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

Ans: Approximately 3 hours

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

Ans: I have used Python and the libraries that I have used are : Sklearn, Pandas, numpy

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

**DATA PREPARATION:**

I have uploaded the two datasets: “train\_salaries\_2013-03-07.csv” and “train\_features\_2013-03-07.csv” into two pandas Data frames: “data\_train\_sal” and “data\_train\_ftr” respectively.

**Investigating the****“data\_train\_ftr” dataframe:**

The jobId column is a unique id.

Looking at the data I found that there are many 'NONE' values for 'degree' and 'major' column. I am sure most of the jobs require degree. So thought of deleting all records for which 'degree' and 'major' columns are NONE where the jobTypes are: 'SENIOR','VICE PRESIDENT','JUNIOR','CTO','CEO','CFO','MANAGER' except 'JANITOR'. The ‘JANITOR’ jobtype may not require a degree or major.

Also thought of dropping all records for which degree - 'MASTERS', 'BACHELORS', 'DOCTORAL' and major is 'NONE". There cannot be Masters, Bachelors or Doctoral degree without a major.

**Investigating the “data\_train\_sal” dataframe:**

The ‘jobid’ column is also a unique attribute in this dataframe. This dataframe is clean. There are no ‘NONE’ values either for ‘jobId’ or ‘Salary’ column.

**Investigating the** **“test\_features\_2013-03-07.csv” file:**

After investigating this file, I found out that the “degree” and “major” columns also have None values. But the pattern of None values is very similar to the “train\_features\_2013-03-07.csv” file. There are ‘NONE’ values for ‘degree’ and ‘major’ where jobTypes are: 'SENIOR','VICE PRESIDENT','JUNIOR','CTO','CEO','CFO','MANAGER' and 'JANITOR'.

**Decision taken:**

I decided not to delete records for "data\_train\_ftr" dataframe/ "train\_features\_2013-03-07.csv" file for which the values for "degree" and "major" are None. If I delete these records and train a model, my model will not be able predict the salaries for new records with "NONE" values for 'degree' and 'major'. As both the "train\_features\_2013-03-07.csv" and "test\_features\_2013-03-07.csv" files have 'NONE' values for 'degree' and 'major' columns.

**Merging the two (train\_features and train\_salaries) datasets and investigating the merged dataset:**

data\_features\_salary = data\_train\_ftr.merge(data\_train\_sal, left\_on='jobId', right\_on='jobId', how='inner')

