Capstone_Report

November 4, 2017

1 Udacity Machine Learning Nanodegree

1.1 Allstate Insurance Claims Prediction – Capstone Project Report

Xiang Cao 10-10-2017

2 Part 1. Definition

2.1 Project Overview

As a important domain of Machine Learning, supervised learning has a wide variety of applications in different areas including computer vision, natural language processing, recommendation system, credit risk prediction, etc.

The essence of Machine Learning is to learn hidden pattern underneath data, data could be unstructured like image, language, or tabular data like stock trading data, customer behavior, etc., or a mixture of them. According to the pattern / knowledge it learnt, we can customize our product and service to people more accurately and efficiently.

In finance/insurance area, there are large amount of user data, with which we can customize finance/insurance product to increase company revenue and improve user experience. Machine Learning provides promising methods to do this.

There are multiple machine learning competitions of this domain on Kaggle. Here are some examples.

- Give Me Some Credit. Build models to predict the probability of default, to determine whether or not a loan should be granted. This is a classification problem.
- Allstate Claim Prediction Challenge. An earlier competition held by Allstate. Predict bodily injury liability insurance claim payments based on the characteristics of the insured's vehicle. This is a regression problem.

In this project, I solved a competition held by Allstate on Kaggle, which aims to develop automated method to predict insurance claims cost, and hence severity. The datasets are provided by Allstate on Kaggle. It's open to public here.

2.2 Problem Statement

The data includes the historical claims cost and customer information for each claim. Each record contains both cateorical and continuous features, and the target is the numerical cost of the claim which is continuous. Thus this is clearly an regression program.

In this project, I am going to predict claim cost using different machine learning models (mainly Gradient boosting and Neural Network).

- **Datasets and Inputs**

The training set has 188318 records and 132 columns, including unique ID, 116 categorical features, 14 continuous features, and target. The target is a continuous variable which indicates the loss of each claim. All the features are anonymous. The testing set has 125546 rows and columns except target.

In this project I split training set into training and validation to train and evalution models. 5-fold cross validation will be implemented of each model. That is, for each fold, 80% of the data is used as training set, the rest 20% is used as validation set.

- **Solution**

Data pre-processing is done first, including checking missing values, data transformation, etc. Then feature engineering, including check the interaction of features, two-way interactions, and some three-way interactions. Multiple supervised models are implemented in this project, including regression, gradient boosting tree, neural network, etc.

The project mainly has the workflow:

- Expolorary data analysis. Check the distribution of continuous and categorical variables, check the correction between continutous variables.
- Feature engineering. Create features interactions, select the good ones with the original features to train models. Also, we may transform the target, or even customize the cost function.
- Split the data into training set and validation set (cross validation). Building machine learning models, tuning parameters.
  - The model includes regression, glmnet, gradient boosting tree, neural network, etc.
- Compare and pick the best model for each method. Build two layer stacking model. These single models consititute the first layor of the stacking model. The second layer chooses either Gradient boosing or Keras nerual network.

2.3 Evaluation Metrics

The cost need be predict is continuous variable. For a regression problem, we usually use Mean Square Error(MSE) or Mean Absolute Error(MAE) as metric. According to the competition evaluation criteria, MAE is used. We are going to build models to minimize MAE.

$$MAE = \frac{1}{n}\sum_{i=1}^{\infty} \mid y_i - \hat{y}_i \mid$$

Both MSE and MAE are common used metrics for continuous variable. Compared to MSE, MAE is a more robust measure of the variability. That is, MAE is more resistent to outliers. In this problem, to decrease the impact of insurance claims outliers, MAE is a good choice.

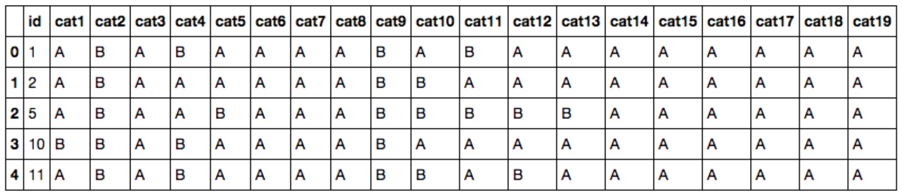
3 Part 2. Analytics

3.1 1. Data Exploration

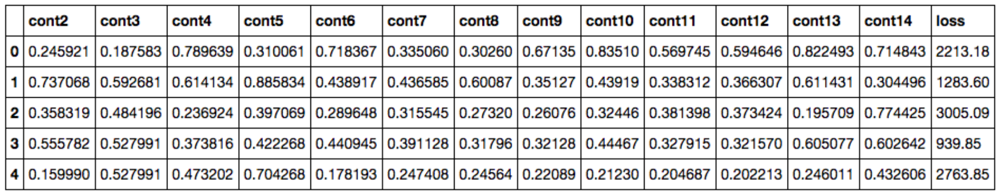
For the data exploration, please check the file `Exploratory_Data_Analysis.ipynb`. The training set has 188318 records and 132 columns, including one target column `loss` and `id` coulumn which is used as unique key each row. Here is the summary:

