Programmer Salary Prediction using Lasso Regression, Random Forest, Gradient Boosting

By CHAI JUNE RONG, CHEW HOO WING, CHIA WEE HANG

Faculty of Computing and Information Technology

Tunku Abdul Rahman University College

Malaysia

Abstract

Software developer or programmer has become a more and more popular choice for future employment ever since the software plays a significant role in our daily life. However, the factors that determine the salary of a programmer is often a question for most of the IT graduates and the answer for that is still something obscure and arguable. Hence, there is a pressing requirement to be able to predict the expected salary of a programmer.

In this research paper, three different machine learning algorithms, Lasso Regression, Random Forests and Gradient Boosting have been studied and applied to predict the programmer's salary based on the dataset of Stack Overflow Developer Survey, 2017 obtained from Kaggle online platform. All these three methods' background, strength, limitations, and their respective applications have been highlighted and discussed thoroughly in this paper. Their prediction results are compared, evaluated and there is an in-depth discussion on their respective performance and parameter tuning. This research also covers some other machine learning concepts such as Bag-Of-Words Models, feature extraction.

Keywords: PROGRAMMER SALARY PREDICTION; STACKOVERFLOW DEVELOPER SURVERY 2017; SKLEARN; CLASSIFICATION; SUPERVISED LEARNING; MACHINE LEARNING

I. INTRODUCTION

1.1 Project background

1.1.1 Existing theory

Generally, we have a strong prior belief about the amount of income of a programmer is largely depends on their skill level, personal capabilities and characteristics. However, some of the experienced programmers have a rather different view on this topic such that they are more willing to relate the salary level of a programmer to environmental factors such as company policy, local competition and stinginess (Mike Cellini,2011). Therefore, it is currently lack of an absolute metrics to determine the salary of a programmer. In addition, software development industry is a continuously upgrading industry with endless of new technologies emerging out every day and hence the prerequisite skills for a programmer is also constantly changing. Thus, it is always a big question for every fresh IT graduates to choose which specialized area of software industry they should devoted to and what related skills they need to master for getting a satisfying remuneration from their future employment. As an attempt to solve this problem, we would like to propose a solution that can assist in predicting the possible amount of salary of a programmer considering various information of different dimensions such as his/her mastered skills, personal preference, educational background, country etc. It was expected to resolve the question about the potential salary range in software industries for unemployed students, job seekers and anyone who seeks for a potential career change respectively. Apart from that, it could also be used by the currently employed programmers to evaluate if they are being paid fairly to a certain extent.

1.1.2 Similar system

At the beginning, the first idea that comes readily into our mind to solve this problem is to build an expert system or more specifically, a knowledge-based recommender system. It will suggest a range of predicted salary based on the information that users have entered. The prediction is based on a limited knowledge base which consists of typical programmers' behaviours, types of programming language, level of education etc. Then, a content-based filtering approach will be implemented to decide which ranges of salary should be returned to user as answer. This primary limitation of this system is the lack of massive data to support the decision-making process and thus the predicted salary result is untenable.

1.2 Problem statement

Ever since we entered the Fourth Industrial Revolution, software has undeniably become an indivisible part of human civilization and is expected to permeate more and more aspects of our life. Thus, the demands for software-related job positions particularly the programmers and developers has been climbing to its highest point and the remuneration provided is astonishingly attractive. Just as expected, hundreds of thousands of people regardless of fresh college graduates or the professionals from other industries have devoted themselves into this young and energetic industry. What makes us feel obscure is such that there is a huge diversity exists within salary range of programming job offers and here comes our problem statement which is how to predict the salary of a programmer based on his/her mastered skills, code style, preference, personal attribute, experience and various other information.

1.3 Objective

The task of this project is to create a programmer's salary prediction solution based on machine learning and text classification to meet the needs of anyone who seeks a reliable answer for that. The research results will be a prediction system that implements our model of machine learning algorithm to output the predicted salary range based on the information user input such as job description, skills mastered etc.

1.4 Advantages/contributions

By carrying out this project, we have realised that the transparency of salary, wages and remuneration provided in an industry is still fairly low even for the software industry which is well-known for its openness. This project could be of great use in helping unemployed IT students to compose their future career planning and employed programmers to assess the fairness of the remuneration they gained now. In the long run, this salary prediction solution can be applied to other occupations too such as accountants, engineers etc provided that the corresponding data set is available. All these are credited to the explosive growth of machine learning in the past few years.

II. LITERATURE REVIEW

2.1 Evaluation of similar applications or research

Section extract from Scikit-learn: Machine Learning in Python in Journal of Machine Learning Research 12 by Fabian Pedregosa et al (2011) [by CHIA WEE HANG]:

Scikit-learn is a Python module integrating a wide variety of state-of-the-art machine learning algorithms for both supervised and unsupervised problems. This package focuses on implementing machine learning algorithm using a general-purpose high-level language so the non-specialist outside the software industries can also perform statistical data analysis in a relatively easy way. Ease of use, performance, and consistency are the core values of this package (Pedregosa et al.,2011).

Section extract from Job Salary Prediction by Anonymous Author(s) (2013) [by CHEW HOO WING]:

The scikit-learn class CountVectorizer involves the converting of a collection of text documents into a dictionary of unique token counts. The dictionary constructed will be in a sparse matrix form. In the fitting procedure, once a token is encountered, that token's corresponding value in the document's feature vector will be incremented. Therefore, this can be used to extract unigram features from the respective entries of all the table fields (Job Salary Prediction, p.2).

Since the source of this section is rather unreliable considering its author(s)' anonymity, we have performed cross validation with the relevant technical documentation from scikit-learn official website which is available at <http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html>

Section extract from Text Classification by Augmenting Bag of Words (BOW) Representation with Co-occurrence Feature by Soumya George K[1], Shibily Joseph[2] (2014) [by CHAI JUNE RONG]:

In the area of natural language processing and information retrieval (IR), the Bag of Words(BOW) model is used as a text simplifying approach to represent a text as an unordered collection of its words, ignoring its grammar and word order. In this context, text classification will be carried out by assigning a weight to every word in a document based on its frequency present in its own document and between different documents. Thus, BOW will be formed from these words together with their own respective weight (George K[1] and Joseph[2], 2014).

Examples for forming a BOW:
Text document 1: "Charlene is a beautiful girl. Charlene is my sweetheart."
Text document 2: "Jason is my cousin."

Dictionary constructed:
{”Charlene”:2 ”is”:3 ”a”:1 ”beautiful”:1 ”girl”:1 ”my”:2 ”sweetheart”:1 ”Jason”:1 ”cousin”:1}

Each document represented as an 8-element vector:

Text document 1: [2,2,1,1,1,1,1,0,0] Text document 2: [0,1,0,0,0,1,0,1,1]

2.2 Technical Background of methods

I. Lasso Regression [by CHAI JUNE RONG]

‘Lasso’ a.k.a. ‘least absolute shrinkage and selection operator’ is a model proposed by Robert Tibshirani in 1996 for estimation in linear models. The emergence of this model is driven by the fact that ordinary least squares (OLS) estimates is not satisfying the data analyst because of its low prediction accuracy (low bias but large variance) and its deficiency in interpretation. Afterwards, there are two standards being proposed for enhancing the OLS estimates which is subnet selection and ridge regression. Subnet selection offers interpretable model but its drawbacks is such that regressors are either preserved or dropped from the model. So, it is a discrete process such that the model generated can be extremely variable. For ridge regression, it is more stable as it continuously shrinks coefficients but then it will not generate an easily interpretable model as it won’t produce any coefficient that is exactly 0. To resolve the drawbacks of both standards, lasso was born. It can produce some coefficients which are 0 while continuously shrinking the others and hence it combines the best of both world with the stability of the ridge regression and generates interpretable models just like the subnet selection. The lasso idea can be generally applied to a wide range of statistical models such as generalized regression models etc (Tibshirani, 1996).

