Capstone Project
Machine Learning Engineer Nanodegree

Jian Zhang
10/19/2016

I. Definition

Project Overview

I decided to start my very first Kaggle competition called "Allstate Claims Severity"[1], which was launched on Oct 10th, 2016. In this competition, the goal is to predict single claim loss based on dataset provided by Allstate insurance company. Allstate is currently developing automated methods of predicting the claim cost, then determine severity of each claim based on customers' record and other information related to every claim. In the past, regression problems like this are successfully solved by boosting method like Gradient Boosting Machine. Until recently, XGBoost becomes very popular and dominates machine-learning competitions. However, ensemble and stack techniques are often used to form an extremely strong learner based on several sophisticated models to stand out in these competitions.

Problem Statement

Predicting the loss of each claim is a regression problem in this case. To be mentioned that, we have to deal with a large dataset this time, which contains over 180000 observations with 132 variables including 100 categorical variables and 32 numerical variables. Kaggle has already split data into training and testing set, which can be download directly from the website. Final prediction is expected to run on the testing set and extract only predicted loss for submission. The Kaggle is going to provide error and ranking for each submission. In this case, we will use the error computed by Kaggle to compare different models we applied in this project.

In order to find the best algorithm that minimizes prediction error, our approach to this problem is fitting different classic models that are known for good in solving classification problems. By comparing their prediction error obtained from Kaggle, we pick one or two models that have the best performance. Then, let's combine these models together with other type of models, like penalized linear regression, inspired by the idea of ensemble methods. Noting that, we don't want to combine similar model in this approach, even though they might have lowest error. The reason why sometimes ensemble methods did work is that different models tend to capture different features or characteristic learning from the data. Combining these models is trying to construct a new model that has both advantages from these individual models. So, we may want to try to combine our best and advanced algorithm with other method, which is design to learn from date in different perspective.

I will provide the performances of each algorithm that we have tried, as well as our ensemble model. Hopefully, this method will outperform other models and provides the lowest prediction error.

Metrics

In this project, evaluation metrics have been already assigned by Kaggle. We will use mean absolute error (MAE) between the predicted loss and the actual loss.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|predicted_i - actual_i|$$

It is more reasonable to use this metric rather than mean square error in this scenario, since the insurance company only cares about actual difference between prediction and actual loss. Again, our goal is to find a model that minimizes the mean absolute error as much as possible.

II. Analysis

Data Exploration

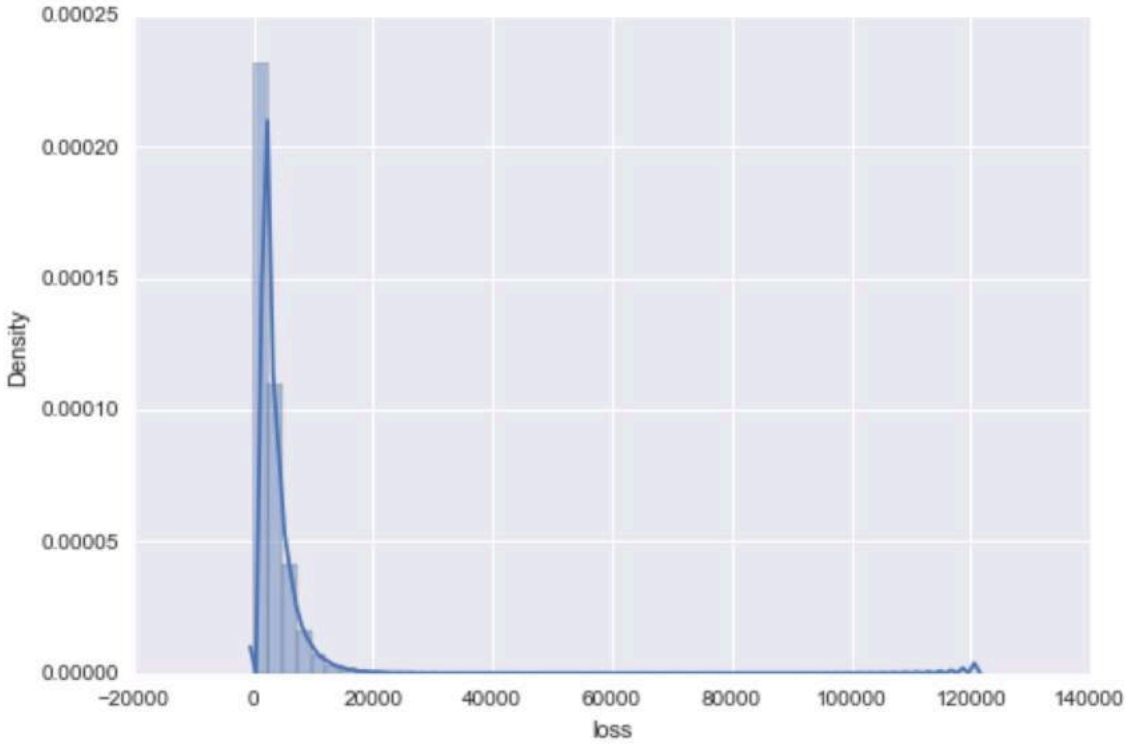
Let's first check variables. In this dataset, it has 116 categorical variables and 14 numerical variables, so we need to factorize those categorical variables before putting them into different models. Also, we check the missing value in the dataset, luckily, we don't need to deal with missing values in this project. Then, we focus on target variable loss, descriptive statistics shows that claim loss seems to have large range, which might not be a good sign when we run different models on it. Noting that sklearn[2] package doesn't recognize categorical data automatically, we will first factorize these values into number before fit in models. In terms of variable selection, LASSO regression will be tried later on and see how it performs. At this stage, let's focus on target variable by plotting histogram of claim loss.