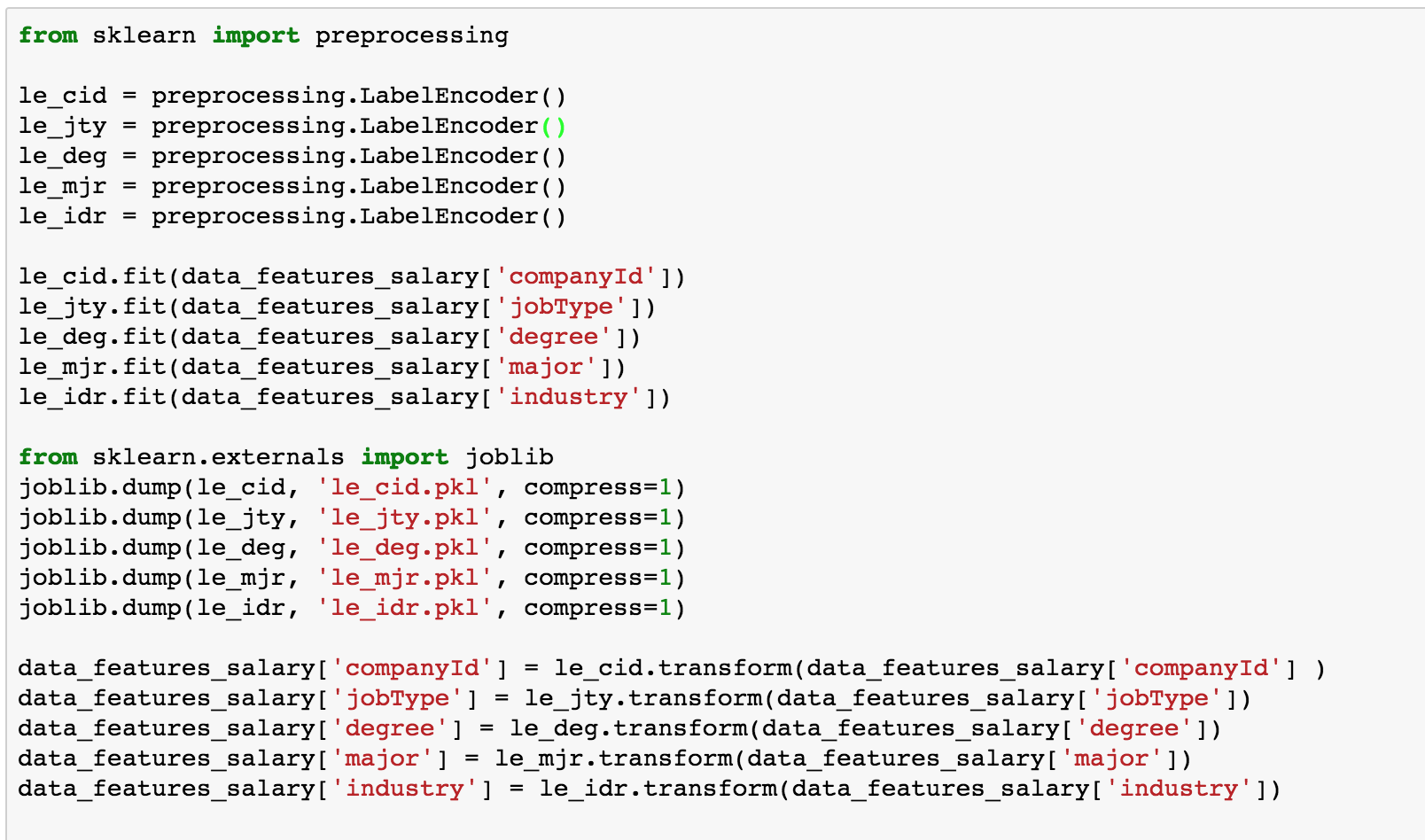
As highlighted in yellow, I have joined the “data\_train\_ftr” and “data\_train\_sal” into a new dataframe “data\_features\_salary”. While joining I did an inner join on “jobid “ as jobid is the unique id for both the dataframes. “data\_features\_salary” dataframe has an extra “salary” attribute now.

**Dropping the “Jobid” column:**

The jobid column is just an unique identifier. It is not categorical or numerical feature from which a machine learning algorithm can learn anything from. I have decided to drop this column from the dataframe.

**Categorical to numerical features***:*

The following attributes are categorical : “companyId”, “jobType”, “degree”, “major” and “industry” are categorical . We must convert them to numerical for machine learning algorithm to learn and build a supervised learning algorithm. I have used “sklearn.preprocessing” “LabelEncoder”. This Encode labels with value between 0 and n\_classes-1. Here, “n\_classes” is the number of categorical in the categorical feature. Code snippet highlighted in yellow below:

**

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

I thought of training a "RANDOM FOREST" or a "GRADIENT BOOSTING" Regressor. The reason I thought of using a Decision tree based bagging or boosting model because : a) Most of the features in the dataset are categorical and categorical features(nominal data) do not have linear interactions. b) The two continous variables("yearsExperience" & "milesFromMetropolis") do not interact linearly with the target variable("salary") as well (this was clear from the scatter plots). So, overall a linear model like a multi variate linear regression or "Support Vector Machine with a linear Kernel" would not suffice. I also thought of try "Suppor Vector Machine with a Non-linear Kernal such as RBF". I will also try a multi-layer neural network model – “MULTILAYER PERCEPTRON REGRESSOR” as a neural network model might be able learn the non-linear relationships among the features. Also, neural networks are good at learning from raw features than in traditional machine learning where it requires to do a bit of feature engineering or at least finding out the most significant features.

I ended up choosing the GB Regressor as it gave me the best score on the test dataset.