II. Random Forests [by CHIA WEE HANG]

With the fundamentals of geometric feature selection, random subspace method and random split selection approach which are respectively proposed by Amit and Geman (1997), Ho (1998) and Dietterich (2000), Leo Breiman has come out with a new idea of classification and regression which is known as random forests in the year 2000. This scheme basically constructs a predictor ensembles of decision trees with each tree inside the ‘forest’ will grow according to a random parameter. Breiman has demonstrated the significant improvements in classification and regression accuracy of this method in a series of papers and it has been proven to be a strong rival to other methods like boosting and support vector machine. There are several strengths by using this method such as fast and easy in training, predictions generated are highly accurate and without overfitting even when the input variables are in a very huge number. All these benefits make the random forest to be one of the most accurate and effective general-purpose learning techniques available (Biau,2012).

III. Stochastic Gradient Boosting [by CHEW HOO WING]

Stochastic gradient boosting was introduced by Jerome H. Friedman in March 1999 as an improvement to his previously proposed gradient boosting technique and thus we need to understand what is a gradient boosting first before we get into the former. Gradient boosting is a gradient-descent "boosting" paradigm that generates highly robust interpretable procedures for both classification and regression. In a boosting process, base/weak learner which alone insufficient to perform the classification will be sequentially fitted into the training data iteratively to form an additive regression model. After a considerable amount of iterations, a single strong prediction rule can be produced with minimized mean-squared error. Motivated by Breiman's "bagging" (bootstrap aggregating) procedure, Friedman incorporate randomization into the gradient boosting by drawing a subsample of training data randomly without replacement from the full data set at each iteration. With this randomization, both the approximation accuracy and execution speed had a substantial improvement and this is how stochastic gradient boosting was born (Friedman,1999).

2.2.1 Discussion on Challenges and Limitations of 3 methods

I. Lasso Regression

By substituting the L2-norm in ridge regression with L1-norm, lasso regression is able to produce coefficients with 0 value but this characteristic could a double-edged sword. On a positive side, it could come in handy in feature selection if we want to select only one particular feature from a group of correlated features. This is because of the regularization parameter which set the value of correlated features to zero and thus automatically dropping them from the model. As a result, an easily interpretable model will be created. As good as it sounds, there are several issues that the L1 penalization could bring about. Firstly, picking only one feature from a correlated group is often arbitrary and the grouped selection becomes impossible. Because of this, group lasso as an extension to lasso has come to its place to serve for the purpose of multicollinearity. Apart from that, lasso regression might also drop some relevant independent variables during its continuous coefficients shrinking process. To avoid this issue, the shrinkage factor lambda must be discreetly determined to establish an appropriate degree of penalization of the system.

II. Random Forests

Random forests is a highly flexible algorithm which can work with either small or large datasets, large number of attributes and can be run in parallel. All in all, this is an elegant and cool concept which can be applied in tons of problem solving efforts in machine learning area. Nevertheless, it is not perfect by any means. The limitation of random forests is actually originating from its own inherent properties. As we mentioned above, it can be very fast to

train but accurate prediction results will often be slow to generate. This is due to the contradiction such that a higher accuracy in prediction results requires more trees exist in the forest which will eventually slow down the whole model. In this case, a powerful GPU would be a great help in running this highly parallel algorithm. Aside the speed, the random selection of features could sometimes be problematic such that it will limit the ability to distinguish the classifier if an uninformative or correlated feature being selected. Random forests seem to be intuitively easy to understand at first but it is quite difficult to delve into how the algorithm actually works internally and hence thus this theory is often misinterpreted.

III. Stochastic Gradient Boosting

Compared to random forests, gradient boosting's training usually takes longer time as the models/trees are being built sequentially rather than parallelly. The new model will be built one at a time to correct any errors made by the previously trained model. This "boosting" procedure is really effective in minimizing errors but it could be easily lead to overfitting too. Overfitting is a condition where the model becomes excessively complex by considering too many parameters and becomes poor in prediction performance. To cope with this circumstance, the shrinkage factor and the maximum depth of the addictive model must be fine-tune.

2.5 Scope

To perform salary prediction based on a collection of massive labelled data, technology in text classification and feature extraction from a text-based dataset is requisite and hence we have reviewed several relevant journal papers to consolidate our understandings in this specific area. By examining through a respectable number of relevant journals in the field of machine learning, our cognition about the artificial intelligence has been drastically developed. We are now able to appreciate the beauty of these formulas and concepts and conduct an in-depth thinking on their meanings, inferiors and the rationality of their existence. We would like to express our respect to all the researchers who have been contributing their wisdom crystal which established a strong foundation for the rapid growth of today's machine learning area.

III. METHODOLOGY

A. Data Gathering

The training data for this project consists of a .csv file downloaded from https://www.kaggle.com/stackoverflow/so-survey-2017/ which is the Stack Overflow Developer Survey, 2017. This file consists of 155 questions (column) and approximately 51,400 developers have provided response (row). Among them, there are close to 12,900 responses with salary field filled-in and these will be our training data set.

B. Data Pre-processing

Certain columns will be omitted such as respondent ID, feedbacks towards the survey as they are irrelevant to the scope of our research question. In order for achieving multiclass classification purpose, the salary field of the records has been categorized into:

1. 20000 (original annual salary range (in USD): between $10000 and $30000)

2. 40000 (original annual salary range (in USD): between $30000 and $50000)

3. 60000 (original annual salary range (in USD): between $50000 and $70000)

4. 80000 (original annual salary range (in USD): between $70000 and $90000)

5. 100000 (original annual salary range (in USD): $90000 and above)

Pre-processing on the data is done such as drop any record contains NA value and string data are converted in classes [0,1] with LabelEncoder. Then the pre-processed data is being split into test and train data on a ratio of 20% / 80%. Afterwards, three different approaches which include lasso regression, random forests and stochastic gradient boosting will be applied to these pre-processed data to predict the salary range of a respective programmer. The originally proposed idea of knowledge by generating bag-of-words models using CountVectorizer was abandoned as the result is unsatisfactory.

C. Final Program Overview

The final program consists of a menu structure which allows the selection for several options:

1.Train and Save Models

This option will first load the dataset, perform data pre-processing, split the test/train data and then stored these data into sav files by using Pickle. Afterwards, all three models (lasso, random forest and gradient boosting) will be trained and trained models will also be saved into their own respective sav files. The rationale for using Pickle to read and write pre-processed data and trained models is to speed up the execution time of other options such as compare models since the models no need to be trained again and gain.

2.Compare Models (Lasso Regression, Random Forest, Gradient Boosting)

This option displays the confusion matrix and classification report of lasso, random forest, stochastic gradient boosting models for user to view, compare and further analyse.

3.Lasso Regression with different alphas

This option displays several plots demonstrate the effect of different alphas used in lasso regression.

4.Feature Importances with Random Forest/Gradient Boosting

This option displays each column/feature with their respective weight on determining the prediction of salary based on the random forest model and boosting model.

5. Random Forest with Parameter Tuning

This option will shows the illustrations that help us to tune the parameter for random forests.

6. Gradient Boosting Learning Rate Tuning

This option will shows the illustration on the effect of different learning rates on the boosting model.

7.Feature Extraction with CountVectorizer (DEMO ONLY)

This option displays the result of feature extraction by using CountVectorizer on the training data set.

6.Exit

This option allows user to terminate the program.