	id	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	...	cont6
0	1	A	B	A	B	A	A	A	A	B	...	0.718367
1	2	A	B	A	A	A	A	A	A	B	...	0.438917
2	5	A	B	A	A	B	A	A	A	B	...	0.289648
3	10	B	B	A	B	A	A	A	A	B	...	0.440945
4	11	A	B	A	B	A	A	A	A	B	...	0.178193

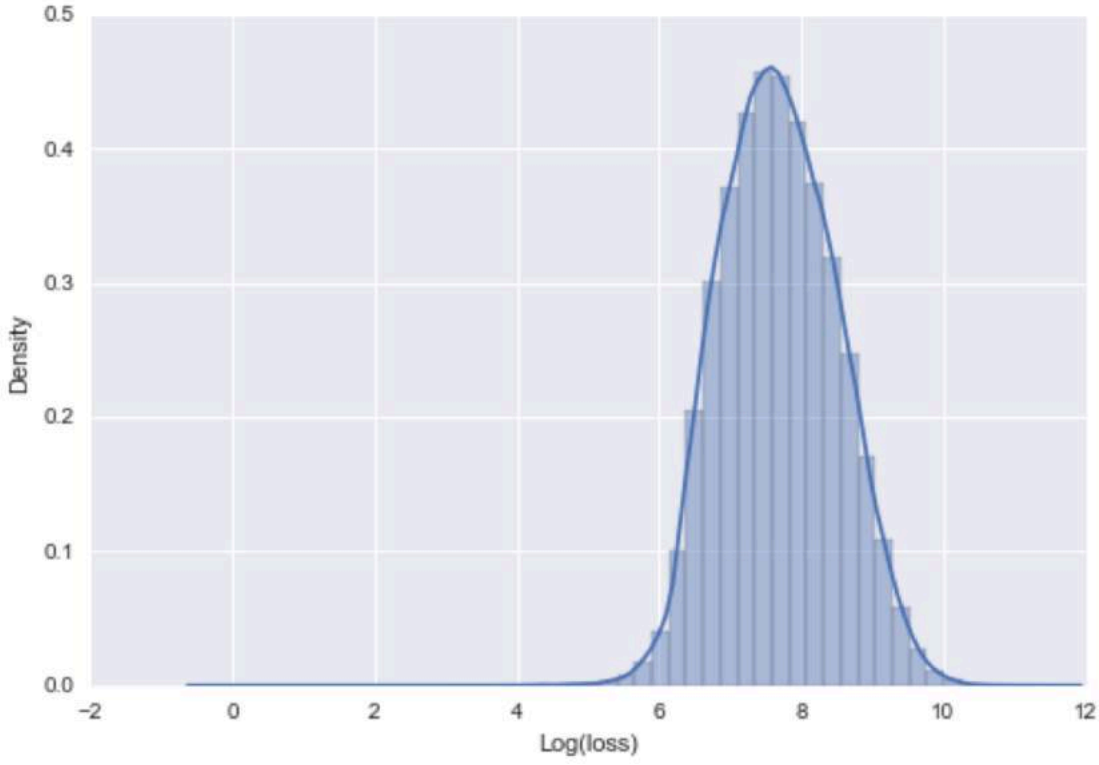
	cont7	cont8	cont9	cont10	cont11	cont12	cont13
0	0.335060	0.30260	0.67135	0.83510	0.569745	0.594646	0.822493
1	0.436585	0.60087	0.35127	0.43919	0.338312	0.366307	0.611431
2	0.315545	0.27320	0.26076	0.32446	0.381398	0.373424	0.195709
3	0.391128	0.31796	0.32128	0.44467	0.327915	0.321570	0.605077
4	0.247408	0.24564	0.22089	0.21230	0.204687	0.202213	0.246011

	cont14	loss
0	0.714843	2213.18
1	0.304496	1283.60
2	0.774425	3005.09
3	0.602642	939.85
4	0.432606	2763.85

