Predictive Modeling for Salary Estimation

Julia Bazarbachian
26781357

Abstract

In this project, we built Machine Learning models that generate salary range estimates for jobs in the field of Software Engineering and Computer science. The dataset was collected by scraping job offers found by means of keyword search on the popular job board site Indeed. A combination of feature extraction and text classification methods were employed in this study to train and test the proposed salary prediction models. The text vectorization techniques that were leveraged are the Bag of Words, Term Frequency, TF-IDF, and N-gram models. The classification algorithms that were employed for salary class prediction are the Logistic Regression, Naive Bayes, and Stochastic Gradient Descent models. Performance metrics such as accuracy, precision, recall, and f1-score were used to compare the success of the models. Cross-validation was used to validate accuracy, and hyperparameter search was used to improve performance results.

1. Introduction

The goal of this project was to produce wage forecasts for job offers in the fields of Software Engineering and Computer Science in the current Canadian market. Despite the wealth of information accessible for the positions being advertised, including roles and responsibilities, required skills, company ratings, and employee reviews, salary information is typically withheld. Discussions about compensation are however a pivotal part of the hiring process. Candidates are frequently expected to take the lead in pay negotiations by expressing their expectations for the proposed role's salary range, while also basing their estimates on their own qualifications and work experiences. Inevitably, remuneration is a major consideration in deciding whether or not to pursue a particular position. With this in mind, I set out to create prediction models that would meet this need by forecasting wage ranges for job ads in my field of study. Both graduates entering the workforce and those searching for a change of career might benefit from such a tool.

In this project, salary data is obtained from prominent job search engine Indeed [1]. The elements of this dataset include the following fields: job title, company name, location, job description, and salary. The input will therefore be unstructured raw text documents. Because most salary offers are presented as ranges with high variability, and a single salary value would not be informative, we aim to predict salary as range values as well. This project thereby falls under the category of text classification with supervised learning models, where unstructured text is input, features are extracted, and a salary class is output.

2. Methodology

Developing my salary prediction models was a multi-step process involving web scraping, dataset preparation, feature engineering, model training, performance evaluation, and hyperparameter tuning. We will go over each step of the project in this section.

2.1. Web Scraping Job Postings

In order to build my dataset, I needed to collect several thousands of job ads. To do so, I used the Indeed Scraper from APIFY [2]. Although I successfully built a web scraper of my own, the website would block it after accumulating roughly 900 job ads. I used Apify's pre-built Indeed scraper to get around this issue and to acquire a higher number of data points. Using keywords "software engineering" and "computer science" as job field entries and "montreal qc", "toronto on" and "vancouver bc" for the location field, I collected job offers for Canada's top 3 tech cities. The result was 6 csv files with the following fields:
positionName: Job title
company: Company name
location: City, province
rating: Company rating
reviewsCount: Number of reviews
url: URL of job ad
id: Unique job key
postedAt: Date of job posting
scrapedAt: Date of job scraping
description: Job description

2.2. Dataset Preparation

Now that we have acquired our content, we proceed to enrichment with the goal of building a dataset fit for our machine learning models. This is an iterative process involving data exploration and cleaning. To begin, all job postings that did not include salary information were removed from the dataset. The id column was then used to remove duplicate jobs. We are now down to 835 jobs from the original 5986 that were scraped. This means that only 13.9 percent of job postings included salary information. The two columns of interest that were explored in this phase were salary and description. Looking at the contents of these cells, we could observe that salaries were given in different formats, such as hourly, monthly, and yearly values. Most were ranges, while some were singular values. Looking at an individual job description value, we could see that it was an unstructured text document. Text data requires preprocessing and vectorization in order to be transformed into features. The following step will thereby involve transforming these values into usable feature and label columns.

2.3. Feature Engineering

In this section, new columns were created from the scraped and cleaned data from the previous steps. The salary column was parsed into a single average salary column, which represents the given salary for singular inputs and the mean of both values for input ranges. Only yearly wages were considered in order for all salary estimates to be on the same scale. Then, using Interquartile Range, outlier values were removed from the salary data. Once outliers were dropped, I explored creating salary bins of different sizes to use as target labels.

Looking at bar plots for different numbers of label classes, I observed that with my collected data, the most balanced number of classes was 2. If I used a larger number of targets with this dataset, I would have had a disproportionate number of observations in each class. If I were to use imbalanced label classes in my machine learning models, accuracy would no longer be a very informative performance metric. In addition to having to change performance measures, I would also need to further pre-process the data in order to handle imbalanced classes. For these reasons, I proceeded with building binary classification models. As for the job description text, preprocessing involved applying Natural Language Processing techniques with NLTK to tokenize, stem, lemmatize and remove stop words.

2.4. Model Training and Performance Evaluation

Now that feature and target columns are preprocessed, I proceeded with the development of classification machine learning models. To encode the text data into feature vectors, I used scikit-learn's CountVectorizer [3] and

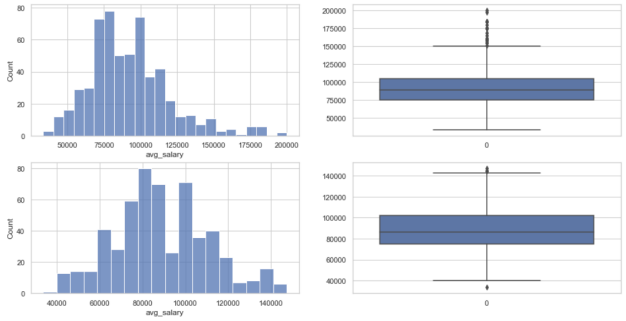

Figure 1. Removing salary outliers.