D. **Classification method 1: Lasso Regression**

$$Cost\ (W) = RSS(W) + \lambda * (sum\ of\ absolute\ value\ of\ weights)$$

$$= \sum_{i=1}^{N} \left\{ y_i - \sum_{j=0}^{M} w_j\, x_{ij} \right\}^2 + \lambda \sum_{j=0}^{M} |w_j|$$

The cost/objective function of Lasso Regression

The illustration above represents the cost function that needed to be minimized along the lasso regression. RSS over here stands for the Residual Sum of Squares which means the sum of square of prediction errors while lambda (λ) here is nothing but the parameter which help to balance the weightage given to reduce the RSS against the sum of absolute value of coefficients. λ can be set as either 0 (equivalent to simple linear regression), infinity/∞ (the coefficients will be zero) or between 0 and ∞ then the coefficients produced will between 0 and the value of the given λ.

To employ lasso regression in our python program, we can conveniently import the Lasso model from the scikit-learn class library specifically the **sklearn.linear_model** class. Here is a sample code for importing and employing Lasso model by using python:

```
from sklearn.linear_model import Lasso
def lasso_regression(data, predictors, alpha):
    #Fit the model
    lassoreg = Lasso(alpha=alpha, normalize=True, max_iter=1e5)
    lassoreg.fit(data[predictors], data['y'])
    y_pred = lassoreg.predict(data[predictors])

    #Return the result in pre-defined format
    rss = sum((y_pred-data['y'])**2)
    ret = [rss]
    ret.extend([lassoreg.intercept_])
        ret.extend(lassoreg.coef_)
        return ret
```

E. Classification method 2: Random Forests

In applying the random forests approaches, there are several aspects that we need to pay more attention to. Firstly, we should avoid the selection of most common unigrams from each data field. If not, the words such as “and”, “the” will flood the resulting feature vector and affect the accuracy of final prediction result. The training data should also be divided into different groups based on salary ranges. This is to single out the features that are highly correlated to salary but occur rarely throughout the corpus. In addition, the maximum depth parameter must be set to a very large number so a wide variety of features will be put into consideration before a decision is made.

Just like the lasso regression, we can also found the Random Forests model in the scikit-learn class library which is the sklearn.ensemble.RandomForestClassifier to be more specific. The following sample code shows part of our proposed implementation of random forests in python:

```
from sklearn.ensemble import RandomForestClassifier as RFC
import numpy as np
import pandas as pd
def random_forest(data, columns=data.feature_names):
        #construct a pandas dataframe
        df = pd.DataFrame(data)
        # create a random uniform distribution between 0,1 and assign ¾ of data
        df['is_train'] = np.random.uniform(0, 1, len(df)) <= .75
        #decompressing data
        df['salary_range'] = pd.Categorical.from_codes(data.target, data.target_names)
        #split train, test into their own dataframes and define features(input variables)
        train, test = df[df['is_train']==True], df[df['is_train']==False]
        features = df.columns[0:4]
        #establish random forest class with number of jobs and number of trees
        forest = RFC(n_jobs=2, n_estimators=1000)
        y, _ = pd.factorize(train['salary_range'])
        #fit model to the training data and predict the salary range
        forest.fit(train[features], y)
        preds = data.target_names[forest.predict(test[features])]
        return preds
```

F. Classification method 3: Stochastic Gradient Boosting

To apply the stochastic gradient boosting in our proposed python-written system, we have first decided to adopt the XGBoost library which is labelled as a scalable, portable and distributed gradient boosting solution. Most importantly, it is considerably faster than scikit-learn's GradientBoostingClassifier. The data is being pre-processed by loadtxt method from numpy class and train_test_split method from sklearn.model_selection class. All these are shown in the sample code below:

```
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
# load data
dataset = loadtxt('survey_results_public.csv', delimiter=",")
# split data into X and y
X = dataset[:,0:100]
Y = dataset[:,100]
# split data into train and test sets
seed = 7
test_size = 0.33
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)
def stochastic_gradient_boosting(X_train, X_test, y_train, y_test):
        # fit model no training data
        model = XGBClassifier()
        model.fit(X_train, y_train)
        # make predictions for test data
        y_pred = model.predict(X_test)
        predictions = [round(value) for value in y_pred]
        return predictions
```

However, considering that applying XGBoost library in Microsoft Window environment requires a series of complicated and miscellaneous steps such as executing commands in GitBash to get the package, download MinGW, installing CMake etc. Moreover, the speed difference between XGBoost and Scikit-learn's GradientBoostingClassifier on training and predicting our relatively small datasets is tolerable. Hence, we finally decided to stick with and Scikit-learn's GradientBoostingClassifier instead of XGBoost by juggling between the benefits and harms.

IV. RESULTS

The following tables showing the classification reports before getting the best train/test-split data:

Methods	Overall Accuracy	Average Precision	Average Recalls	Average F1-score
Lasso Regression	0.21	0.39	0.21	0.19
Random Forest	0.20	0.07	0.20	0.09
Gradient Boosting	0.17	0.23	0.17	0.13

TABLE I. Comparison of accuracy, precision, recalls, f1-score between different models

Annual Salary Range	Precision	Recall	F1-score
20000 ($10000 - $30000)	0.58	0.02	0.04
40000 ($30000 - $50000)	0.49	0.20	0.29
60000 ($50000 - $70000)	0.15	0.40	0.22
80000 ($70000 - $90000)	0.15	0.57	0.24
100000 ($90000 and above)	0.33	0.00	0.01

TABLE II. Classification Report of Lasso Regression

Annual Salary Range	Precision	Recall	F1-score
20000 ($10000 - $30000)	0.20	0.98	0.34
40000 ($30000 - $50000)	0.00	0.00	0.00
60000 ($50000 - $70000)	0.21	0.27	0.24
80000 ($70000 - $90000)	0.00	0.00	0.00
100000 ($90000 and above)	0.00	0.00	0.00

TABLE III. Classification Report of Random Forest

Annual Salary Range	Precision	Recall	F1-score
20000 ($10000 - $30000)	0.16	0.62	0.25
40000 ($30000 - $50000)	0.24	0.05	0.08
60000 ($50000 - $70000)	0.17	0.22	0.19
80000 ($70000 - $90000)	0.21	0.09	0.12
100000 ($90000 and above)	0.33	0.05	0.09

TABLE IV. Classification Report of Gradient Boosting

The following tables showing the classification reports after getting the best train/test-split data:

Methods	Overall Accuracy	Average Precision	Average Recalls	Average F1-score
Lasso Regression	0.22	0.30	0.22	0.18
Random Forest	0.50	0.28	0.50	0.36
Gradient Boosting	0.58	0.53	0.58	0.51

TABLE V. Comparison of accuracy, precision, recalls, f1-score between different models

Annual Salary Range	Precision	Recall	F1-score
20000 ($10000 - $30000)	0.40	0.01	0.01
40000 ($30000 - $50000)	0.48	0.19	0.27
60000 ($50000 - $70000)	0.17	0.45	0.25
80000 ($70000 - $90000)	0.17	0.66	0.26
100000 ($90000 and above)	0.00	0.00	0.00

TABLE VI. Classification Report of Lasso Regression

Annual Salary Range	Precision	Recall	F1-score
20000 ($10000 - $30000)	0.00	0.00	0.00
40000 ($30000 - $50000)	0.49	0.96	0.65
60000 ($50000 - $70000)	0.00	0.00	0.00
80000 ($70000 - $90000)	0.00	0.00	0.00
100000 ($90000 and above)	0.53	0.74	0.62

TABLE VII. Classification Report of Random Forest

Annual Salary Range	Precision	Recall	F1-score
20000 ($10000 - $30000)	0.78	0.44	0.57
40000 ($30000 - $50000)	0.57	0.90	0.70
60000 ($50000 - $70000)	0.29	0.02	0.04
80000 ($70000 - $90000)	0.26	0.10	0.14
100000 ($90000 and above)	0.57	0.80	0.66

