Justin Zhang

Predict Salary Through Job Description

In the 21st Century, more and more things are done through the web, and that includes job applications. The job description's posting are written in words, some words are more important than others. Through supervised machine learning algorithms, job salary could be predicted based off of the location, industry, and the description of the posting. The machine learning algorithm could be implemented into a job search site where it can estimate salary range. This implementation can help applicants filter through thousands of jobs in the search engine, and gives the site a competitive advantage over other sites without the functionality.

Data

The data for this capstone is obtained from kaggle (https://www.kaggle.com/c/job-salary-prediction/data), and the file name is (Train_rev1.zip). Since it is from kaggle, most of the data wrangling has been done. Each row contains a job posting, and the columns describe different properties of the posting. The significant columns include 'FullDescription', which contains the job posting's content. 'Title' column is the name of the job. 'LocationNormalized' column is the location of the job after processing the 'Location' column. 'Category' column is the industry of the job posting. Each posting is identified by an unique 'ID'. Time of each job posting is not specified.

Data Wrangling

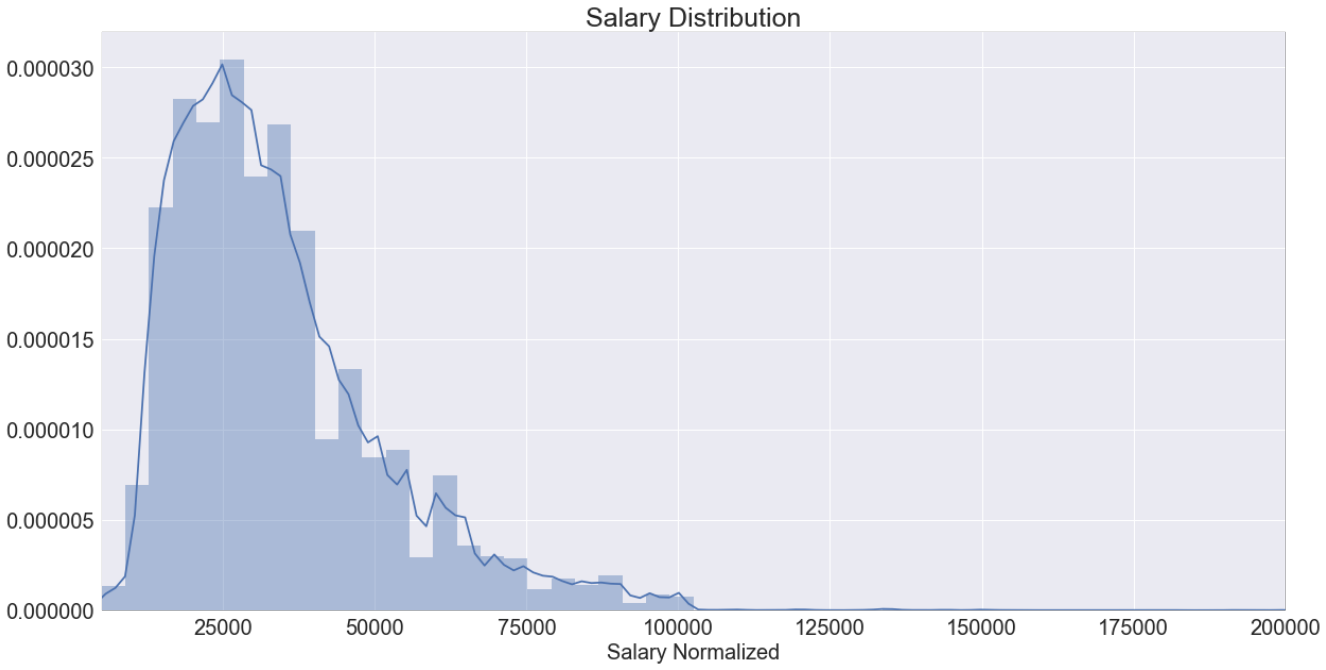
The kaggle competition has been cleaned prior to publishing, and only one missing value has been detected under the 'Title' column. For simplicity, I removed that one entry since the dataset has 244,768 records. 'ContractTime', 'ContractType', and 'Company' columns have over 63,000, 179,000, and 32,000 respectively. Therefore, these columns will not be used during model building.

For interpret-ability of the resulting accuracy, ‘SalaryNormalized’ column has been divided into above and below median salary (30,000). By dividing salary into two category turns the project into a binary classification problem, and an accuracy score of 50% and above indicates a working model.