- In the 130 features, there are 116 categorical variables and 14 continuous variables
- For all the rest 130 columns, there is no missing value.
- For all continuous variables, the maxima is less than 1, and minima is great than 0. It looks like the data is standardized/scaled by the provider.
- 72 out of 116 categorical variables have 2 unique value. Variable `cat116` has the maximal number of unique value 326.
- The target variabls is pretty skewed. We would tranform it before fit models.
- The training set and testing set have similar distribution, which means the model we trained on training set through cross-validation could also be used to predict testing set.
- Some distribution of several continuous variable have multiple peaks, which look like were converted from categorical variable.

Here is a sample of the data, including the head 5 rows of first 20 columns and last 14 columns.



	id	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	cat10	cat11	cat12	cat13	cat14	cat15	cat16	cat17	cat18	cat19
0	1	A	B	A	B	A	A	A	A	B	A	B	A	A	A	A	A	A	A	A
1	2	A	B	A	A	A	A	A	A	B	B	A	A	A	A	A	A	A	A	A
2	5	A	B	A	A	B	A	A	A	B	B	B	B	B	A	A	A	A	A	A
3	10	B	B	A	B	A	A	A	A	B	A	A	A	A	A	A	A	A	A	A
4	11	A	B	A	B	A	A	A	A	B	B	A	B	A	A	A	A	A	A	A



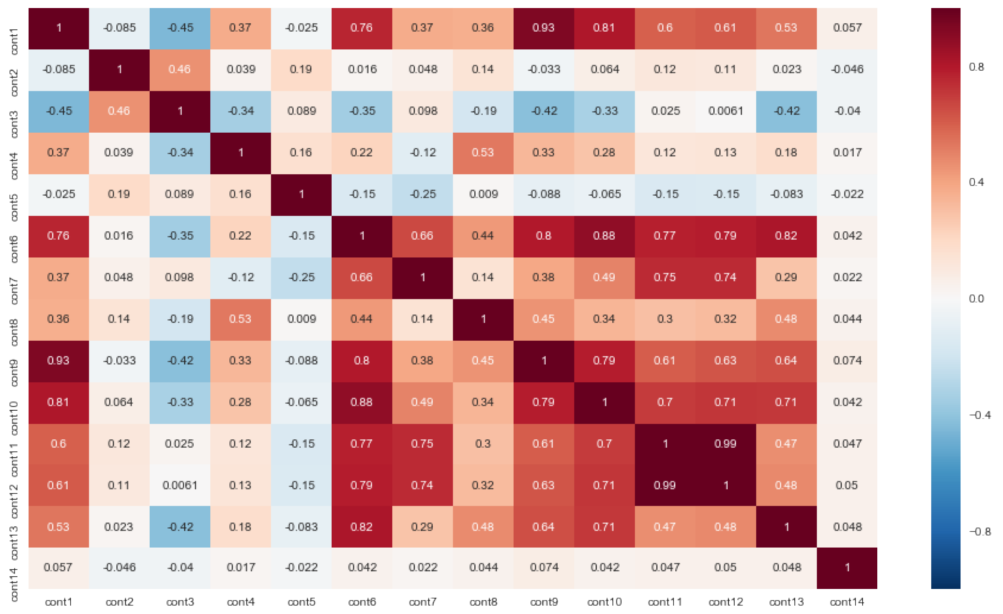
	cont2	cont3	cont4	cont5	cont6	cont7	cont8	cont9	cont10	cont11	cont12	cont13	cont14	loss
0	0.245921	0.187583	0.789639	0.310061	0.718367	0.335060	0.30260	0.67135	0.83510	0.569745	0.594646	0.822493	0.714843	2213.18
1	0.737068	0.592681	0.614134	0.885834	0.438917	0.436585	0.60087	0.35127	0.43919	0.338312	0.366307	0.611431	0.304496	1283.60
2	0.358319	0.484196	0.236924	0.397069	0.289648	0.315545	0.27320	0.26076	0.32446	0.381398	0.373424	0.195709	0.774425	3005.09
3	0.555782	0.527991	0.373816	0.422268	0.440945	0.391128	0.31796	0.32128	0.44467	0.327915	0.321570	0.605077	0.602642	939.85
4	0.159990	0.527991	0.473202	0.704268	0.178193	0.247408	0.24564	0.22089	0.21230	0.204687	0.202213	0.246011	0.432606	2763.85

