**Questions**

Please **briefly** answer the following questions. Try to be as concise as possible while still giving complete answers – you will have the opportunity to add more detail and explanation regarding your approach via a follow-up discussion.

1. How long did it take you to solve the problem?

It took me 2 weeks to solve the problem with 2 hours a day.

1. What software language and libraries did you use to solve the problem?

I used python3 and its libraries, NumPy, SciPy, Pandas, xgboost, scikit-learn, and visualization tools (matplotlib and Seaborn).

1. What steps did you take to prepare the data for the project? Was any cleaning necessary?

I had to convert features from “*Object*” to “*Category*” type first. I removed very few rows in the data which were missing target variable, “salary” upon which all training was supposed to take place. Otherwise, there was no NAN or missing data. I converted all categorical features into numerical values using *onehot* and *label* encoding. Specifically, “industry,” “major,” “companyId,” and “jobId” were non-ordinal (and hence onehot encoded) while “degree” and “jobType” were considered ordinal (and hence label encoded). Since all 1,000,000 “jobId”s were unique and “companyId” was irrelevant in terms of salary prediction, I removed them both. I converted long “Int64” datatypes into short “Int8” and “Int16” if possible, to reduce memory use. Given that the goal of the problem was to investigate the salary for entry-level vs. senior-level data science roles, I decided to create 4 categoricals depending on numerical values of “yearsExperience” feature. [0, 2] was considered entry-level while [5, 9] as senior level.

1. What algorithmic method did you apply? Why? What other methods did you consider?

I applied three high-score algorithms: (Linear)*Ridge*, *GradientBoostingRegressor*, and *XGBRegressor*. These three algorithms produced the *least mean squared error*, MSE as our metric of the problem among 8 of those. Other algorithms I considered were, *Lasso, RandomForestRegressor, DecisionTreeRegressor ,LinearRegressor, and KNeighborsRegressor.*

1. Describe how the algorithmic method that you chose works?

After tuning hyper-parameters of the 3 best algorithms, I decided to choose *XGBoost* which has the lowest MSE for both entry-level and senior-level data science roles. The reason I chose *XGBRegressor* over *GradientBoostingRegressor* was that our “train” data was a large-scale problem with 1,000,000 unique data points whose features were mostly categoricals. It seems that *xgboost* package and its Python interface is faster than scikit-learn implementation of normal gradient boosting. *xgboost* is simply an ensemble method that combines many decision trees to create a strong model out of many weak learners. It works by building trees in a serial manner, where each individual tree corrects the error of the previous one. In the absence of any randomization of this methodology, a strong parameter tuning is utilized; this model is sensitive to hyper-parameters. It uses shallow trees which makes it smaller in memory and faster in prediction. This model has a hyper-parameter called “learning-rate” which controls the strength of each tree in correcting the errors of previous ones; for complex models, a higher “rate” is advised. Another hyper-parameter is “n\_estimators” which is the number of trees that controls the same strength mentioned above; higher number corresponds to more complex problems. This model would have worked better if my data was not as sparse as it is given considerable number of categorical features.

1. What features did you use? Why?

I used all features in the original “train” data except “jobId” and “companyId” as they were completely irrelevant after investigating the importance of features.

1. How did you train your model? During training, what issues concerned you?

Instead of implementing a simple grid search over the parameters of each model, training and evaluating a regressor for each combination to assess how good the model is, and to avoid information “leaking,” I split the data into three sets: the *training* set to build the model, the *validation* set to select the parameters of the model, and the *test* set to evaluate the performance of the selected parameters. Specifically, I trained *XGBRegressor* algorithm on 80% of “train” data in order to score the 20% “validation” data. For a better estimate of the generalization performance, instead of using a single split into training and a validation set, I used *cross-validation* to evaluate the performance of each parameter combination using the *GridSearchCV* class which implements the methodology in the form of an estimator. Fitting the *GridSearchCV* object not only searches for the best parameters, but also automatically fits a new model on the whole training dataset with the parameters that yield the best cross-validation performance. Specifically, with the best hyper-parameters (base\_score=0.5, booster='gbtree', gamma=0, learning\_rate=0.2, max\_depth=6, n\_estimators=100, reg\_alpha=0, reg\_lambda=1, tree\_method='exact'), I made sure that I am hitting the required MSE for entry-level and senior-level data science roles among all “training” dataset by training the same model over entire “train” dataset. While it wasn’t difficult to hit the required accuracy on the 4 distinct subsets of “train” dataset (entry, junior, senior, and principle), it was challenging to get the same high accuracy on the entire dataset all at once where “yearsExperience” is encoded into 4 ordinal distinct values.

1. How did you assess the accuracy of your predictions? Why did you choose that method? Would you consider any alternative approaches for assessing accuracy?

While for any regression model, we have a plethora of evaluation metrics, e.g. *mean squared error* (MSE), *root mean squared error* (RMSE), *mean absolute error* (MAE), and *coefficient of determination* (R^2), I considered *means squared error* (MSE) for assessing the accuracy of my prediction as requested by the program administrators. While in general R^2 is a more intuitive metric to evaluate regression models, the business decisions in this problem was made based on MSE giving incentive to tune my model using this.

1. Which features had the greatest impact on salary? How did you identify these to be most significant? Which features had the least impact on salary? How did you identify these?

In a descending order, “industry,” “major,” “jobType,” “yearsExperience,” and “degree” have the greatest impact on salary. “milesFromMetropolis” and “companyId” have the least impact on salary. I identified this by applying a feature importance on the combined “train” data in which “industry” and “major” features were added to the engineered data with encoded features. “companyId” was removed right from the beginning just after checking its overall statistical distribution which seems to be relatively uniform. However, I kept “milesFromMetropolis” conservatively as I assumed (which turned out to be correct to some extent) that employees who are paid higher prefer to stay close to their job location.

1. Please explain any additional work that you did as part of this project.

For the training dataset, I made visualizations of salary distribution in the form of boxplots (using *Seaborn* python package) marginalized over any one of the features, i.e. “major,” “degree,” “industry,” “jobType,” “yearsExperience,” and “milesFromMetropolis”. I also plotted the correlation matrix among numerical and ordinal categorical features in order to find any mutual correlation between any two of such features to justify the features used and engineered previously.