Exploratory Data Analysis

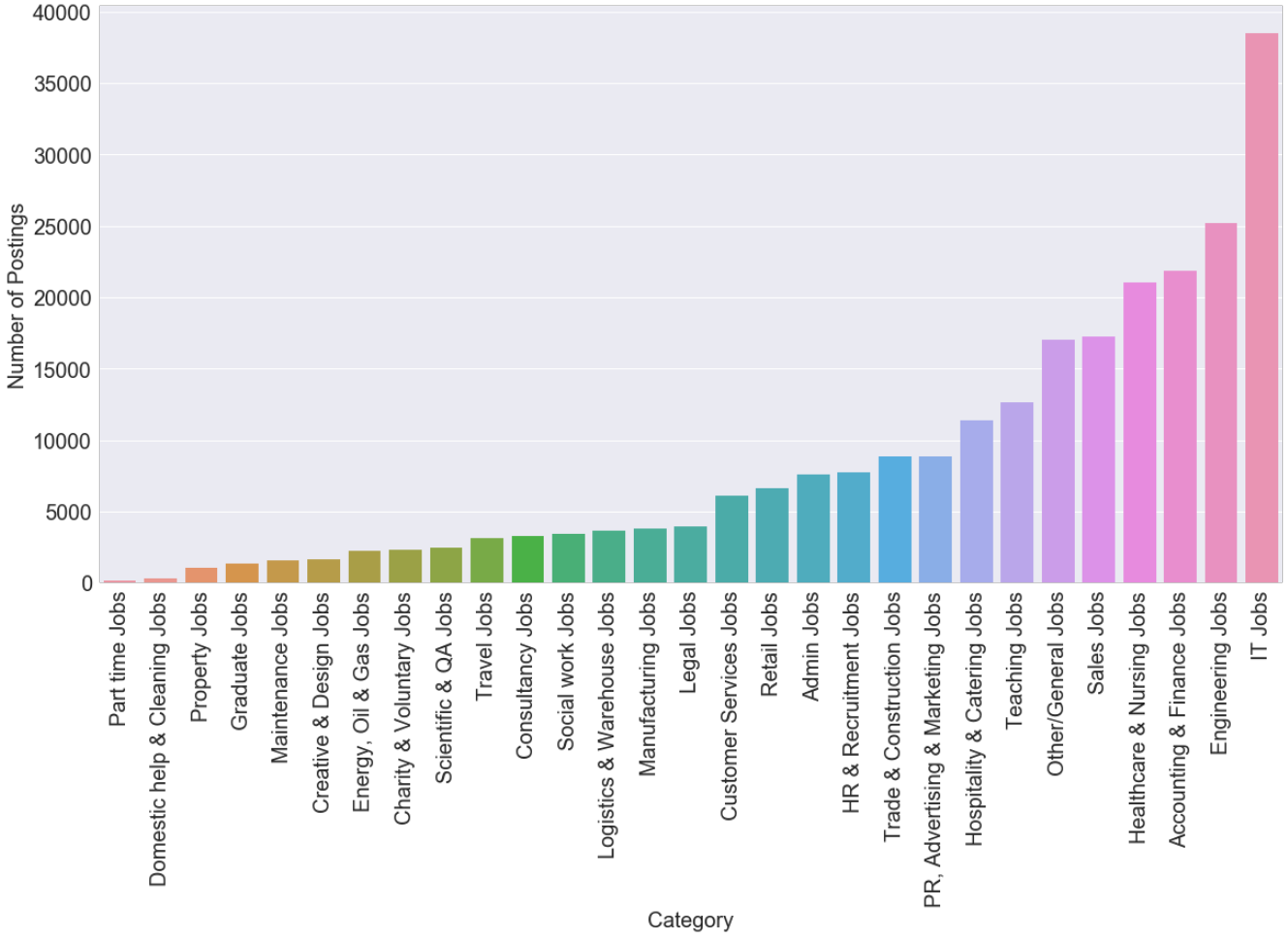
This step of the project is for understanding the data, and it is especially true for a natural language processing project since the data will aggregated and then transformed into numbers during the model building stage.

To understand the data better, the median and mean of the ‘SalaryNormalized’ column is calculated, and they are 30,000 and 34,122 respectively. Then a plot is made to visualize the distribution.