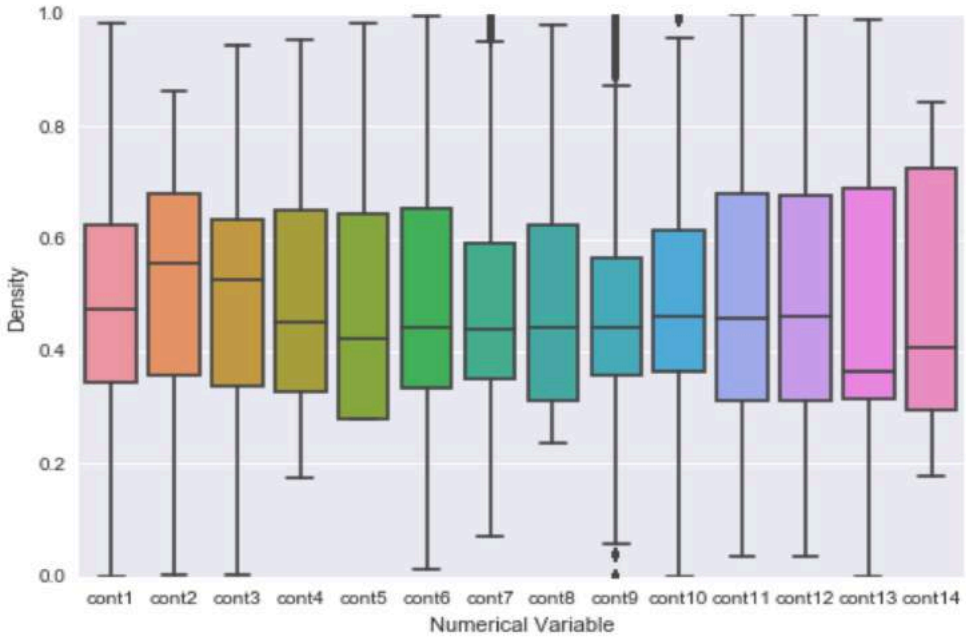
Histogram of claim loss is also shown below and we can see the data ranging from zero to 120000, but most of them group within small region. The distribution has heavily skewed to the left and has an extremely long right tail, indicating that we might need to scale on the target variable.