TABLE VIII. Classification Report of Gradient Boosting

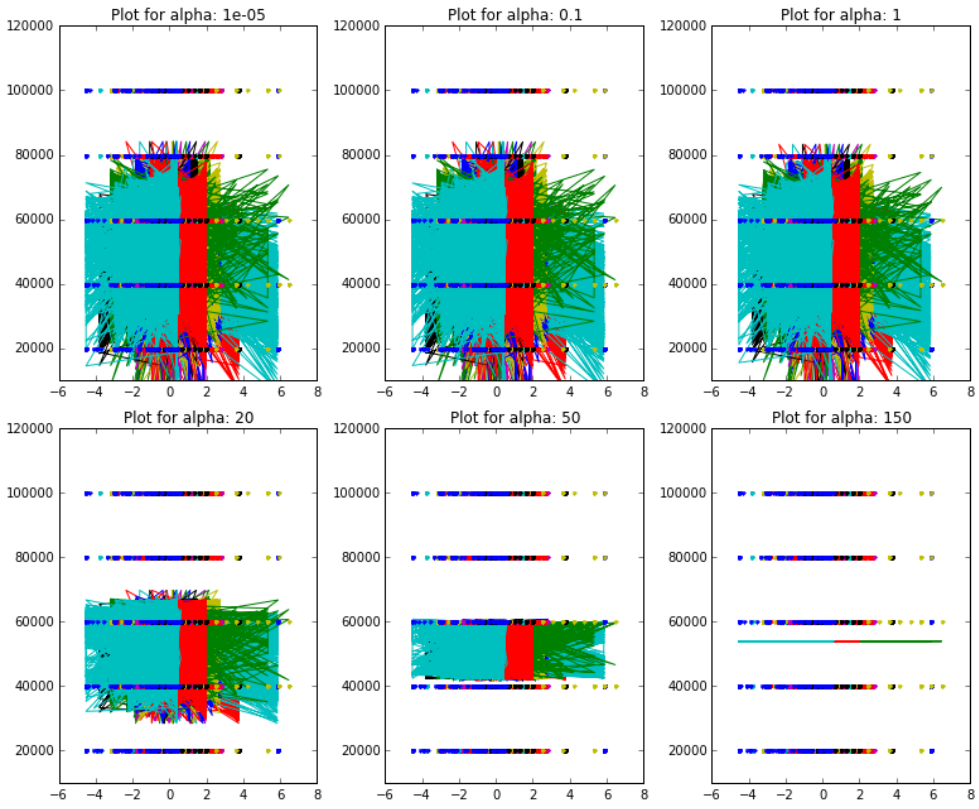

Fig.1 Plot showing the effect of increasing value of alphas (include all the features)

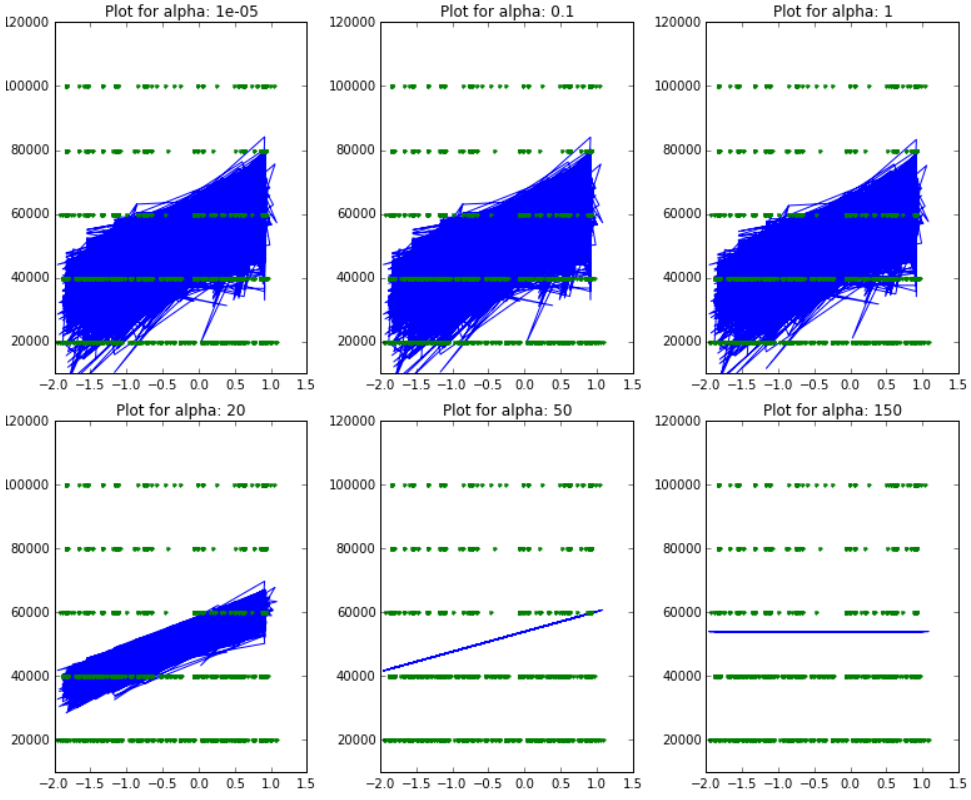

Fig.2 Plot showing the effect of increasing value of alphas (ONLY "COUNTRY" feature)