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

**Random Forest**: Random Forest is a bagging algorithm. Bagging stands for Bootstrap Aggregation.It can applied to any model but historically it was most often used with Decision trees. Intuitively, Bagging tends to reduce the variance of our model. I would like to explain in the following steps:

1. It creates new training sets by sampling the training set with replacement, so there will generally be duplicates with the training sets. This process is call bootstrapping.
2. We can bootstrap many times say 200. Now, a Decision Tree is trained on each of these boot strap
3. Now, use all these 200 Decision trees are used to make predictions. For, Regression, it simple averages out the results. For classification, it does majority voting

**Gradient Boosting (This is the model I final chosen and why I will discuss later):** Gradient takes a different approach compared to Random Forest. Random Forest trains uses deep decision and they get trained on the bootstrapped samples parallely. The goal is to reduce the variance by averaging the results. Gradient boosting starts usually with a not very deep DT, it could be a DT with just one split. Then its takes the error from the first DT and passes it to the second DT. The second tree models the error from the first tree and record and passes the error to the third DT so on and so forth. GB focusses on modelling the errors from the previous steps, this a sequential process compared to RF which is parallel. There are lots of evidences that GB outperforms RF, but GB could end up formulating a very complex model and there is a chance of overfitting. Usually, GB requires hyper-parameter tuning to reduce variance.

**Adaboost** : Adaboost and Gradient Boosting are very similar. but there is a little difference:

1. For Adaboost, during the sequential modelling process, it re-weighs the data points so that poorly model data points in the previous iteration gets higher weight in the current iteration. I don’t want to go the mathematics of Adaboost as this is not the model that I had ultimately chosen.
2. For Gradient boosting, it keeps redefining the prediction target to a residual between the ground truth and the overall predictor results. It uses Gradient Descent optimization, I will try to explain in terms of the squared error regression where it teaches a Model of the form **y^ = F(x)** by minimizing the loss function **1/2(y – F(x))^2** in every iteration. In GB, it starts with a very weak model say **Fz(x).** Then GB improves on **Fz(x)** in every iteration, by building a new model by adding an estimator (**h(x))** to provide a better model**:**

