ASSIGNMENT 2

KUMAR GAURAV

ROLL NUMBER
10010821

SALARY PREDICTION

I used Python v2.7 platform for this assignment. Based on the size of data set and the amount of computation required, python seemed to be the better choice.

- **FINAL APPROACH:**

  Feature Extraction: **TIME TAKEN :less than 200 seconds**

  - For **time and term** features, we needed to fill the missing the data using the description. I filled as much as the information available in description. I represented time and term by 1,0 and -1

Time feature	Term feature
Full time : 1	Permanent : 0
Part time :-1	Contract : 1
Empty : 0	Empty : -1

    I tried to give 0 to that feature value which occurred most number of times, so that I have least non-zero elements to be filled in the sparse matrix in later stage.
  - For **category** and **source,** I have extracted unique category and source. I have given one hot bit representation.
  - For **location** and **company**, I replaced them with their median, maximum and minimum salaries. For example, in location section I replaced London by its median salary, maximum salary and minimum salary.

    For the company section, I replaced the empty block with average median, average maximum and average minimum of all the existing companies median, maximum and minimum salaries

    Abandoned Approach: There were missing data in companies section. First I tried to fill them, but there were a lots of company names which were common English words like review, become, team, solution, the candidate etc. I created a stop word list based on these words to remove them but after that the number of missing companies found were to less as compared to total data set size. So, I abandoned this approach.

    For empty company, I replaced it with average of median, average of maximum, average of median and average of minimum salaries of existing companies.
  - For **Title** feature, I calculated the tf-idf values of the words in the titles. Initially it took time to compute, but I eventually optimised by using a different structure of dictionary. I created a big dictionary in which each key represented a word and a value represented by the dictionary which contained the key as document number and value as term frequency. For example,

    In dict={word1:{4:2},word2{3:1,2:1}}, word1 has a term frequency 2 in document 4 and word2 has a term frequency 1 in document 3 and 1 in document 2

    Then using one more dictionary and a list, I calculated the tf-idf values of each word in each title.
  - Similarly for **description,** I calculated the tf-idf values of each word.
  - Based on the idf values of words, I took top (high to low idf) 5000 words from title and similarly 5000 words from description. I did this because words with high idf values are the ones which are more important.
  - I used **sparse matrix** representation for storing the matrix formed and saved in .pkl format using cPickle.

Though I did the above steps on 1.5 lakh data set, time taken was around 190-200 seconds.

Training:

- I trained the data set using Random Forest and SVR.
- For **Random Forest**, after some manual testing on small data set, I selected the parameter values which gave almost best results. I took number of trees=50, min_samples_split=30, n_jobs=1.
- For **SVR**, I took C=1e3, epsilon=0.2, kernel='linear'. For textual data, linear kernel gives good results.

  MEMORY PROBLEM: When I tried to train using **random forest** on data set larger than 5000, it showed memory error. So, I had to divide the whole data into chunks of 5000 and trained 28 Random forest models for data set of 1.4 lakh.

  For **SVR**, I had to divide the data into chunks of 2500. As I increased the size the time increased almost exponentially. For 1200, it took around 12 minutes. I trained it on 84,000 samples which gave me 70 models.

Testing:

- Before using the model, I had to do the feature extraction on Test.csv.
- I filled the **missing time, term values** from description in Test.csv
- I calculated the tf values of only those words in **title and description** which were in top 5000 of sorted list from title and top 5000 of sorted list from description of training set based on idf values. I calculated tf-idf of these words using their tf from test data and idf from training data. If I had taken idf from test data, it won't have given much of an information and also, idf varies with number of samples in test data.
- I used the **unique categories and sources** list made during feature extraction for training, for building the test sparse matrix.
- For **location and company**, I used the same technique as in training. I replaced them with their median, maximum and minimum salaries. For a new location or new company or missing company, I replaced them with the average of median, average of maximum and average of minimum salaries of existing companies and locations.
- I created a **sparse matrix** same as in training and saved it in .pkl format.
- **Testing with Random Forest:** I tested on a small data set of 100 samples. It gave a root mean square error of around 11765.87 dollars and mean absolute error of 8388.87 dollars.

  For a data set of 6800 samples, root mean square error was 12602.62 dollars and mean absolute error was 8414.15 dollars.

  Problem: When I tried to test it on large data set, a memory error was prompted. So, I divided the test data set into chunks of 4000.
- **Testing with SVR**: I tested it on a small data set of 100 samples. It gave me a root mean square error of 26749.7 and mean absolute error of 18250.77. For a data set of 6800 samples, the mean square error was 24839.83 and mean absolute error was 18752.31.

  Problem: When I tried to test it on large data set, it went very slow. So, I had to divide the test data into chunks of 1000 samples.

LEARNING:

- When working with large data set, we got to divide it into small chunks and take the mean of all the small models.
- The results obtained clearly reflected that Random Forest performs far better than SVR both in time complexity and error. For training the models, the time taken by random Forest was very less as compared to SVR. The MSE obtained from random Forest is quite low as compared to MSE from SVR.
- We need to select the best/optimum parameters in SVR model and Random Forest Model to get best results i.e. minimum error. We could have used Grid Searching in Python to get the best parameters. I did it manually by changing the value of parameters based on the knowledge of parameters.
- The **problems** faced during feature extraction, training and testing have been mentioned above at the time of describing about feature extraction, training and testing.

OPTIMIZATION: **BONUS MARKS**

- Memoization: I used the concept of memorization where ever possible. I tried to feature extraction in just one iteration over the training data/test data.
- The tf-idf calculation for 1.5lakh samples was done in less than 140 seconds, which took hours and hours for other students. Similarly, the rest of the code also ran quite fast.
- For tf-idf calculation, I used a better dictionary of lists data structure in python. This dictionary contained words as its keys and the value of these keys was another dictionary which contained document number and term frequency of the word. For example,
  In dict={word1:{4:2},word2{3:1,2:1}}, word1 has a term frequency 2 in document 4 and word2 has a term frequency 1 in document 3 and 1 in document 2

- I used same structure of lists in dictionaries as described above, for company, location, category, source, time and term ,which was used for making row vector, column vector and data vector to be used for making coo_matrix(a type of sparse matrix). This really fastened the process.
- For making the large matrix which contained words as columns and samples as rows, I used sparse matrix using sklearn in python. For saving this matrix, I used cPickle with HIGHEST_PROTOCOL, which really accelerated the saving process and also reduced the size of saved file. For doing all the feature extraction and saving it in sparse matrix took around **170-180 seconds**.

SUBMISSION: I have submitted the python files, report, read me, sparse matrix and required csv files via moodle. Trained Models have been submitted by dropbox(https://www.dropbox.com/sh/tr6j1lm8nbsyj85/DZK7gDDfKY)