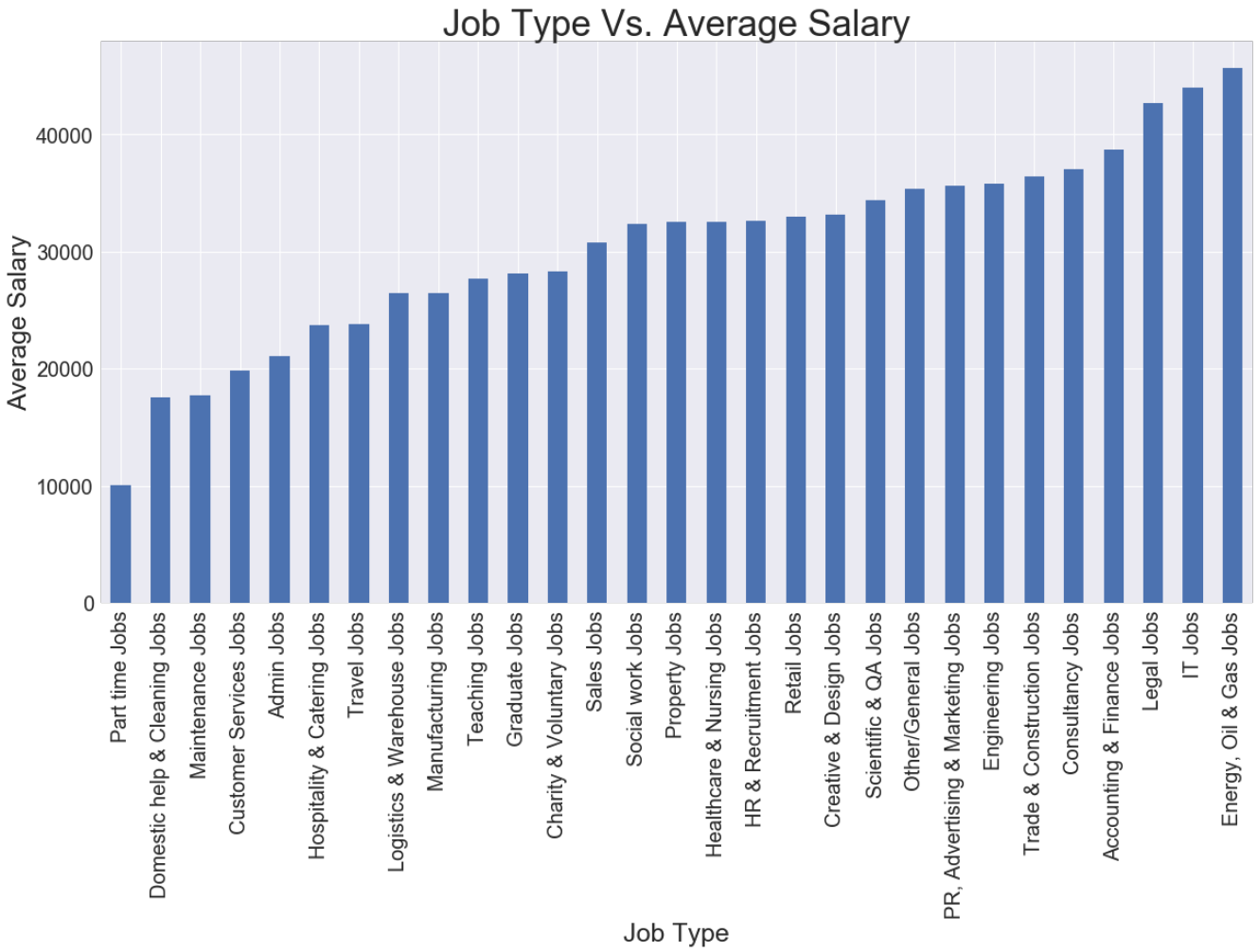


From the graph, distribution is observed to be right skewed and most of the job postings are centered around 30,000. This corresponds to the mean and median of the normalized salary, but without the graph, it would be hard to know that the distribution is right skewed.

For the next analysis, a graph for the ‘Category’ column is plotted. Using pandas library’s groupby method, the counts of the most common industries are plotted in ascending order.