**Fz+1(x) = Fz(x) + h(x) = y(ground truth)**

**Or, h(x) = y – Fz(x)**

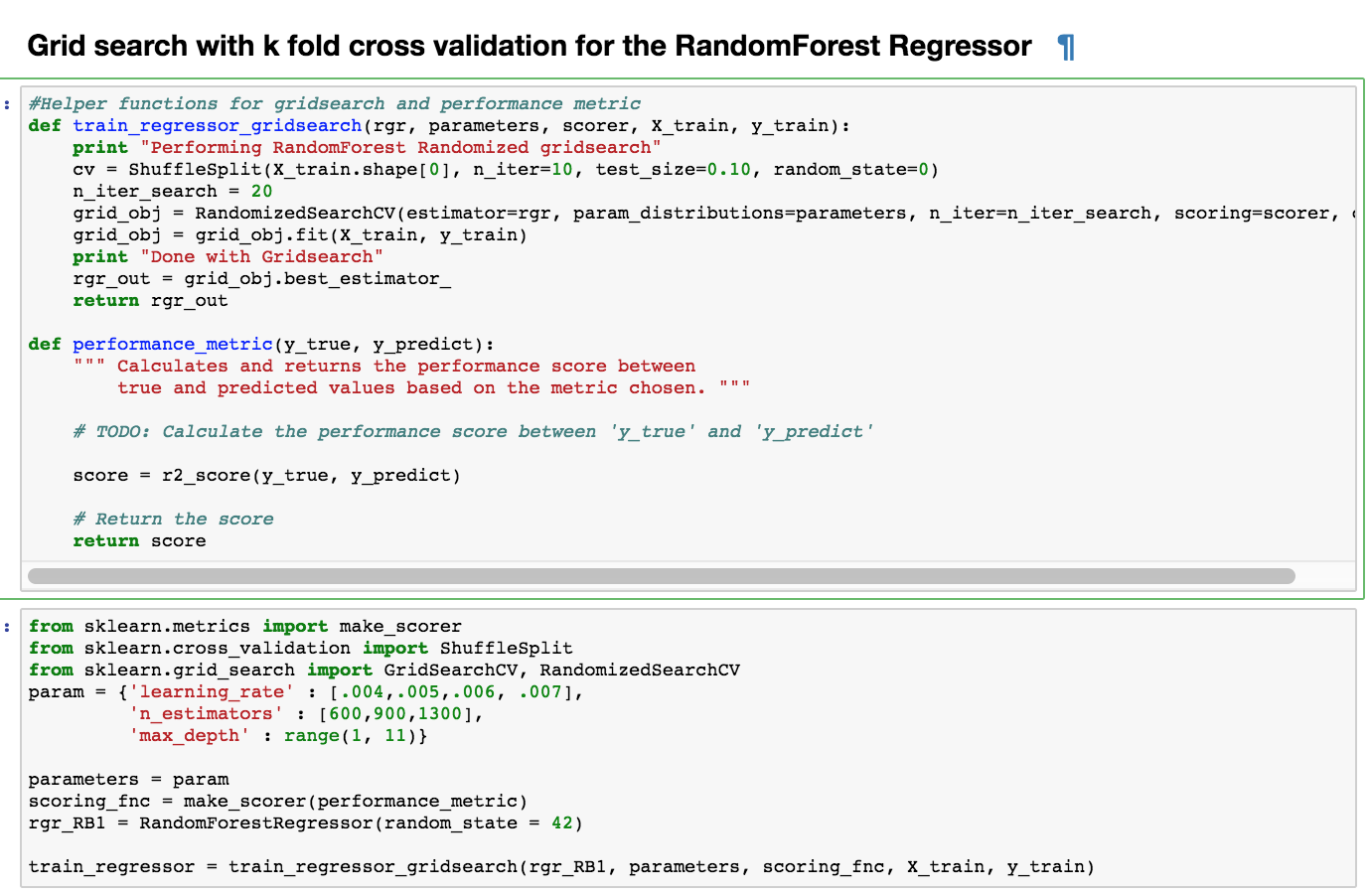
So, basically GB is trying to fit h(x) to the residual **y- Fz(x)** instead of the popular squared loss function

**Multilayer Perceptron Classifier:** This MLP is probably the only neural network based algorithm available within the sklearn API outside of Tensorflow, Keras etc**.** The reason I chose a Neural Network Classifier is because intuitively Neural networks are good with descriptive or non-linear features**.** Again, I also understand the constraint of training big networks in my local machine, I would have to tame myself my training a small network. Nevertheless, if I were to use GPU, I tend to believe that a big enough network with say 10 hidden layer each with 100s of neurons might give an accurate model. In my local machine, I found very comparable accuracy with my GB model and the MLP Regressor model.

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Parameters used** | **Train R2 Score** | **Test R2 Score** |
| **RandomForestRegressor** | **n\_estimators = 300** | **0.9619** | **0.7288** |
| **GradientBoosting Regressor** | **n\_estimators = 2000** | **0.7635** | **0.7645** |
| **adaboost Regressor** | **n\_estimators = 600** | **0.5101** | **0.5142** |
| **MLPRegressor** | **(hidden\_layer\_sizes = (300,150,70,20), max\_iter = 3000, learning\_rate = 'adaptive',learning\_rate\_init = 0.009,**  **tol = 1e-6)** | **0.7582** | **0.7595** |
| **MLPRegressor** | **(hidden\_layer\_sizes = (500,280,150,70,20), max\_iter = 4000, learning\_rate = 'adaptive',learning\_rate\_init = 0.009,tol = 1e-6)** | **0.7617** | **0.7632** |

**As we can see in the above table, The performance of GB Regressor is very much comparable to MLP regressor. And another interesting thing I have noticed for MLP is, its train and test score has improved little bit when I used a bigger network (500,280,150,70,20) vs (300,150,70,20). As I mentioned earlier, with dual core 16 GB CPU based machine I could hardly train a bigger network than these. But, I have a strong intuition, if I could train a big network ( 10 hidden layers** with 100s neuron each) I could end up with a much better model.