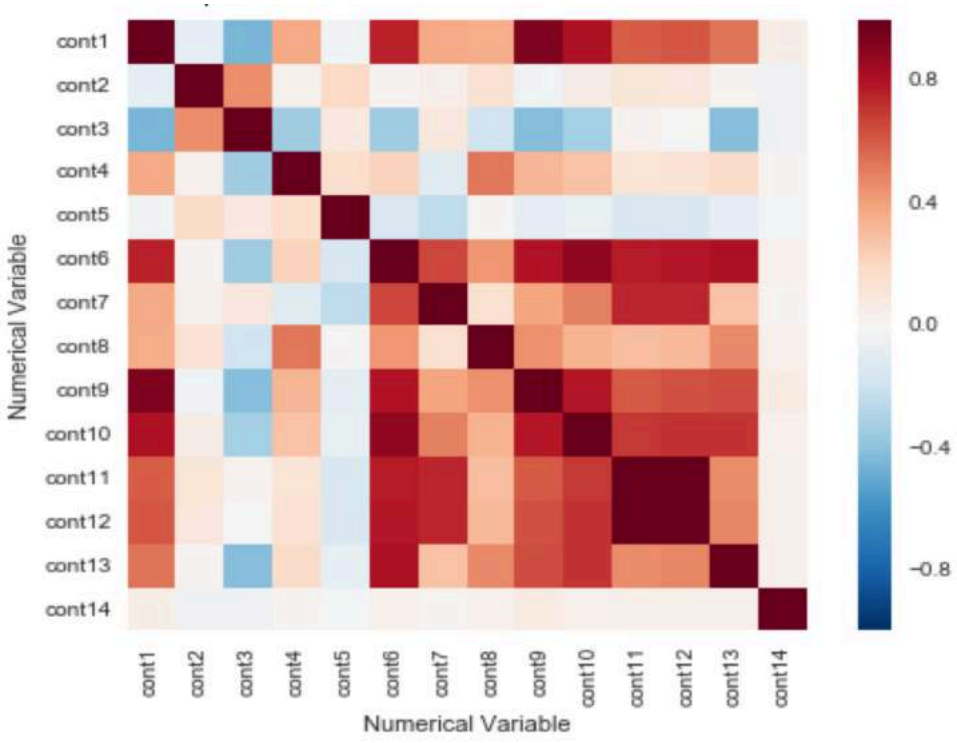


After log transformation on claim loss, we get the histogram that displays transformed distribution is perfectly following normal distribution. Apparently, we have taken the right move to transform the data, and scaling will reduce convergence time when training models on the large dataset like this.



The next step is to check predict variables, since most of them are categorical data and we will factorize them, what we need is to visually analyze the rest 14 numerical variables. First, we obtain boxplots of each variable in one graph and compare them together. As far as we observe, nearly all the variables have similar shape of distribution, they tend to be concentrate in the middle and symmetrically spread out. So no further scaling is needed before applying different algorithms. Then, heatmap is produced based on covariance matrix of these 14 variables. It is obvious that variables from **cont6** to **cont13** have larger correlation with each other, and **cont1** seems to have strong correlation with **cont6** to **cont13**, since Kaggle didn't provide additional information about what these variables are, what we can conclude so far is that they are definitely related and predict variables are not mutually independent.





Algorithms and Techniques

Since this is a regression problem, linear regression is designed for dealing with this kind of problem. Linear regression is used as benchmark algorithm due to it is easy to interpret and fast training time, however, it is unlikely to outperform to solve complex problem like this. Taking into account that the dataset has 130 predicting variables, we might need to try penalized method like Ridge and LASSO regression, they can shrink coefficient in magnitude to select significant variables and they can be trained extremely fast.

Also, large dataset like this is perfect for sophisticated methods like ensemble methods. Random Forest is performed, as well as fashionable extreme gradient boosting called XGBoost. The idea behind Random Forest is training several tree models based on randomly selected variables fitted in bootstrap training set and average the results of these trees together. Essentially, it ensemble many tree models together to deliver averaged results and this turns out very powerful, but obviously very time consuming. XGBoost is derived from Gradient Boosting Machine, which is a boosting tree based method similar to Random Forest, XGBoost takes it to the next level by adding well-defined regularization term to prevent over-fitting. The package available online is written in C++ and enable program to run on multiple cores, which makes training process very fast. Finally, we try to ensemble these methods together to get a new model. Hopefully, we can obtain significant improvement on the leaderboard.

Benchmark

Linear regression has been chosen as benchmark algorithm in this project. Noting that we will perform log transformation to the claim loss, which is our target variable. And we found out that target variable approximately followed normal distribution shown in the graph above, which meet assumption of ordinary least square. Also, linear regression is easy to interpret and fast-implemented to solve regression classification, making it a good benchmark algorithm in this case. Then, we run linear regression on the testing set based on 130 predicting variables, and leaderboard returns mean absolute error of **1268.28016**, in the meantime, the first ranking submission got mean absolute error of 1105.

The performance is more than I expected for a simple model like this. Even the most well tuned and sophisticated model is going to outperform linear regression by about 10%.

III. Methodology

Data Preprocessing

Like we mentioned before, our target variable has a huge range and heavily skewed to the left. Scaling is performed on the claim loss to reduce variation in the data and it approximately follows normal distribution after log transformation. This transforms the data to meet assumption of linear regression that we are going to use in the project and reduce convergence time during training process.

Since sklearn[2] package can't recognize categorical data automatically, we manually factorize values of 116 categorical variables into numbers. Then, we exam the rest numerical variables by boxplot and scatter plot. They seem like have similar ranges and some of them are highly correlated. In general, we don't need to select or omit any variables or observation in this stage, due to no abnormalities have shown up.