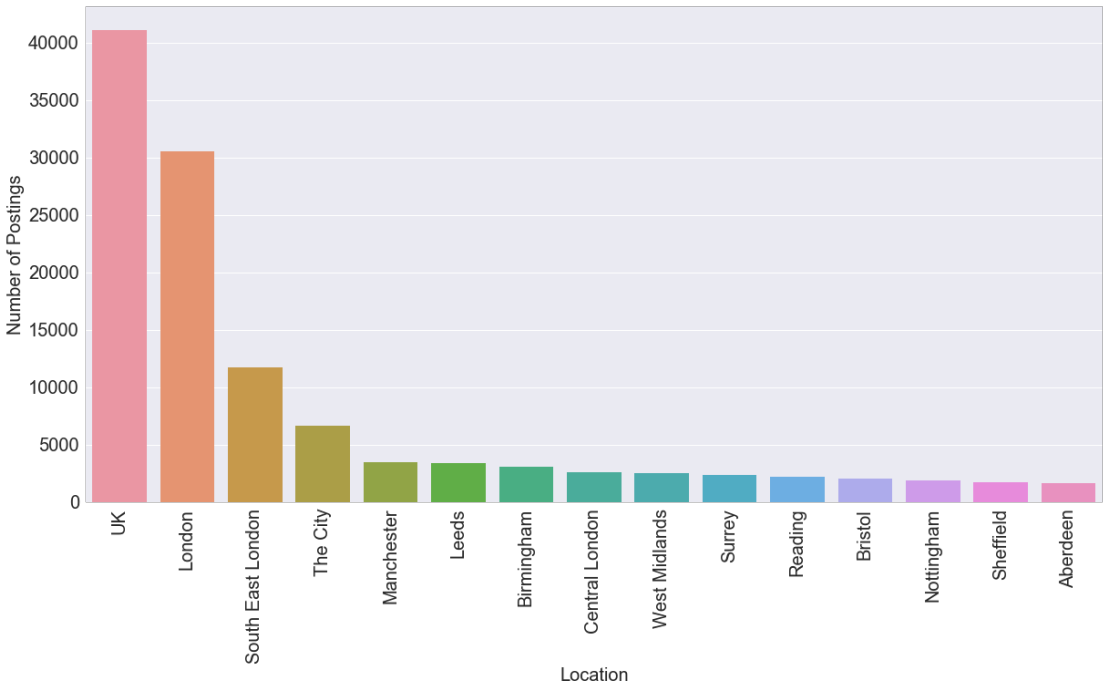


Out of all the job postings, IT and Engineering jobs are the most abundant while part time jobs are the least in quantity. The low abundance in part time jobs could be due to the reason that most part time jobs are not offered on job posting sites. While the high supply of IT and engineering could be due to the high qualifications for such jobs. To find out the relationship between ‘Category’ and ‘SalaryNormalized’, another groupby graph could be constructed.

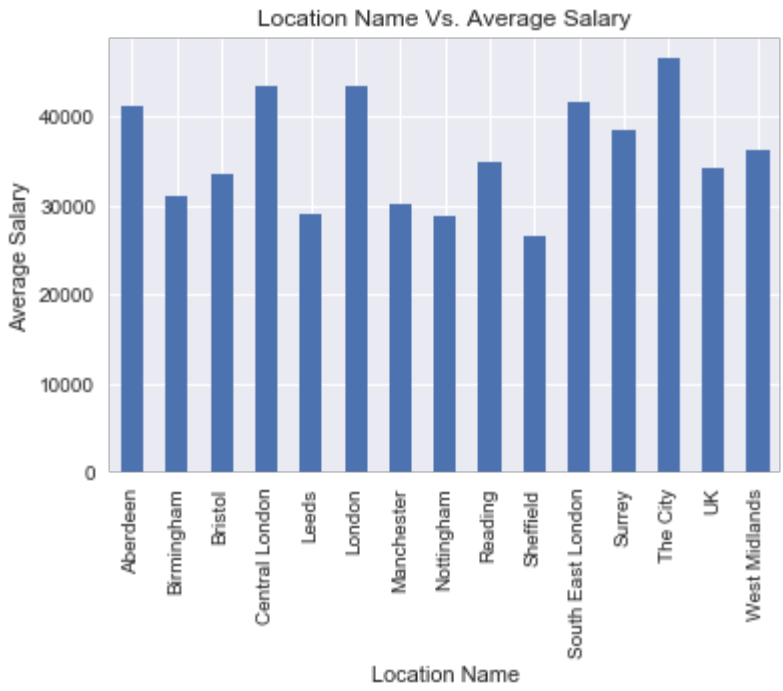


Average salary is on the y-axis and the job type is on the x-axis. Part time job is again on the bottom of the list, confirming the assumption that part time jobs usually have lower skill cap/qualification. Surprisingly, 'Energy, Oil & Gas Jobs' is at the top of the list. This is due From this analysis, a conclusion could be drawn that salary is partially determined by the supply of the job as well as the qualification of the job.