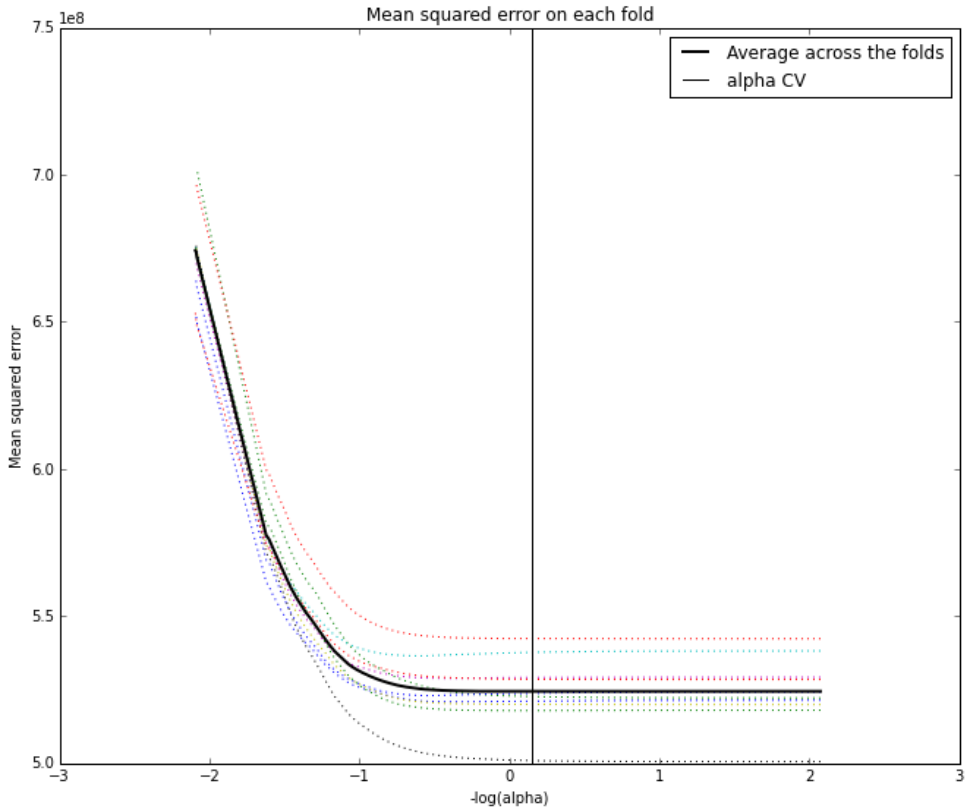

Fig.3 Plot showing the Mean Squared Error of Lasso Regression on each fold

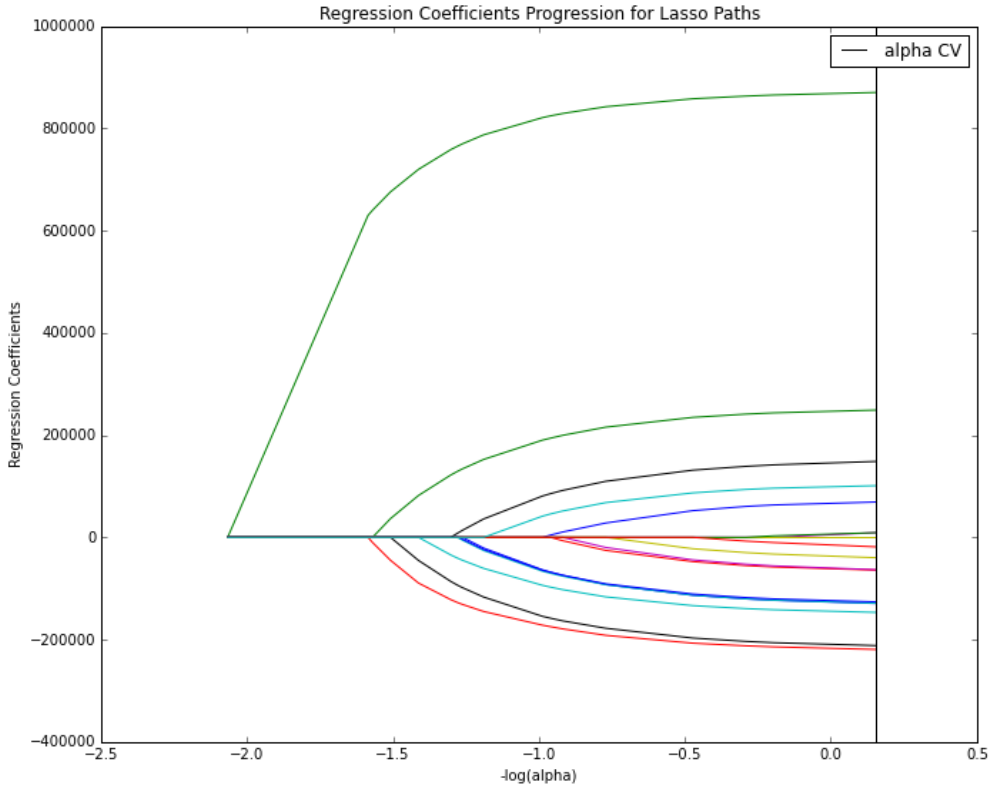

Fig.4 Plot showing the relative importance (regression coefficients) of every variable

```
Variable Names and their respective regression coefficient

{'Race': 1126.3077715560648, 'CompanySize': -1436.4540236508133, 'EmploymentStatus': -1635.8594783803064, 'YearsProgram':
-1430.5872528446976, 'Country': 9683.1421264501987, 'University': -2454.4823860924021, 'DeveloperType': -440.99299939761755,
'ProgramHobby': 770.88229280857956, 'HomeRemote': -2353.8187299820961, 'HighestEducationParents': -206.43108238916548, 'Gender':
103.02290745219989, 'TabsSpaces': -1402.3845343273017, 'MajorUndergrad': 0.0, 'FormalEducation': 103.22452080768085, 'PronounceGIF':
1657.2457627645383, 'YearsCodedJob': -706.09891663165422, 'HaveWorkedLanguage': -712.5838958896029, 'CompanyType': 2756.5526738797457}
```

Variable	Regression Coefficients
Country	9683.1421264501987
CompanyType	2756.5526738797457
PronounceGIF	1657.2457627645383
Race	1126.3077715560648
ProgramHobby	770.88229280857956
FormalEducation	103.22452080768085
Gender	103.02290745219989
MajorUndergrad	0.0
HighestEducationParents	-206.43108238916548
DeveloperType	-440.99299939761755
YearsCodedJob	-706.09891663165422,
HaveWorkedLanguage	-712.5838958896029
TabsSpaces	-1402.3845343273017
YearsProgram	-1430.5872528446976
CompanySize	-1436.4540236508133
EmploymentStatus	-1635.8594783803064
HomeRemote	-2353.8187299820961
University	-2454.4823860924021

TABLE IX. Variable Names and their respective regression coefficient (Lasso Regression)