Ridge Regression

First, we have implemented ridge regression. Compared to linear regression, it adds penalty term in the model to control model complexity aiming at reducing overfitting shown below. The first term in the loss function is ordinary least square error computed in linear regression. α is penalty parameter and β stands for coefficients. Hence, larger alpha is going to result in small model and reduce complexity. To be mentioned that, ridge method will shrink small coefficient close to zero.

$$\hat{\beta} = \text{argmin}(X\beta - y) + \alpha\|\beta\|^2$$

Model is imported from sklearn[2] package (version 0.17.1). Before implementing ridge regression, we need to choose optimal alpha by cross validation error and we have try alphas in 0.1, 1 10 and 50, eventually we choose alpha of 0.1 and leave other parameter as default value, then run the whole dataset on ridge regression. The leaderboard returns mean absolute error of **1268.19379**, which makes slightly small improvement comparing to linear regression. This gives us brief idea that linear model might not make huge improvement no matter how we tune the model.

LASSO Regression

LASSO regression shares the similar idea of penalize complexity of model like ridge regression. The only different is that it penalizes absolute value of coefficient, which makes it more powerful in feature selection because it will shrink certain coefficient to zero.

$$\hat{\beta} = \text{argmin}(X\beta - y) + \alpha\|\beta\|$$

Model is also imported from sklearn[2] package. Similarly, we choose optimal alpha based on cross validation error selecting from 0.1 to 50 and result favors 0.1. After running the model on the whole training set, we get horrible result of **1459.67382** from leaderboard. This is a clear warning that we should not omit any variables in this case. Massive useful information have been dropped and significantly reduce model performance.

Random Forest

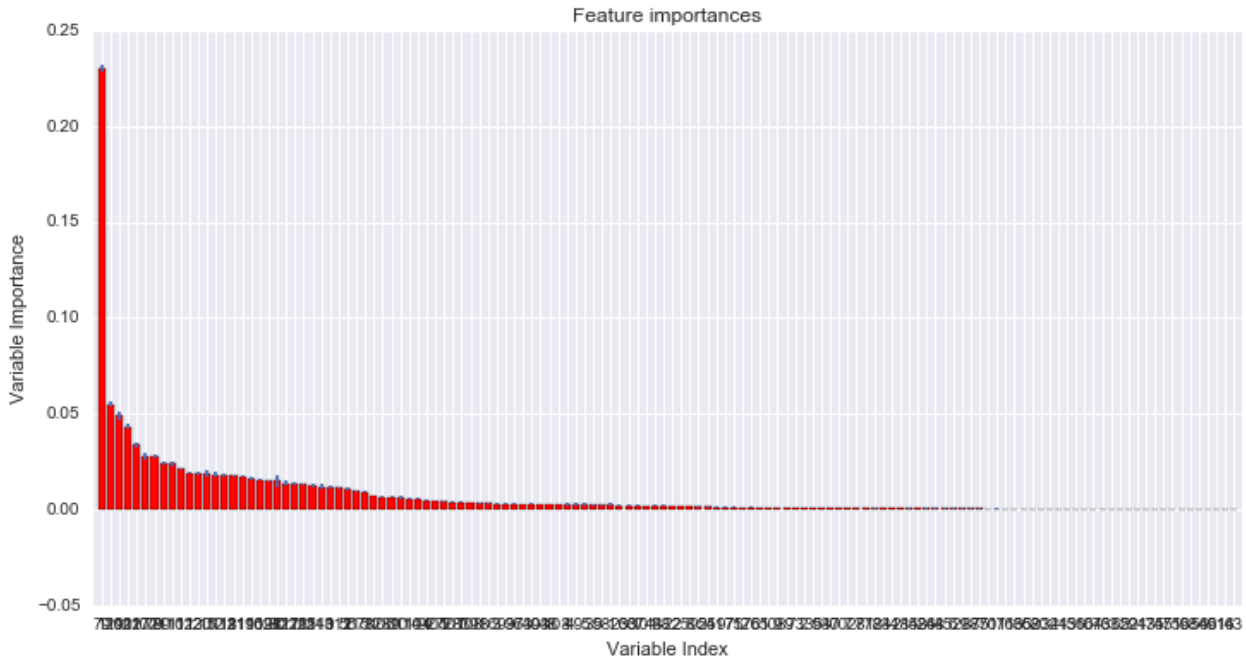
Random forest is very powerful in both regression and classification. It ensembles single tree models together and provides strong prediction. In most case, it tends to overfit the dataset due to flexibility. In this project, we have a relatively large dataset that is perfect for ensemble methods like random forest. We import module RandomForestRegressor from sklearn[2] package, there are several important parameters to control model complexity like number of trees and maximum depth. Take into consideration of limited computational capacity of local machine, we start

training random forest on the whole dataset by 50 trees and set other hyper-parameter as default value. It took **328.2011** seconds to train the data, and leaderboard result is **1204.55448**, which is a huge leap from what we get from penalized regression methods.