Using the same method, another analysis is done on 'LocationNormalized' and its relationship with salary. A bar graph is constructed with the top 15 most posted locations, and then another bar graph is constructed to visualize the relationship between those 15 locations and its salary.

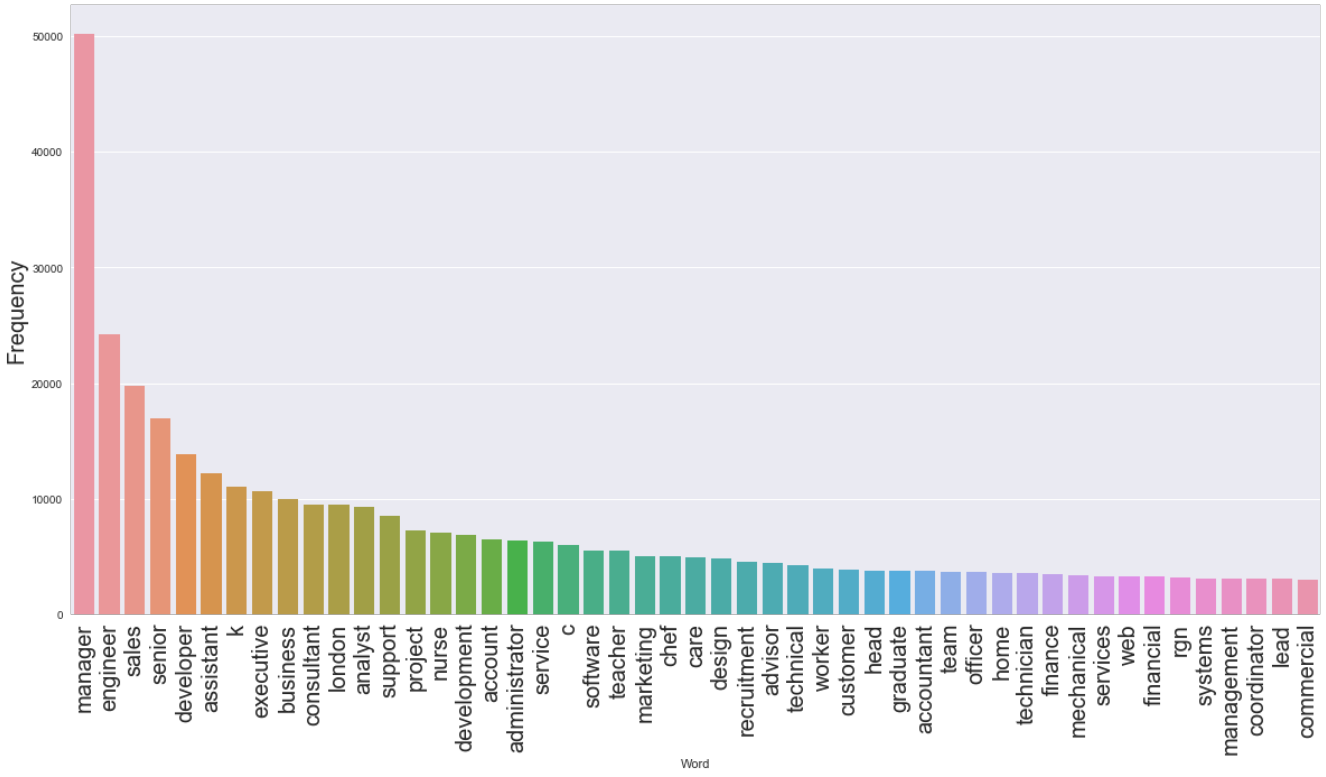


This graph has UK, London, and South East London as the top three most popular posting locations. Just from the glance of it, a relatively safe assumption could be made that the location column will not be very informative in predicting salary. London is within UK, and South East London is not a city, but a region of London. This shows poor information by not having a standardized location description.

