly under 30 seconds. The population size was deliberately chosen to be a multiple of the maximum degree of parallelism which was four. The offspring calculations was not ran in parallel since it was just a small degree (almost negligible) of the computational load of the GA and there is also a downside of parallelism with more complicated code (with the negative side effect of

bugs creeping into the system), thread scheduling overhead etc.



Genetic algorithm to decision tree adaptation
Fig. 10

Above is a picture depicting the relationship between the genes (1 is true, 0 is false) for each individual and the actual impact on the decision tree. If a bit in genes is set for a particular index, the corresponding token at the same index is used as an attribute for the decision tree.

Since the title feature was free text and thus may contain multiple tokens, cases where multiple sub nodes should be visited often arises. E.g., assume a “senior developer” title description for the subtree above. In these cases the average of the predicted salaries are used for the prediction.

If the decision tree has no realized nodes for a title description a “default” node is used. When looking at the outcome of the GA we see that the default node is not commonly used. This is intuitively correct since the GA should optimize the diversity of the title tokens used and thus avoiding the default node to gain information.

As a side note, the true average salaries are shown in fig. 11.

Result

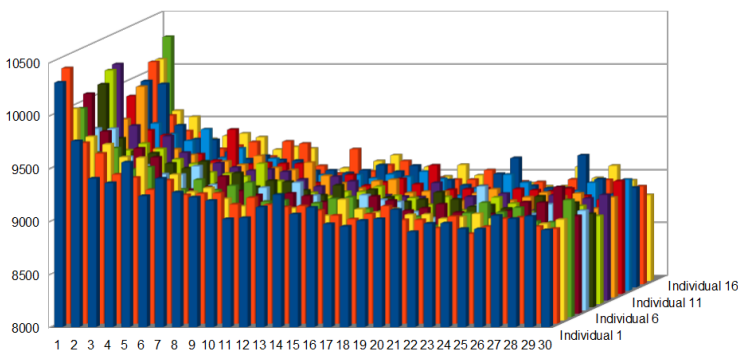


Fig. 11

Average error: 8895.44397

IV. Post processing

An attempt was made to combine Decision Trees and ANN where the tree calls a specific ANN based on the Location, but no significant improvements were achieved.

V. Results

The Best result obtained was for the hybrid method.

Best result statistics for each method are mentioned in fig 10.

| Method | Error |
|-------------------------|-------|
| Mean Benchmark | 13257 |
| Random Forest Benchmark | 7569 |
| ANN | 10091 |
| DT | 9676 |
| RF | 15586 |
| Hybrid Method | 8895 |

Fig. 12

A relation between the data attributes from our best prediction is depicted graphically.

The variation in average salary with Category is shown in fig 13.

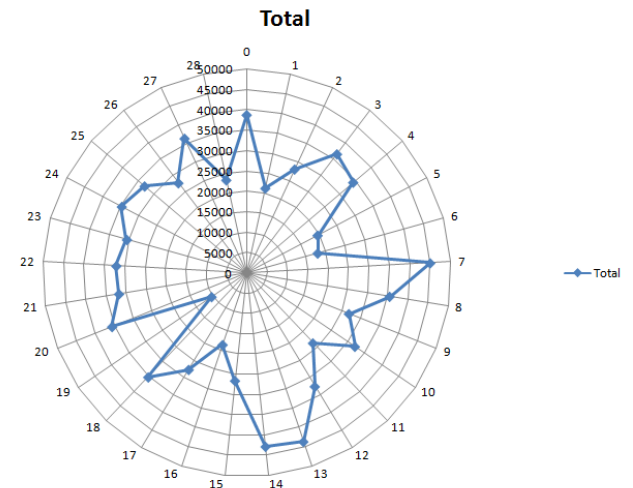


Fig 13.

The variation in average salary with ContractTime is shown in fig 14.

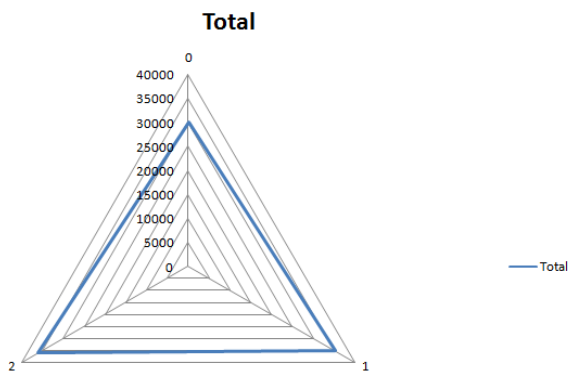


Fig 14.

The variation in average salary with ContractType is shown in fig 15.

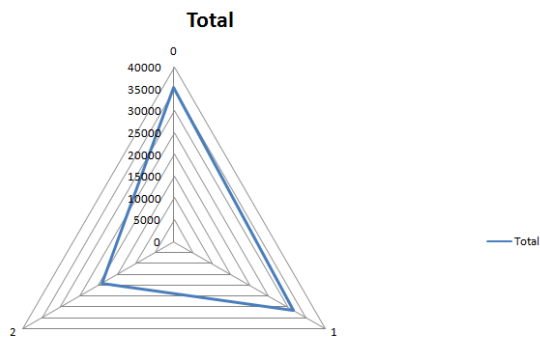


Fig 15.

VI. References

- [1]Kaggle (2010) Kaggle data mining competitions, [online]
Available: <http://www.kaggle.com>
- [2]Adzuna (2010) Adzuna job advertising website, [online]
Available: www.Adzuna.co.uk
- [3]Mälardalen University
Available : <http://www.mdh.se/>
- [4]Unknown author (2012) Intelligent Systems website. [online]
Available:
<http://www.mdh.se/studieinformation/VisaTillfalle?anmalningskod=14008&termin=20131>
- [5]Unknown author (2012) IDT department at Mälardalens Högskola. [online]
Available: www.mdh.se/idt
- [6]Wikipedia (2010) python wiki page. [online]
Available:

[http://en.wikipedia.org/wiki/Python_\(programming language\)](http://en.wikipedia.org/wiki/Python_(programming_language))

[7]PyGraph (2012) pygraph graphing library for python. [online]

Available:

<http://sourceforge.net/projects/pynetwork/>

[8]Alex Berson & Stephan Smith,(2004) “Data warehousing, Data Mining, & OLAP”,

[9]Paper of classification by RandomForest by Andy liaw 2002

[10]work done by Neil Schemenauer

Available: <http://arctrix.com/nas/>