Then, let's try to make the model more complex. In this case, we fit 1000 trees without changing other parameters on the whole dataset. Unfortunately, my computer seems to be less capable of doing this and ends up finishing it in **6596.1319** seconds. However, the leaderboard results show 10-points improvement to **1193.30873**. Noting that it almost took 2 hours to train random forest with 1000 trees, this is sign to stop spending time on tune the model on the local machine and fit more computationally efficient model on the dataset.

Also, we have obtained feature importance list and plot based on random forest. From the graph, we can tell that the No. 79 variable is most important in predicting claim loss comparing to other variables. Since it is a categorical variable and Kaggle didn't provide us additional information about this, we cannot dig deeper about this. Our suggestion is that insurance company should pay close attention to first 1/3 variables that have the most influence on the prediction and consider them as indicator of high risk.

```
[ 79, 129, 100, 122, 117, 102,  78,  99, 110, 111, 123,  11, 120,
 118,  52, 119, 121, 116, 109, 128,  80, 127, 125, 126, 124, 113,
   0, 112,  56, 115,  71,  82, 106,  81,  90, 114, 104,  92, 105,
  72, 107,  83, 108,  91,  86,  93,  96,   3,  74,  30,  38, 103,
   4,  94,  49,   5,  98,   1,  36,  26,  35,  37,  43, 101,  48,
  22,  25,   8,  95,  24,  51,  97,  76,  65,  12,   9,  10,  75,
  89,  73,  39,  53,   2,   7,  40,  70,  18,  27,  87,  44,  28,
  23,  15,  42,  64,  84,  85,  29,   6,  88,  77,  50,  16,  17,
  13,  66,  41,  45,  60,  20,  58,  67,  32,  31,  46,  33,  59,
  62,  21,  57,  34,  47,  19,  68,  55,  54,  69,  61,  63,  14])
```



XGBoost

XGBoost stands for Extreme Gradient Boosting derives from Gradient Boosting Machine, which dominates machine-learning competitions in recent years. The main reason why it always outperforms is that XGBoost has well-define regularization term on the model complexity to overcome over-fitting on the testing set. Also, XGBoost is written by C++ and run extremely fast even on local machine.

XGBoost[3] package should be installed before fitting in model on Python. XGBoost is a complex model that has many variables available to tune. For a quick start and due to limited computational power of local machine, we set eta (learning rate) to 0.01, which is used to control over-fitting, then we set subsample to 0.8, meaning that model randomly select 80% data from training set to grow trees helping to prevent over-fitting as well. Min_child_weight is another important parameter to tune, in linear regression, value of this parameter stands for minimum number of instances needed to be in each node, and we set it to 3 in this case. The most important action to take is specifying learning task as linear regression by setting objective as reg:linear.

Before training XGBoost, training and testing set should be transformed into XGBoost Dmatrix, then we quickly start training XGBoost model by setting learning rate to 0.01 and learning task to linear regression with 1000 trees. Amazingly, it took only **372.8244** seconds to train and give us mean absolute error of **1142.68285**!!! Remember that we need nearly 2 hours to train random forest with 1000 trees, XGBoost outperforms random forest in both accuracy and training time.

IV. Results

Model Comparison

So far, we get all the results from leaderboard of each model we use as well as their computational time. It is a clear choice that XGBoost has a better performance trained within less computational time. Noting that the first-rank result on leaderboard is **1105**, and we believe a well-tuned XGBoost is going to improve our result to around 1120. Noting that it takes about 6 minutes to train a single XGBoost on the local machine, so tuning parameter on the large dataset might take quite a long time, since the model already has very strong predictive power, we didn't tune XGBoost in this project and add it as future work expected to be done on the AWS.

	MAE	Training Time
Linear Regression	1268.28016	2.458
Ridge Regression	1268.19379	0.4847
LASSO Regression	1459.67382	0.8525
Random Forest(n=50)	1204.55448	328.2011
Random Forest(n=1000)	1193.30873	6596.1319
XGBoost	1142.68285	372.8244

Model Ensemble

Then, we decide to ensemble these methods that we have used so far. The main idea behind this is that each model captures features and variation of data from different from different aspect. Ensemble method is trying to combine these abilities together to form a strong learner. It is essentially the same idea with mechanism of random forest. In this case, we are going to use Ridge regression as representation of linear regression methods, random forest as representation as ensemble methods and XGBoost as boosting methods. Essentially, boosting method is still a tree-based ensemble method similar to random forest, so high correlation is expected between the predictions generated from random forest and XGBoost.