3.2 2. Exploratory Visualization

Continuous Variables Correlation

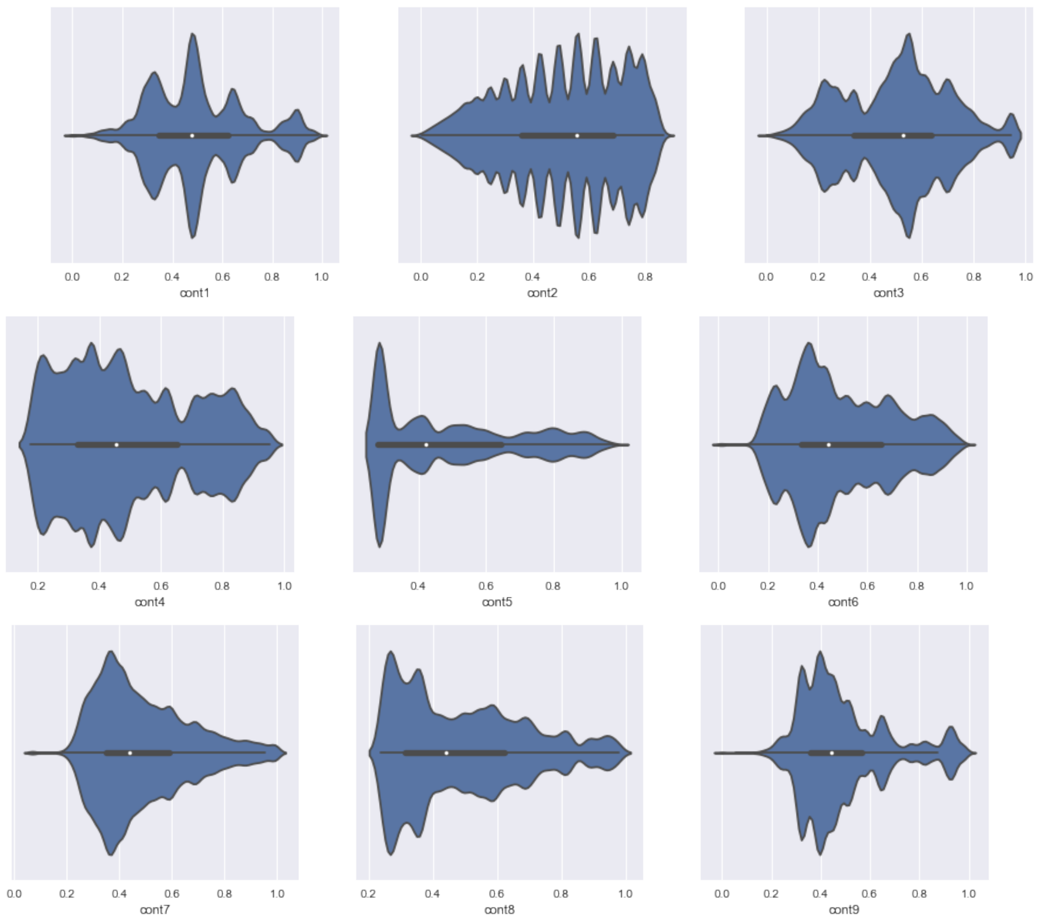
There are few variables highly correlated. For example, the correlation between `cont1` and `cont9` is 0.93, and correlation between `cont11` and `cont12` is 0.99. In Statistics, for the purpose of interpretation we need remove highly correlated variable to solve the multicollinearity problem. However in this project the purpose is prediction, it's not mandatory to remove correlated variables. We would include all feature in order to get more information. One possible disadvantage of more features is overfitting. We would use methods like regularization to mitigate this issue.