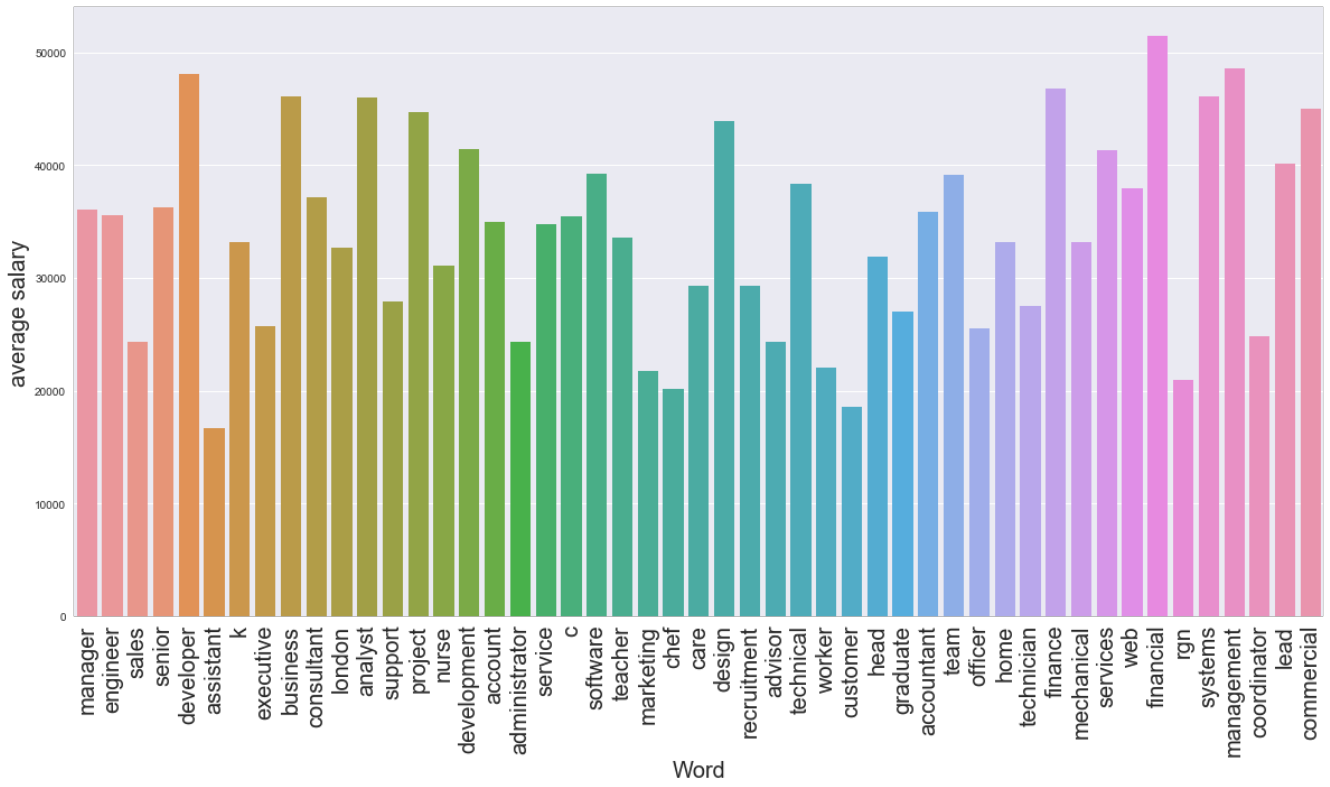


The x-axis label order is preserved in both graphs and no relationship between average salary and location name is observed. This analysis further confirms that location column is a poor predictive variable in salary, due to over generalization and poor standardization.

Next, ‘Title’ column is examined. This time, nltk is used to remove stopwords and punctuation. Then ‘Counter’ method from ‘collections library is used to count the most frequent words in the job title.



From the dataframe constructed from the top jobs and the frequency of the words, a visualization of the top 50 most common words are plotted in descending order. Modifying the dataframe more, another column is added for the average salary of the job titles containing each of the top 50 words.