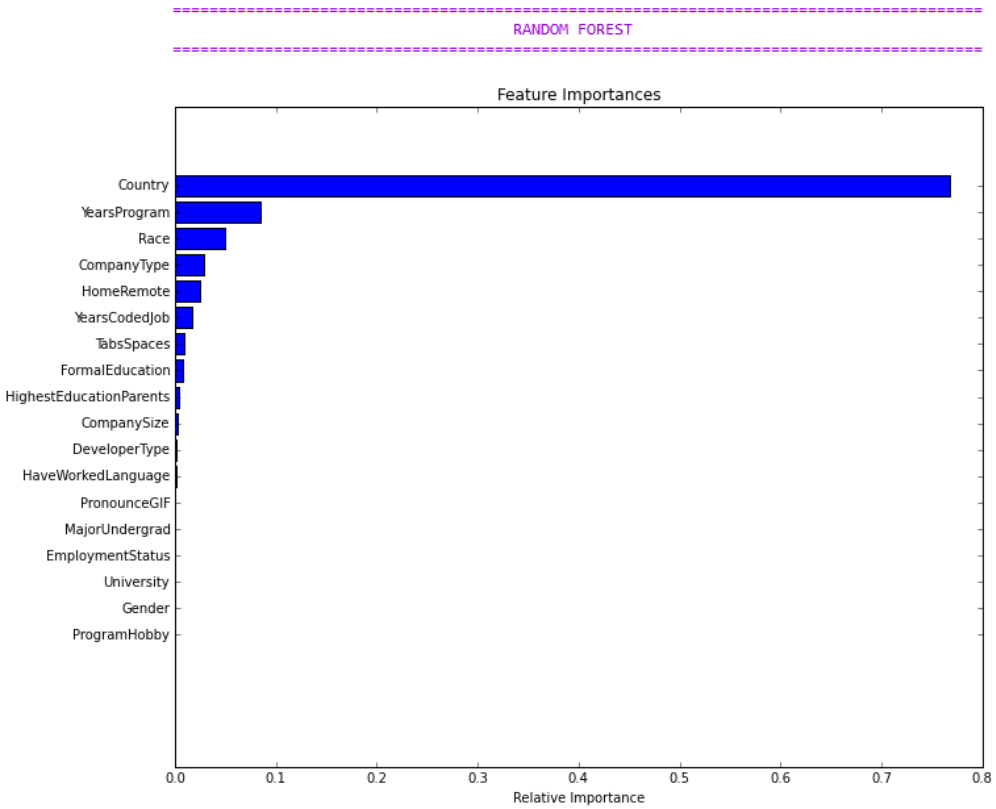

Fig.5 Bar chart showing the relative importance using Random Forest

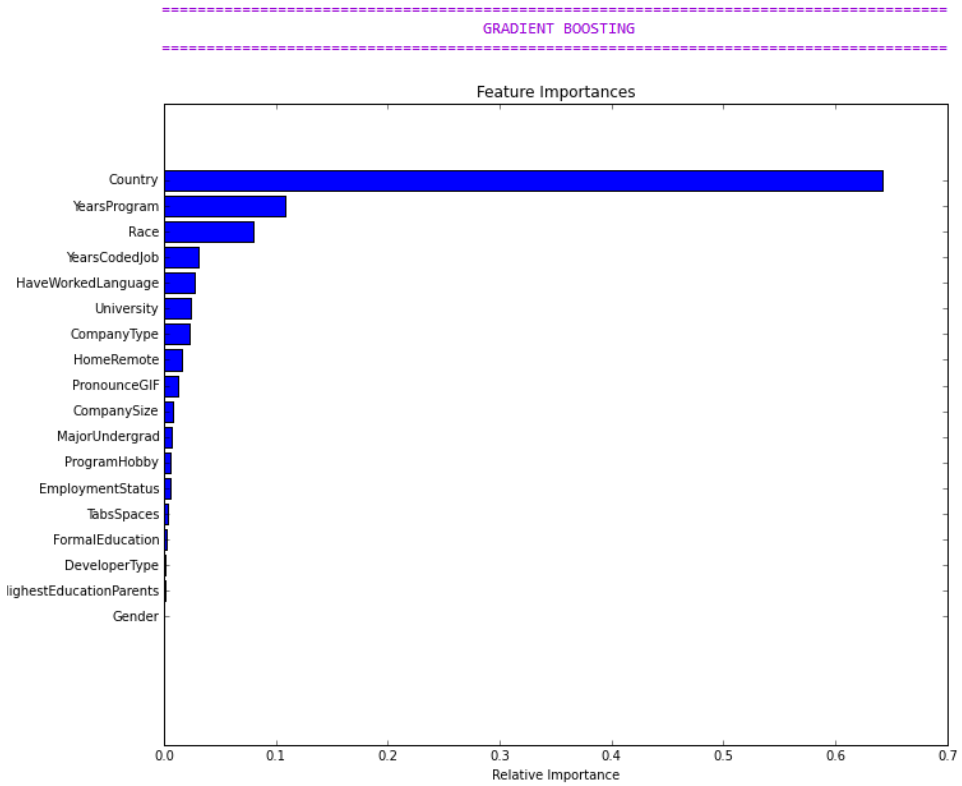

Fig.6 Bar chart showing the relative importance using Gradient Boosting

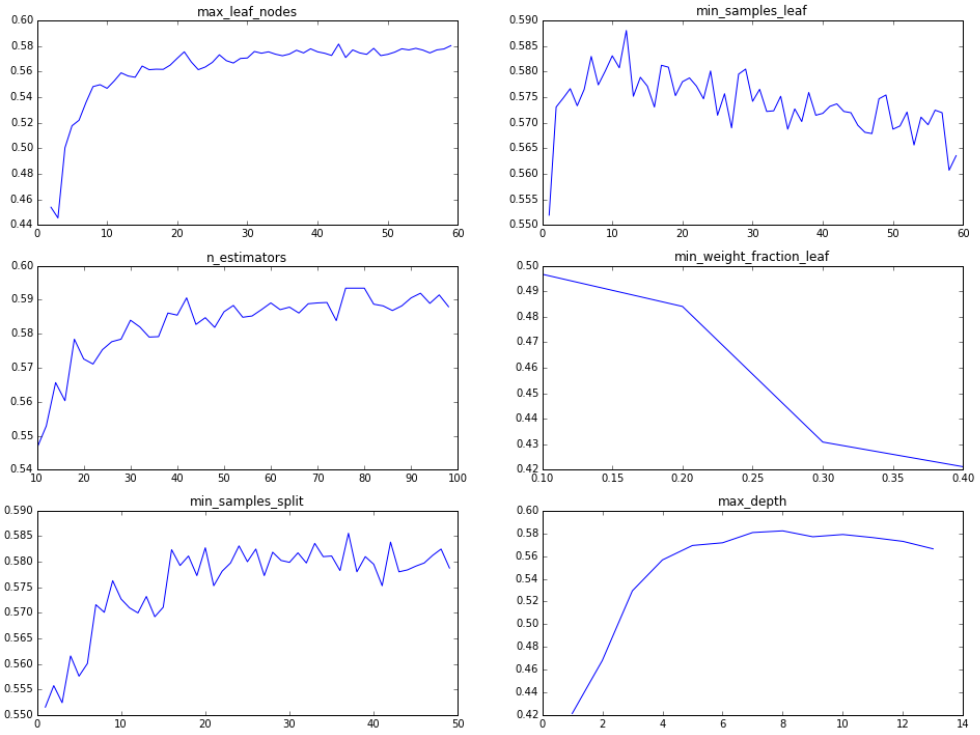

Fig.7 Plot showing the range of different parameters in Random Forest before tuning

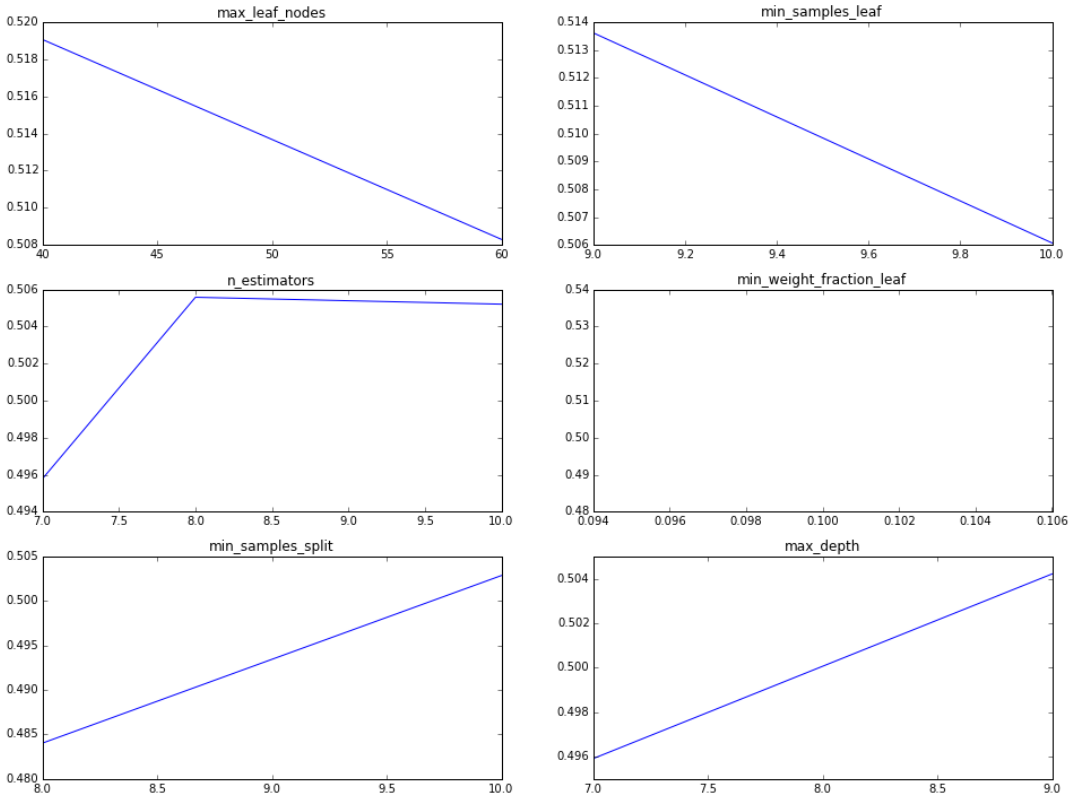

Fig.8 Plot showing the range of different parameters in Random Forest after tuning

```
Model with rank: 1
Mean validation score: 0.5305)
Parameters: {'max_leaf_nodes': 40, 'min_samples_leaf': 9, 'n_estimators': 7, 'min_weight_fraction_leaf': 0.1, 'min_samples_split': 8, 'max_depth': 9}

Model with rank: 2
Mean validation score: 0.5220)
Parameters: {'max_leaf_nodes': 60, 'min_samples_leaf': 10, 'n_estimators': 8, 'min_weight_fraction_leaf': 0.1, 'min_samples_split': 8, 'max_depth': 9}

Model with rank: 3
Mean validation score: 0.5206)
Parameters: {'max_leaf_nodes': 40, 'min_samples_leaf': 9, 'n_estimators': 8, 'min_weight_fraction_leaf': 0.1, 'min_samples_split': 10, 'max_depth': 7}

Model with rank: 4
Mean validation score: 0.5181)
Parameters: {'max_leaf_nodes': 60, 'min_samples_leaf': 9, 'n_estimators': 10, 'min_weight_fraction_leaf': 0.1, 'min_samples_split': 10, 'max_depth': 7}
```