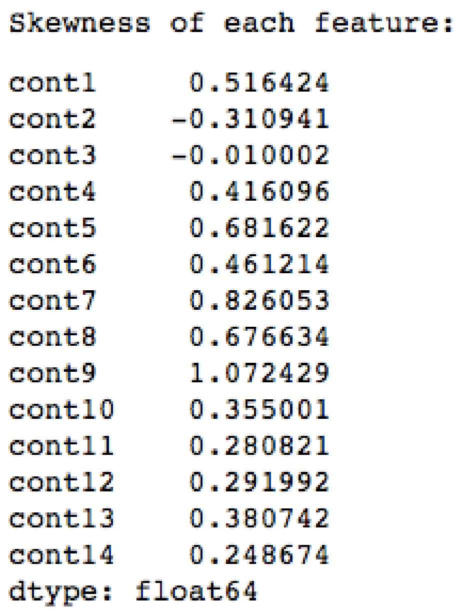
Distribution of Continuous Variables



	cont1	cont2	cont3	cont4	cont5	cont6	cont7	cont8	cont9	cont10	cont11	cont12	cont13	cont14
cont1	1	-0.085	-0.45	0.37	-0.025	0.76	0.37	0.36	0.93	0.81	0.6	0.61	0.53	0.057
cont2	-0.085	1	0.46	0.039	0.19	0.016	0.048	0.14	-0.033	0.064	0.12	0.11	0.023	-0.046
cont3	-0.45	0.46	1	-0.34	0.089	-0.35	0.098	-0.19	-0.42	-0.33	0.025	0.0061	-0.42	-0.04
cont4	0.37	0.039	-0.34	1	0.16	0.22	-0.12	0.53	0.33	0.28	0.12	0.13	0.18	0.017
cont5	-0.025	0.19	0.089	0.16	1	-0.15	-0.25	0.009	-0.088	-0.065	-0.15	-0.15	-0.083	-0.022
cont6	0.76	0.016	-0.35	0.22	-0.15	1	0.66	0.44	0.8	0.88	0.77	0.79	0.82	0.042
cont7	0.37	0.048	0.098	-0.12	-0.25	0.66	1	0.14	0.38	0.49	0.75	0.74	0.29	0.022
cont8	0.36	0.14	-0.19	0.53	0.009	0.44	0.14	1	0.45	0.34	0.3	0.32	0.48	0.044
cont9	0.93	-0.033	-0.42	0.33	-0.088	0.8	0.38	0.45	1	0.79	0.61	0.63	0.64	0.074
cont10	0.81	0.064	-0.33	0.28	-0.065	0.88	0.49	0.34	0.79	1	0.7	0.71	0.71	0.042
cont11	0.6	0.12	0.025	0.12	-0.15	0.77	0.75	0.3	0.61	0.7	1	0.99	0.47	0.047
cont12	0.61	0.11	0.0061	0.13	-0.15	0.79	0.74	0.32	0.63	0.71	0.99	1	0.48	0.05
cont13	0.53	0.023	-0.42	0.18	-0.083	0.82	0.29	0.48	0.64	0.71	0.47	0.48	1	0.048
cont14	0.057	-0.046	-0.04	0.017	-0.022	0.042	0.022	0.044	0.074	0.042	0.047	0.05	0.048	1

title

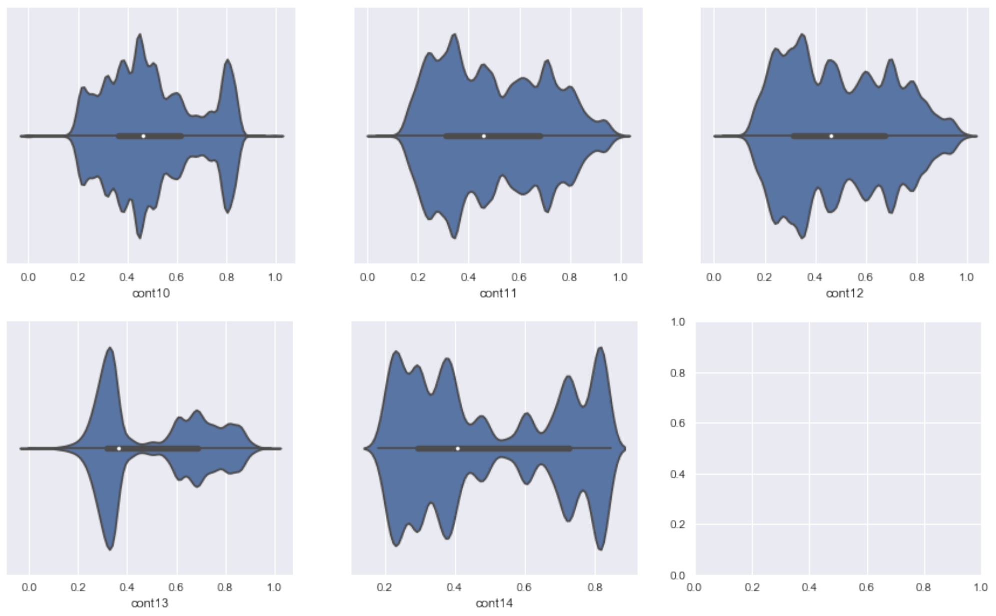




```
Skewness of each feature:

cont1      0.516424
cont2     -0.310941
cont3     -0.010002
cont4      0.416096
cont5      0.681622
cont6      0.461214
cont7      0.826053
cont8      0.676634
cont9      1.072429
cont10     0.355001
cont11     0.280821
cont12     0.291992
cont13     0.380742
cont14     0.248674
dtype: float64
```