Among all the models, the GB Regressor has the best “R2 score” (rsquared) metric the test dataset : .7645. We could also see in the table that the Random Forest Regressor has the best score on the train dataset (.9619) but a test score of .7288. It’s most probably overfitting or not able to generalize well on the test dataset. Ideally I would have performed gridsearch on the RB model to find the most optimal hyper parameters with K-fold cross validation, but I could hardly do in my low memory/ CPU laptop. The time complexity for hyper-parameter tuning computations is in the polynomial orders :



The final model I chose is the GBRegressor model highlighted in blue in the table above.

6. **What features did you use? Why?**

I started with Univariate feature analysis. Except Jobid (just a unique ID). Jobid would no way impact the

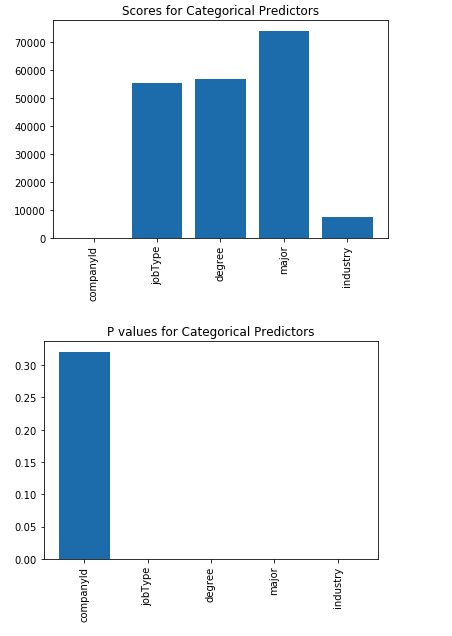
, we have five categorical features: **companyid, jobtype, degree, major and industry**

and two continuous features: **yearsExperience** and **milesFromMetropolis**.

From univariate perspective, I need to analyse how each feature co-relate to the target salary (continous feature)

I have used sklearn’s SelectKbest with f\_regression. F regression finds the correlation between each feature(regressor) and the target. And converts the correlation to an F-score based on ANOVA analysis (ANOVA- analysis of variance between a pair of Random variables.). And converts to a p-value. Higher F score and lower p-value means stronger co-relation of the regressor with the target. Specifically, the low **p-values** means strong significance.

Here’s what I found out:



From the above plot, the score for companyId is the least with very high value of P much above 0.5(95% confidence critical range). Others have pretty good score and close to zero P-value. These plots gave me evidence that all features other than “companyId” have good to ok co-relations with “Salary”. These features could significantly contribute to the changes in salary.

So, for RF, GB and Adaboost models I have dropped the feature “companyID”. Although, decision tree based models like RF and GB do not require to do univariate feature analysis. Univariate feature analysis are mostly used for linear models. I will also check the feature importance of my features after training the GB model (including the “companyid” in one case and excluding “companyid” in another) to check whether my univariate analysis are comparable to feature importance results.

**[ 0.06121651 0.12295385 0.05052422 0.28186786 0.20128709 0.13519009**

**0.14696039]**

**Index([u'companyId', u'jobType', u'degree', u'major', u'industry',**

**u'yearsExperience', u'milesFromMetropolis'],**

**dtype='object')**

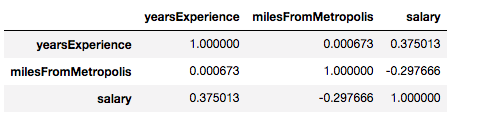
From the above, I could see the coefficient of importance for “CompanyID” is just 0.06 which is quite low compared to others. This is at least in-line with what I found out with univariate feature analysis.

I have also found that the performance of the GB models with or without “companyid” as feature are very comparable. The train / test R2 Scores for both the models are 0.76 and 0.76 respectively. So, overall, my analysis says feature set with or without “companyid” does not make much difference.

For MLP regressor, I used all the features (except JobId which is just a unique ID). The reason for that is Neural Networks are good at learning non-linear relationships from raw features. Feature analysis/ engineering etc. are not required for neural networks and that is one of the many reasons they are popular. They tend to find relationships on their own. NN tend work to work well when we have lots and lots of data points and lots and lots of features. In this case, we have 1 million data points and eight features, although **8\*1000000** is not a big enough matrix for a NN, still I would want to try out NN as most of my features are categorical and have non-linear relations among them and the target variable.