In general, predictions that have smaller correlation tend to have better prediction power after ensemble, however, ensemble method doesn't guarantee improvement every time. First, let's compute prediction of XGBoost with prediction of Ridge regression and random forest separately using Pearson correlation, it also returns the p-value for testing non-correlation.

As expected, correlation between XGBoost and Ridge regression is less than correlation between XGBoost and random forest. Noting that, correlation between XGBoost and random forest is around 0.95, which is close to 1 indicating that they are highly correlated. At this stage, the correlation indicates that it is better to combine Ridge regression with XGBoost, or ensemble these three methods together. Rather than combining XGBoost and random forest, since they might model data from similar aspect.

	Pearson Correlation	P value
XGBoost vs Ridge	0.876566	0
XGBoost vs Random Forest	0.957972	0

We try six combinations to evaluate effectiveness of ensemble method based on leaderboard score. Also, we need to vote more weight on better-performance model like XGBoost, and assign less weight on less-performance weight like Ridge regression or random forest. Each combination is shown below. Ensemble method doesn't work at this time, even the lowest mean absolute error is worse than simple XGBoost predictor. It is clear that the high weight that XGBoost has in the combination, the better performance of whole combination will get. Also, this result suggests us we might need to ensemble different models that at least have similar performance with XGBoost, and combine less performance models might not work in this regression problem.

$$ensemble1 = 0.9 * XGBoost + 0.1 * Ridge\ Regression$$
$$ensemble2 = 0.8 * XGBoost + 0.2 * Ridge\ Regression$$
$$ensemble3 = 0.5 * XGBoost + 0.3 * Random\ Forest + 0.2 * Ridge\ Regression$$
$$ensemble4 = 0.5 * XGBoost + 0.4 * Random\ Forest + 0.1 * Ridge\ Regression$$
$$ensemble5 = 0.6 * XGBoost + 0.3 * Random\ Forest + 0.1 * Ridge\ Regression$$
$$ensemble6 = 0.9 * XGBoost + 0.1 * Random\ Forest$$

	MAE
Ensemble1	1144.51408
Ensemble2	1149.72216
Ensemble3	1152.72033
Ensemble4	1153.94942
Ensemble5	1148.08036
Ensemble6	1142.89203

Each combination of ensemble model delivers excellent result, however, none of them surpass single XGBoost model. It is interesting to see that ensemble model is going to perform better if weight on XGBoost is higher. Like we mentioned before, rule of thumb of ensemble method is that combine predictions that are less correlated is likely to result in better improvement. However, it seems like it doesn't work in this regression problem. We first try to combine XGBoost and Ridge regression due to lower Pearson correlation between predictions, the results are even getting worse while weight on Ridge regression gets higher. This suggests that we should assign more weight on stronger learner and less weight on weak learner.

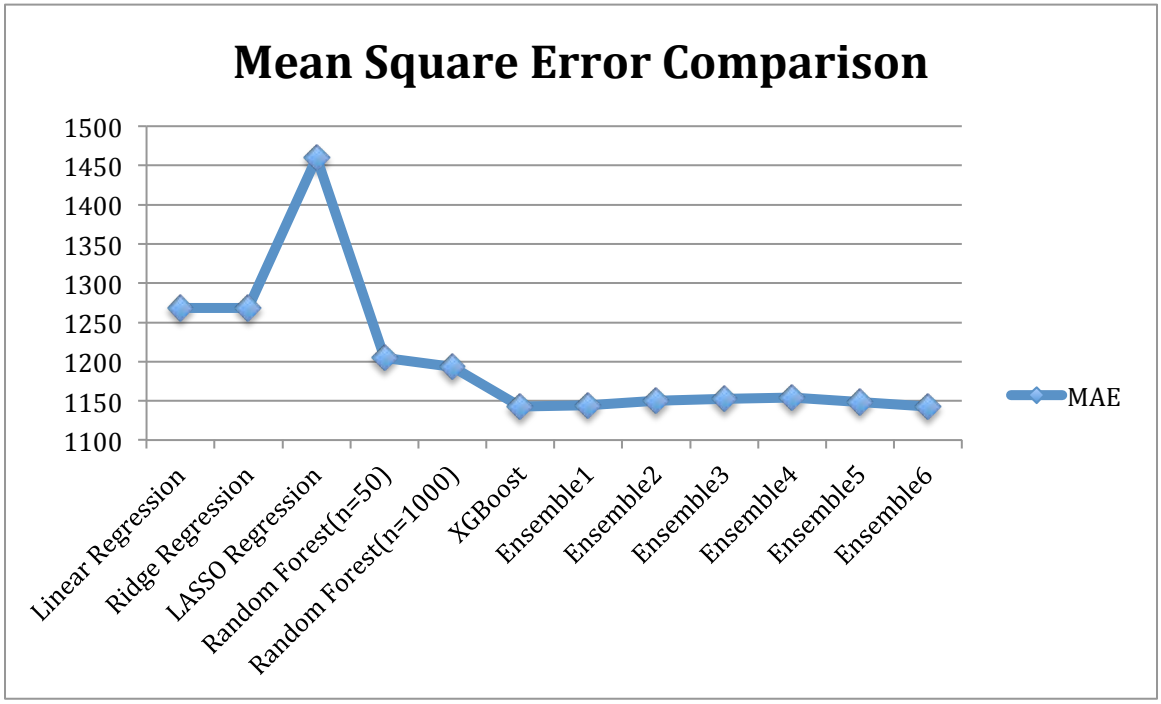
Next, we add Random Forest into combination and limited weight on Ridge regression not greater than 0.2. Similar trend can be observed that ensemble model tend to perform better if weight on XGBoost is increasing, supporting what we find that optimal combination should assign more weight on stronger learner. Last, we try combination with only XGBoost and Random Forest, in this case, it performs surprisingly well when we set weight on XGBoost to 0.9 and 0.1 to Random Forest. The last model provides similar prediction power compared to single XGBoost model.