TfidfTransformer [4] functions. Using different parameter values and pipelines, features were vectorized using the Bag of Words, Term Frequency, Term Frequency-Inverse Document Frequency and N-gram models. Since different vector representations of text data produce different results, multiple encodings were chosen for the sake of comparison.

Next, three machine learning classification models were trained for each of these vectorization models in order to produce salary range predictions. The classifiers that were trained were Logistic Regression, Naive Bayes, and Stochastic Gradient Descent, models that were known to perform well for text classification applications. Performance was then compared by generating a classification report, cross-validation scores, and confusion matrices.

2.5. Hyperparameter Tuning

The final part of my data analysis and classification project consisted of hyperparameter tuning for improved model performance. Scikit-learn's GridSearch [5] function was used to tune hyperparamters and find the settings that gave the best results on the test data. GridSearch accuracies were then calculated for each model and results were collected for comparison.

The specific dataset I employed, the combination of feature vectorizations and classification models, as well as my salary range outputs are my original work. Based on my preliminary research, salary prediction was mostly performed using regression models with historic data of user reported wages that also included correlated variables such as years of experience, level of education, etc. Salary prediction for web scraped job offers can also be found as an end-to-end data analysis project, but the proposed solutions all looked for jobs with the same position title, created small feature vectors by extracting specific information from job descriptions (contract type, seniority level, etc), and created binary classification models to predict weather a job's salary would be above or below a calculated median value. As

```
Accuracy score: 66.7%
Cross-validated scores: [0.6741573  0.73033708 0.64772727 0.68181818 0.69318182]
                    precision    recall  f1-score   support

(33499.999, 86500.0]       0.62      0.79      0.69        53
 (86500.0, 147500.0]       0.74      0.55      0.63        58

            accuracy                           0.67       111
           macro avg       0.68      0.67      0.66       111
        weighted avg       0.68      0.67      0.66       111
```

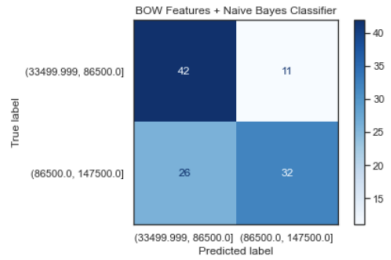

Figure 2. Performance Evaluation Output.

```
Parameter setting that gave the best results on the hold out data:
{'clf__alpha': 0.001, 'vect__max_features': None, 'vect__ngram_range': (2, 3)}
Grid search accuracy:
 99.1% train accuracy
 76.6% test accuracy
```

Figure 3. Hyperparameter Tuning Output.

such, I believe that my work adds to the existing job offer wage prediction solutions.

3. Experimental Results

Table 1 summarizes accuracy scores for all 9 models trained and tested with different vectorization and classification model combinations. Naive Bayes and SGD algorithms performed better when job description text was encoded with the TF-IDF model, while the Logistic Regression algorithm performed best with the Bag of Words vectorization model. These results changed when accuracy was optimized through hyperparameter tuning. An accuracy of 76.6% was recorded for both Naive Bayes and SGD classifiers with BoW vectorization when parameter values were changed, while Logistic Regression accuracy improved to 73.9% with Tf/Tf-idf models. Looking at the confusion matrices for each model, it can also be observed that the models are all more efficient at classifying (33499.999, 86500.0] salary range jobs.

Model	Vectorization	Classifier	Accuracy	Optimized Accuracy
1	BoW	LogisticRegression	70.3%	70.3%
2	TF	LogisticRegression	66.7%	73.9%
3	TF-IDF	LogisticRegression	67.6%	73.9%
4	BoW	MultinomialNB	66.7%	76.6%
5	TF	MultinomialNB	65.8%	73.0%
6	TF-IDF	MultinomialNB	68.5%	71.2%
7	BoW	SGDClassifier	70.3%	76.6%
8	TF	SGDClassifier	71.2%	72.1%
9	TF-IDF	SGDClassifier	74.8%	73.0%

Table 1. Performance accuracy results.

4. Conclusions

In the end, no definitive inference could be made about which vectorization method outperformed the others or which classifier produced a more accurate salary range estimate since the combination of the latter two along with the tuning of each models' hyperparameters produced different results. What my own tests did conclude however, was that my best models could predict a job's salary range class with an accuracy of 76.6%, which is acceptable for a first attempt. That being said, there is undoubtedly room for improvement in future attempts to solve the job offer salary prediction models. Firstly, adding more data points and preprocessing the data to mitigate unbalanced salary range classes would allow for multi-class classification, which in turn would produce a more informative salary range prediction. Next, I would spend more time in the feature engineering phase, testing other text embedding techniques such as word embeddings with neural network algorithms. Lastly, I would attempt to reduce the dimentionality of the feature vectors in hopes of improving classification results.

REFERENCES

[1] "Job Search Canada Indeed." https://ca.indeed.com/

[2] "Indeed Scraper Apify," Apify. https://apify.com/hynekhruska/indeed-scraper

[3] "sklearn.feature_extraction.text.CountVectorizer", scikit-learn. https://scikit-learn/stable/modules/ generated/sklearn.feature_extraction.text.CountVectorizer.html

[4] "sklearn.feature_extraction.text.TfidfTransformer," scikit-learn. https://scikit-learn/stable/modules/ generated/sklearn.feature_extraction.text.TfidfTransformer.html

[5] "sklearn.model_selection.GridSearchCV," scikit-learn. https://scikit-learn/stable/modules/ generated/sklearn.model_selection.GridSearchCV.html