\*\*\* *Please note that I have a good background on neural network architectures. I could have discussed elaborately on neural networks ( such as weights, biases, nonlinear functions such as sigmoid, tanh, RELUSs. Regularization techniques such as L2, dropout etc, optimizing gradient descent with batching, RMS prop, adam optimization. etc.) But I didn’t not chose the MLP regressor as my final model and hence refrained from discussing neural network architecture in detail. Given a chance, I would love to discuss in a phone call*

For the two continuous features: **yearsExperience** and **milesFromMetropolis, I wanted to do find out the co-relations with the target “salary”:**



It looks like “yearsExperience” has some +ve co-relation with salary and “milesFromMetropolis” has some -ve co-relation with salary. I wanted to keep both of them in my feature set.

**7. How did you train your model?**

I performed a train test shuffle split on the dataframe for training (data\_features\_salary) – 90% allocated for training and 10% for testing. I would train the model on the 90% train set and check the accuracy on the 10% unseen test set.

During training, what issues concerned you? I wanted to do a 85-25 split on train test, but went ahead with 90-10 as I want maximize that data usage for training models. Another thing that concerned was imbalance in the categorical features between the train and test sets. So, I did test train shuffle split. The test\_train\_split module under sklearn.crossvalidation library automatically shuffles the data before splitting. And, maximizing the dataset for training (90-10) would also fortify the balance of features between the train and test splits. Another important concern was bias vs variance. Test-train provides great intuition whether a model is more bias than variant or more variant that bias. A model with pretty good score on the train set and but so good score on the test set could be a variant model, as it has learnt well from the training set but not able to generalize well on the test set (say 90% accuracy on train and 60% accuracy on test). A model with not so good score on the both the train and test set could be a biased model than variant. The model is not complex enough to learn from the train set (for e.g. 60% on accuracy on train and 60% on test).

8. **How did you assess the accuracy of your predictions?**

I used r2 score for accuracy. In statistics, the coefficient of determination, denoted as R2 or “R squared”, is a number that indicates the proportion of variance in the dependent variable that is predictable from the independent variable. The coefficient R^2 is defined as (1-u/v), where u is the regression sum of squares ((y\_true – y\_pred)\*\*2.sum() and v is the residual sum of squares ((y\_true-true.mean())\*\*2.sum(). Best possible score is 1.0 . A constant model that always predicts the expected value y, disregarding the input features has an r^2 score of 0.0

**Why did you choose that method?**

I wanted to bring in the perspective of minimizing the sum of squared error in a regression setup : (y\_true – y\_predicted)\*\*2.sum()

I could ideally check the sum of squared error for my model. Heuristically, lower the sum of squared error better the model. But, let’s think little deeper : model built on a **smaller dataset** **VS** a model built on **larger dataset**. The model built on a larger dataset would always have a larger sum of squared error compared to a model built on a smaller sum of squared error, although both the models could be fitting perfectly on the training dataset. Actually, this one of the shortcomings of sum of squared error as an evaluation metric, as we add more data the sum of the squared error would almost certainly go up but that does not necessarily mean that my fit doing a worse job. An evaluation metric that does not have this shortcoming is R^2 “r squared”. It effectively answers my question how much in the change in the output is explained by the change in my input. R^2 would always be between 0 and 1.

**Would you consider any alternative approaches for assessing accuracy?**

I could also consider Sum of squared error as an evaluation metric to compare the performance of the models. All the models are trained and tested on the same dataset. So, “sum of squared error” would also be a good comparison of performance although it has a big shortcoming as discussed above.