Fig.9 Showing the best performed parameters for random forests

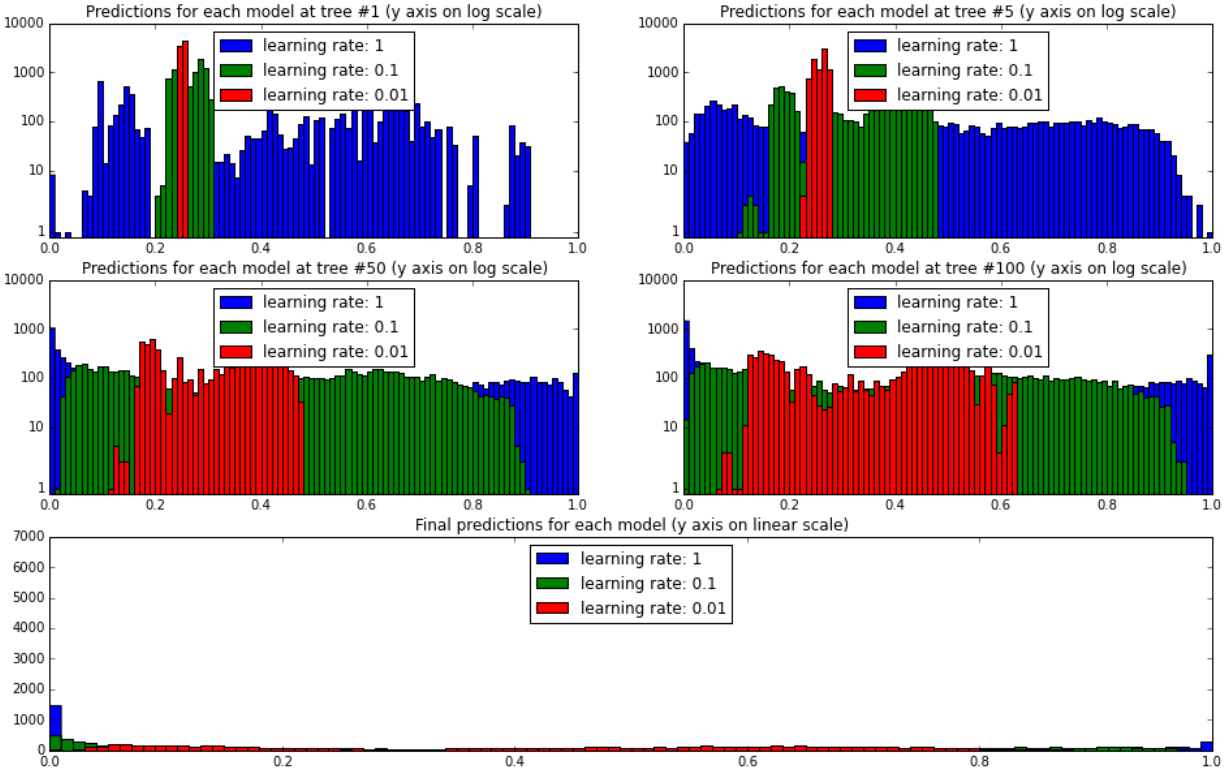

Fig.10 Plot showing the effect of different learning rate on each stage of Gradient Boosting prediction

V. DISCUSSION

From the first set of the classification reports, we can see that all three models perform poorly with lasso regression being the far-fetched winner here with the score of 0.21 for overall accuracy and 0.19 for average F1-score. We were shocked with this poorly performed results such all three algorithms can only predict the correct classes for approximately 20% of the test data. What make us more disappointing is such that this poorly performed prediction result remains the same even after we put in effort in tuning the parameters for respective models. Then we started to shift our attention from the machine learning algorithms to the dataset itself.

Initially, the pre-processed data was split into test/train data based on 20/80 ratio with a constant random state of 42. By the way, 42 is chosen because of the fun fact such that it is the answer to life, the universe and everything as Douglas Adams stated in his The Hitchhiker's Guide to the Galaxy fiction story. A constant random state can help us to get the reproducible prediction result every time we train and test models by ensuring the same train/test data are used for each time. This is a generally good approach but it is also the main factor that cause the poor performance of our prediction results. The reason behind is that the first test/train data we get at first is not suitable for random forests and gradient boosting and makes them keep producing unsatisfactory result even after parameter tuning. Hence, we decided to run several cross-validation tests to figure out the best test/train data for the best prediction results. After getting the best performed test/train data, they are being saved in sav files by using Pickle library and these data will then be constantly used for this project.

During the cross-validation process in obtaining the best performed test/train data, one interesting facts that we noticed is that both the random forests and gradient boosting have been well improved in their prediction performance but lasso regression remains the same poor performance with around overall 20% of correct classes predictions.

From the second set of classifications reports (with the best performed test/train data), gradient boosting model becomes the best performer here with 0.58 of overall accuracy and 0.51 Average F1-score. This is a huge contrast compared to the previous set of result as the boosting model being the poorest performed algorithm with only 0.17 score for both the overall accuracy and average F1-score. Random forests is the close second by hitting score of 0.50 of overall accuracy and 0.36 of average F1-score which is also a huge leap forward compared to the previous results. With only around 0.22 of the overall accuracy and 0.18 of average F1-score, lasso regression, the ‘former champion’ for the previous set of result has now become the laggard among the three algorithms as there is no significant improvement it gained from the new set of test/train data.

Apart from the classification reports, the Fig.3 plot has shown us the abnormally high mean squared error for this lasso model (ranged from 5.0*1e8 to 7.0*1e8) on each fold which directly tell us the lasso regression is really struggling on this task. The primary reason that the lasso regression performed so poor in this programmer salary classification is such that using regression as classifier often requires a very good discretization process on the data. In our case, the salary data is just being simply classified into 5 classes based on the range of their values and this is probably what makes the lasso regression generate such a low accuracy prediction result. However, get to the bottom of the matter, regression which output is real value is often not an appropriate approach for classification as in this case. One exception might be the logistic regression which assigns probability and serve well in binary class classification but since our project is a multi-class salary range prediction and hence it also does not suit well here not to mention our proposed lasso regression which is based on the linear regression model.

If we ignore the inborn defects of regression to be used in this salary range classification, the strength of lasso regression which is feature selection also doesn't serve well here. Applying different alpha values such as 20, 50 ,150 does make certain features being dropped and this is what we intend to achieve initially. However, we have found that dropping these features has not improve accuracy of lasso model but instead makes it even worse. The reasons behind are well illustrated in Fig.1. In Fig.1, we can clearly see that all the features have different gradients/fittings and hence they are independent from each other. Thus, dropping any one of them is not an ideal approach and we should stick with the either alpha 1, 0.1 or below and they basically perform the same. This is corresponding to the black vertical line in both Fig.3 and Fig.4 that tell us the optimum alpha value which is approximately 0.1. In short, lasso regression is not a suitable candidate for this AI application.

Shifting our attention from the lasso regression to the random forest and gradient boosting, the primary focus will be on the how to optimize their parameter values as they both can gain significant improvement in prediction results after proper tuning.