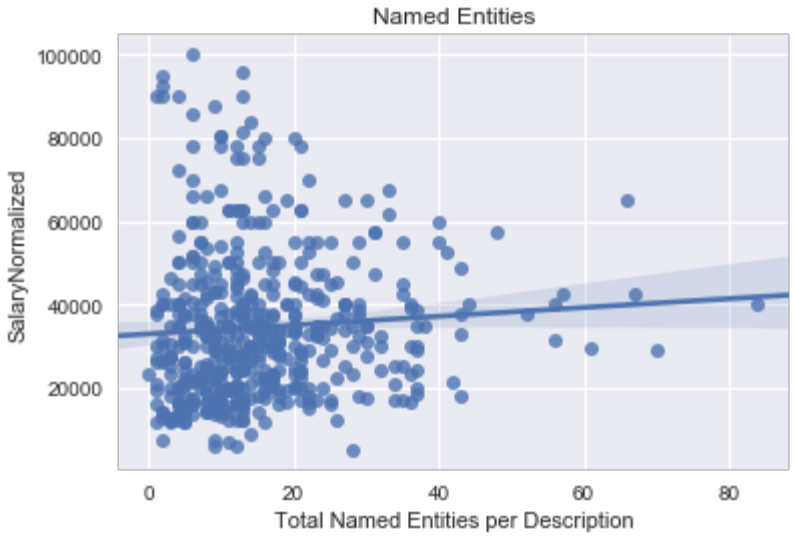


In order to view the relationship between words' frequency and average salary, the top 50 words are kept in their original order. There is no observed relationship between the frequency of the word and average salary. The most frequent word (manager) and the least frequent word (commercial) have similar average salary, and average salary throughout the distribution fluctuates greatly. From this analysis, a conclusion could be made that the words in the title description do not have a strong impact on salary.

For the next graph, the relationship of resume length vs. salary is plotted. This is done using spaCy library, which is a natural language processing tool similar to NLTK. Each 'FullDescription' row is tokenized and then word count is totaled for each row.

The distribution for this graph is scattered with no observable relationship. Looking at the best fit line, it can be inferred that content length has a slightly positive relationship with the salary. But this relationship is marginal, therefore the impact on accuracy score is going to be minimal.

Another graph is made to visualize the number of named entities, which includes names, organization, etc… but excludes punctuation and filler words such as pronouns and verbs.



Again, this graph does not have observable pattern. Therefore, the number of named entities will have relatively low impact on salary prediction.

Justin Zhang

Modeling

In order to execute supervised machine learning on text data, a corpus is constructed from the training data which includes all the words from each document. The documents consist of the aggregation of strings in each column of the row.

Tfidf is used for this project since it is different from countvectorizer, because countvectorizer gives the equal weight to to all the words while tfidf calculates how important each word is to a specific document in respect to the corpus. For example, a word appearing in every document is going to be less important compared to a word appearing in a few documents only. This weighing technique tells the machine learning model during classification as it is told which word weighs more or less.

First, to aggregate the text in ‘Title’, ‘FullDescription, ‘LocationNormalized’, and ‘Category’ column, all the strings are added together and stored as X for predictive variables. The target variable is y, which is the ‘SalaryNormalized’ column. Using train-test-split, X and y are split into 75% testing and 25% training set.

Importing TfidfVectorizer from sklearn.feature_extraction, TfidfVectorizer is instantiated with no parameter. Runninng fit-transform on xtrain lets the vectorizer to learn the vocabulary in xtrain and then return a term-document matrix, saving to tfidf_train. Then transform is performed on xtest to return a term-document matrix on the testing data using the corpus generated from the training data, saving to tfidf_test.

After fit-transform of the dataset using tfidf, the data is ready for machine learning models. For the first model, Multinomial naive bayes from sklearn.naive_bayes is used. This is a simple and most commonly used model for natural language processing due to its speed and predictive power. After instantiating the model, to fit the mode tfidf_train and ytrain are used as the variables. tfidf_train is

used instead of xtrain, because machine learning models can only take numbers as arguments. For multinomial naive bayes without parameters, a training score of 81.27% and testing score of 80.42% is achieved. This result means that, if the model is run on real world data, the model performs 30.42% better than random guessing, since this is a binary classification problem.

Tfidf has many parameters to tune, a very useful parameter is the stop_words parameter. Setting it to “english”, this parameter removes all the pronouns and verbs that offer little to no value in helping predict the target variable. By removing the stopwords, multinomial naive bayes is able to improve training accuracy to 81.41% and testing accuracy to 80.56%. This is a marginal increase in accuracy, and it is due to tfidfvectorizer’s built in parameter of max_df=1, which eliminates words that appear in every document.