In terms of robustness, linear model like Ridge regression is relatively simple formed without much flexibility, so it is very stable in general. Random Forest introduces randomness by fitting on the bootstrap training set and randomly selecting variables. The prediction tends to change dramatically when overall number of trees is set to a small value. In this case, we fit the model with 1000 trees, which ensure stability of final model. Similarly, we have trained 1000 trees in XGBoost model, so that robustness of model is guaranteed. Since each individual component learner is robust, we are sure that combination of these models are also robust and will deliver stable result when big change occurs in the training set.

Justification

In all, our best performance model is XGBoost so far, and it delivers amazing result of **1142.68285** comparing to benchmark result of **1268.28016**, in the meantime, the first result in the leaderboard is around **1105** right now. XGBoost has produced state-of-art prediction power trained within small amount of time. We expect ensemble models might provide better results, however, it doesn't work in this regression problem, but it suggests that ensemble model tend to improve if we combine strong learners and assign more weight on better models.

V. Conclusion and Future Work



In this project, we challenged Kaggle competition aiming at predicting claim loss based on 130 variables provided. This is my first Kaggle competition and also my first time to deal with big dataset that has over 180000 observations. Computational efficiency becomes more important in this case, it is usual to run several hours to train a sophisticated model on local machine. Eventually, XGBoost outperforms other methods and becomes the obvious choice due to strong performance and fast training time.

We started with applying exploratory data analysis to get brief idea what features we have in the dataset after factorizing every categorical variables provided by Kaggle. Log transformation has been applied on target variable to make it approximately follow normal distribution and contribute to less training time. None of the variables have been omitted in this case, because LASSO regression showed bad result if variable selection is performed. Then, penalized regression and tree-based ensemble methods like random forest and XGBoost are fit on the training set. Additionally, we try to ensemble different model to form a strong learner in this project, however, it turns out that new model is actually weaker than single XGBoost regressor.

Since this is my first time dealing with such big dataset. Some problems occur unexpectedly, I first thought my computer went down when fitting random forest with 1000 trees. This provides me deeper understanding of why people design distributed system in order to reduce training time. Also, this competition is a real-world problem that could help insurance company control potential risk, which makes me feel I am doing something meaningful rather than finishing an assignment. Joining Kaggle community and reading several insightful post on the forum

significantly increases my learning process. I am totally hooked by machine learning and definitely will go deeper to improve model performance in this project.

As to the future work, I think we should do feature engineering on the variables. From variable importance plot, we can see that some of variables have significant importance compared to others. Polynomial or quadratic terms might increase prediction power if they are construct in the right way. Beyond this, more advanced model like neural network could get even better result than XGBoost in a big dataset like this. I will learn how to run Mapreduce on Amazon Web Service to train neural network. Finally, we may consider ensemble or stack neural network as well as well-tuned XGBoost to achieve a better result.

Reference

[1] Allstate Claims Severity, Kaggle. https://www.kaggle.com/c/allstate-claims-severity

[2] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[3] Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." arXiv preprint arXiv:1603.02754 (2016).