**9. Which features had the greatest impact on salary?**

The features importance from GBregressor:

**[ 0.06121651 0.12295385 0.05052422 0.28186786 0.20128709 0.13519009**

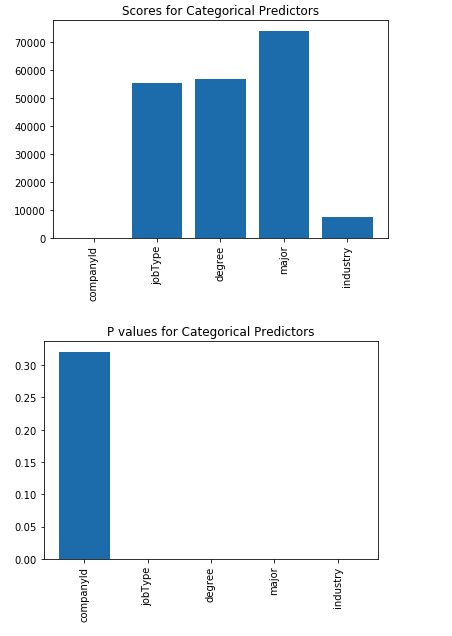
**0.14696039]**

**Index****([u'companyId', u'jobType', u'degree', u'major', u'industry',**

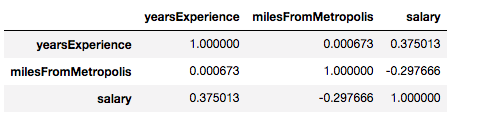
**u'yearsExperience', u'milesFromMetropolis'],**

**dtype='object')**

Feature scores from univariate feature analysis:



For the two continuous features: **yearsExperience** and **milesFromMetropolis,** I wanted to do find out the co-relations with the target **“salary”:**



It looks like “yearsExperience” has some +ve co-relation with salary and “milesFromMetropolis” has some -ve co-relation with salary. I wanted to keep both of them in my feature set.

Now, feature importance from GB and Univariate feature analysis gave me comparable results. From both of these analysis, Its evident that “**major”** and is most significant feature and would have the greatest impact on salary. The other significant features would be **“industry”,** **yearsExperience, milesFromMetropolis, job type, degree**

**[ 0.06121651 0.12295385 0.05052422 0.28186786 0.20128709 0.13519009**

**0.14696039]**

**Index([u'companyId', u'jobType', u'degree', u'major', u'industry',**

**u'yearsExperience', u'milesFromMetropolis'],**

**dtype='object')**

A very practical way to look at the greatest impact feature on the target is to recursively remove features and train the model and see the R2 squared value differences. After removing “major” from the feature vector, I trained the Gradient Boosting regressor model:

|  |  |  |
| --- | --- | --- |
| **FEATURE SET** | **R2 score on train dataset** | **R2 Score on test Dataset** |
| **['companyId', 'jobType', 'degree', 'major', 'industry',     'yearsExperience', 'milesFromMetropolis']** | **0.76** | **0.76** |
|  |  |  |
| **excluding companyid - [ 'jobType', 'degree', 'major', 'industry',     'yearsExperience', 'milesFromMetropolis']** | **0.76** | **0.76** |
|  |  |  |
| **excluding major - [ 'jobType', 'degree',  'industry',  'yearsExperience', 'milesFromMetropolis']** | **0.74** | **0.74** |
|  |  |  |
| **excluding industry - [ 'jobType',  'degree','major',  'yearsExperience', 'milesFromMetropolis']** | **0.66** | **0.67** |
| **excluding degree - [ 'jobType','major',  'industry','yearsExperience', 'milesFromMetropolis']** | **0.74** | **0.75** |

After R squared analysis, I would consider both “major” and “industry” to be most significant features

**How did you identify these to be most significant?**

I performed univariate feature analysis using K best features with F regression available with the sklearn library. I also looked at the feature importance vector from the trained gradient boosting model.

I also did R Squared metric analysis by training the GB model with and without this feature on the feature vector and found R2 metric value has the steepest decline without this feature.

**Which features had the least impact on salary? How did you identify these?**

From above ,it’s also evident that the “companyID” (trailed by “degree”) is the least significant feature and would have the least impact on salary. The F score on this feature is the least with a very high P value. The feature importance value from the GB model feature importance vector is also the least.

As per the table above, I have also experienced that with or without “Company\_ID” I have not any seen major difference in the R squared metric values for the GB model. Actually the R squared values on the train and test sets are almost similar with little differences in the 3rd and 4th decimal points

\*\*\* I have not considered “Job id” here as it is just a unique identifier and would have the least or no impact on the target.