Next up is tuning the ngram_range parameter of the tfidf_vectorizer. The ngram_range parameter takes in the lower and upper boundary of n-values for different n-grams to be extracted. For ngram=1 is unigram, which takes each token one at a time while ngram=2 takes in two tokens at once. The best ngram_range parameter is ngram_range=(2,3), and it boosted training accuracy to 92.07% and testing accuracy to 86.42%. ngram_range=(3,4) actually gave the best accuracy score, but it took exponentially longer to run than ngram_range=(2,3), and the increase in score is not significant.

MultinomialNB is a fast and simple model, but a more complicated model such as random forest could yield better result due to its ability to handle over-fit models. With no parameter in TfidfVectorizer, random forest increased training accuracy 99.45% and testing accuracy 84.41%. It is normal to have an over-fit training model for random forest, but the testing accuracy increased by 4% just by using a different model.

To incorporate some parameters into the TfidfVectorizer, stopwords are eliminated and max_df is decreased to 0.7, meaning tokens appearing in more than 70% of the documents are eliminated. With this parameter, only the testing accuracy increased, but only marginally by 0.25%. Just like

MultinomialNB, RandomForestClassifier's accuracy did not increase significantly from the stopwords and max_df parameter.

For the last model, LinearSVC from sklearn.svm is used. With empty parameter in TfidfVectorizer and LinearSVC, training accuracy is 92.26% and testing accuracy is 87.63%. This is highest testing accuracy out of the three models when it is run without parameters. There are multiple methods for support vector machine, LinearSVC is chosen because it scales better with large sample while its counterpart, SVC, have a harder to scale samples over 10,000. This dataset has over 180,000 samples.

Adding in stopwords= 'english', max_df=0.7, and ngram_range=(2,3), LinearSVC achieved a training score of 99.56% and testing score of 91.34%. This is the highest score achieved out of the three models with parameter implementation. To fine tune the model further, GridSearchCV from sklearn.model_selection is used. There is only one necessary parameter to tune for LinearSVC, the C parameter, which is the regularization parameter that controls the trade-off between misclassification and width of the margin. Through GridSeachCV, the best parameter for C is 3. Incorporating C parameter into LinearSVC and the rest into TfidfVectorizer, a training score of 99.83% and testing score of 91.45.

During exploratory data analysis, it was assumed that 'Title' and 'LocationNormalized' columns did not affect the target variable significantly. Therefore, at this step, the final model is run without the two columns as the predictive variable. With solely 'FullDescription' and 'Category' as the predictive variables and all the final parameters incorporated, the training accuracy is 99.74% and testing accuracy is 91.06%. Removing the two predictive variable actually hurt accuracy score, but only slightly. Even though 'Title' and 'LocationNormalized' had a weak relationship with 'SalaryNormalized', the relationship is still positive. The amount of features added from the two

columns is minimum compared to the full description of the posting, so over-fitting did not become an issue.

Conclusion

	MultinomialNB	Random Forest	LinearSVC
No parameter training	80.42%	99.42%	92.26%
No parameter testing	81.27%	83.55%	87.63%
No stopwords/ max_df=0.7 training	81.42%	99.42%	92.45%
No stopwords/ max_df=0.7 testing	80.55%	84.69%	87.61
With ngram training	92.07%	99.27%	99.56%
With ngram testing	86.42%	84.34%	91.34%
All parameter + GridSearchCV: C=3 training			99.83%
All parameter+ GridSearchCV: C=3 testing			91.45%

Justin Zhang

The best model out of the three is LinearSVC. It is a very fast model and is able to handle large sample of data. With combination of model selection and parameter tuning, I was able to increase from 80.42% to 91.45%, a total of 11.03% increase in testing accuracy. When tuning the model, removing stopwords and setting limit for max_df only helped the accuracy marginally while ngram_range was able to increase accuracy score significantly. This is due to TfidfVectorizer grouping words together to derive meaning from the word phrases instead of making prediction based off each individual word's frequency.

For this problem, it is divided into a binary classification problem for interpret-ability reasons, but it could easily be divided into a three category classification problem for more practical applications. For example, dividing it into range 30,000-60,000, 60,000+, and 30,000-.

This project proved that simply from the description of the job posting, it is very feasible to get an accurate estimate of salary. The model could prove useful for website like indeed to provide salary estimate on job postings.