In tuning parameter of random forests, a cross-validated grid-search approach is being applied. Initially, a parameter grid consists of parameters such as max_depth, n_estimators, min_samples_leaf etc. are used with a respectively wide range of values for each parameter. Then a plot as in the Fig.7 was generated and allow us to analyse and narrow down the parameters range accordingly. For example, min_samples_leaf will be narrowed to a range between 8 and 11 as the highest mean validation score falls in this range. This narrowing process was iteratively carried out until a plot such as the Fig.8 is generated and the model with best rank (as in Fig.9) will be adopted for the recommended parameters values for our random forests model.

Similarly, the same concept is being applied to the parameters tuning in gradient boosting too but in a less tedious way which can be seen in Fig.10. The only parameter being tuned here is learning rate which affect the most in preventing the overfitting by slowing down the learning process. Other parameters such as tree-related parameters and subsample have also been tuned and tested but the result isn't satisfactory and hence we decided to stick with learning rate tuning only. By the way, this is also the reason why our originally proposed "stochastic gradient boosting" is now being written as "gradient boosting" only as the random factor, subsample is not being used anymore. What Fig.10 has shown us is such that high learning rate will push more and more predictions out of the 0 and 1 scope as the number of trees increase during boosting while low learning rate is able to keep more predictions during the boosting process. Based on the illustration of Fig.10, learning rate of 0.01 was chosen for our boosting model to reach an optimal balance between prediction performance and the training speed such that any value lower than 0.01 can be computationally intractable.

By looking back at the second set of classifications reports, we can actually get some extra information aside from the overall accuracy and F1-score for each model. For example, gradient boosting generates an overall well-balanced prediction result among all the salary ranges classes such that the precision and recall for each class are fairly distributed except for the '60000' class. Interestingly, lasso regression which is the overall poorest performer is actually being the best in predicting the salary range of '60000' and '80000' classes. On the other hand, random forests only focused its predictions on the '40000' and '100000' these two classes whole neglecting the other three classes making it reach exceptionally high recall score for these two classes. This can actually lead to an interesting situation such that lasso regression instead of random forests will be adopted to complement the weakness of gradient boosting in predicting the '60000' and '80000' salary range classes. The reason behind is to cover as more salary ranges as possible considering the purpose of our AI application so the random forests, despite being the second-best algorithm, will not be employed.

Aside from the performance measurement, there are other important information that we can get from this project too such as the feature importance. In this case, lasso regression doesn't directly give us the relative importance of each feature like random forests and gradient boosting so the approach based on the regression coefficient of each features is adopt. From the TABLE IX, Fig.5 and Fig.6, we can clearly see that 'Country' being the feature that contribute the greatest weight on the programmer salary prediction. Other factors such as 'CompanyType', 'Race' and 'YearsProgram' also play important roles in determining a programmer's salary. This actually tell us a fact such that a programmer's technical skills such as worked programming language, developer type and education level didn't affect much on his/her remuneration but largely depends on the environment and the company joined.

VI. CONCLUSION

By assessing the performance of all three proposed machine learning algorithms on programmer salary prediction, we can clearly see that the prediction result is barely satisfactory since the best performer, gradient boosting only scores an overall accuracy of nearly 60%. The primary limitation for this work is probably the dataset and how we pre-processed the data. The rationale behind this statement is such that our originally proposed feature extraction with CountVectorizer was not bring into play because of its unsatisfactory result due to our shallow understanding in this area. In addition, a simple but lack in thoughts approach has been used to divide the salary values into five different classes for classification purpose. Besides that, the huge impact of different test/train datasets on the prediction results of random forests and gradient boosting basically tell us that the data could largely be the limiting factor that bottleneck these algorithms' prediction performance. Hence, there is still a lot of room for future enhancement to be made especially on the data part such as obtaining a higher volume of dataset that covers more programmers and more aspects and a more appropriate way to pre-process the data. Aside from the data itself, our parameter tuning on machine learning algorithm is also barely satisfactory with only very few parameters being tuned in a rough way. All in all, we can basically conclude that this project still has a huge potential to be improved and our works has initially proved its feasibility and practicality. Considering the strong demands of IT graduates on the programmer salary prediction, tons of future amendment and enhancement on this project is something foreseeable.

REFERENCES

Anonymous, 2013. *Job Salary Prediction.* [Online]
Available at: http://www.cs.ubc.ca/~nando/540-2013/projects/p58.pdf
[Accessed 4 7 2017].

Biau, G., 2012. Analysis of a Random Forests Model. *Journal of Machine Learning Research,* Volume 13, pp. 1063 - 1095.

Brownlee, J., 2016. *How to Develop Your First XGBoost Model in Python with scikit-learn.* [Online]
Available at: http://machinelearningmastery.com/develop-first-xgboost-model-python-scikit-learn/
[Accessed 8 7 2017].

Carnevale, R., 2015. *Understanding Gradient Boosting, Part 1.* [Online]
Available at: http://rcarneva.github.io/understanding-gradient-boosting-part-1.html
[Accessed 1 8 2017].

Cross, A., 2015. *Random Forests in Python with scikit-learn.* [Online]
Available at: http://www.agcross.com/2015/02/random-forests-in-python-with-scikit-learn/
[Accessed 7 7 2017].

Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, ¨, 2011. Scikit-learn: Machine Learning in Python. *The Journal of Machine Learning Research,* Volume 12, p. 2825–2830.

Haden, D., 2016. *First Kaggle Script: Tuning Random Forest Parameters.* [Online]
Available at: https://www.kaggle.com/hadend/tuning-random-forest-parameters
[Accessed 30 7 2017].

Ho, T. K., 1998. The Random Subspace Method for. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE,* 20(8), pp. 832-844.

JAIN, A., 2016. *A Complete Tutorial on Ridge and Lasso Regression in Python.* [Online]
Available at: https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-ridge-lasso-regression-python/
[Accessed 8 7 2017].

JAIN, A., 2016. *Complete Guide to Parameter Tuning in Gradient Boosting (GBM) in Python.* [Online]
Available at: https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/
[Accessed 1 8 2017].

Jeremy Howard,JQ Veenstra,Waleed Kadous,Tao Xu,Piotr Płoński,Ricardo Vladimiro, Li Yang,Eric Nunes,Arjen P. De Vries , 2011. *When is a random forest a poor choice relative to other algorithms?.* [Online]
Available at: https://www.quora.com/When-is-a-random-forest-a-poor-choice-relative-to-other-algorithms
[Accessed 6 7 2017].

Laurens van de Wiel,Peter Flom,Manish Tripathi,Joshua Kim,Francesco Gadaleta, 2015. *Are there any disadvantages or weaknesses to the L1 (LASSO) regularization technique?.* [Online]

Available at: https://www.quora.com/Are-there-any-disadvantages-or-weaknesses-to-the-L1-LASSO-regularization-technique
[Accessed 6 7 2017].

Lukas Meier, Sara van de Geer and Peter Bühlmann, 2008. The group lasso for logistic regression. *Journal of the Royal Statistical Society: Series B ,* Volume 70, pp. 53-71.

Sculati, R., 2015. *Behaviour analysis through Machine learning techniques,* Finland: HAAGA-HELIA.

SHARMA, G., 2016. *Lasso Regression in Python, Scikit-Learn.* [Online]
Available at: https://tektrace.wordpress.com/2016/04/09/lasso-regression-in-python-scikit-learn/
[Accessed 28 7 2017].

Soumya George K1, Shibily Joseph2 , 2014. Text Classification by Augmenting Bag of Words (BOW) with Co-ocurrence Feature. *IOSR Journal of Computer Engineering (IOSR-JCE),* 16(1), pp. 34-38.

Tibshirani, R., 1996. Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological),* 58(1), pp. 267-288.