title



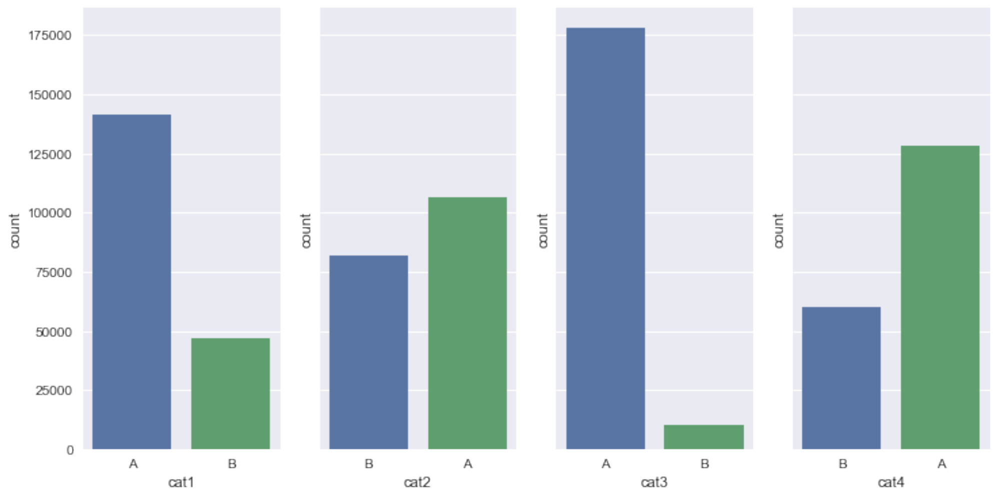
As we mentioned before, some variables may converted from categorical variables.

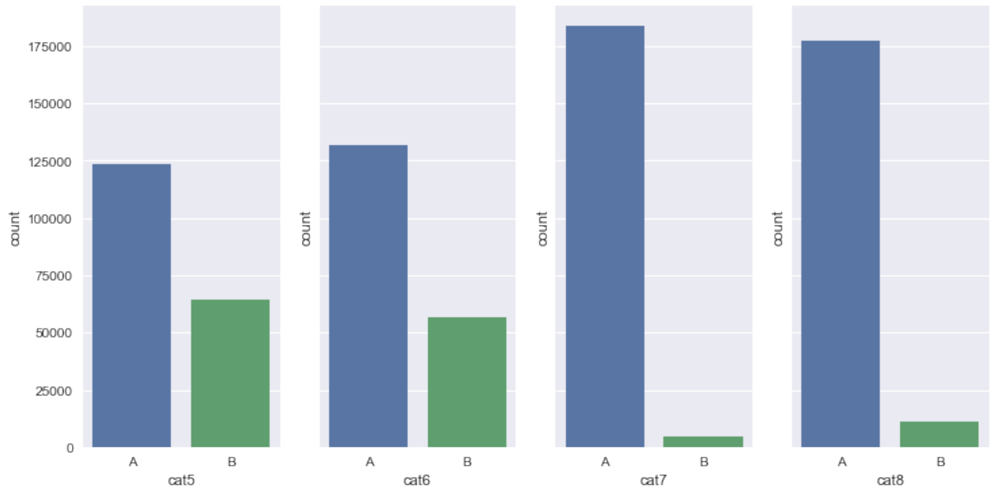
Skewness

The skewness of continuous variables are not too bad. We also did Box-Cox transformation on the continuous variables whose skewness is greater than 0.25. Since it didn't improve the performace obviously, we didn't do this in the final models.

Categorical Variables

Most of the categorical variables only have two unique values. Here we show the bar plot of first several variables.





Training vs. Testing

We use machine learning technique to check whether training set and testing set are consistent. The idea is to combine training set and testing set, use one classifier like logistic regression seperate them. If the performance of the classifier is similar to random guess (AUC approximates 0.5), then it means training set and testing set are quite similar. For more details please refer Adversarial Validation. The AUC of logistic regression classifier is 50.1%, which means our classifer cannot find difference between training set and testing set. It indicates training set and testing set are drawn from the same distribution.

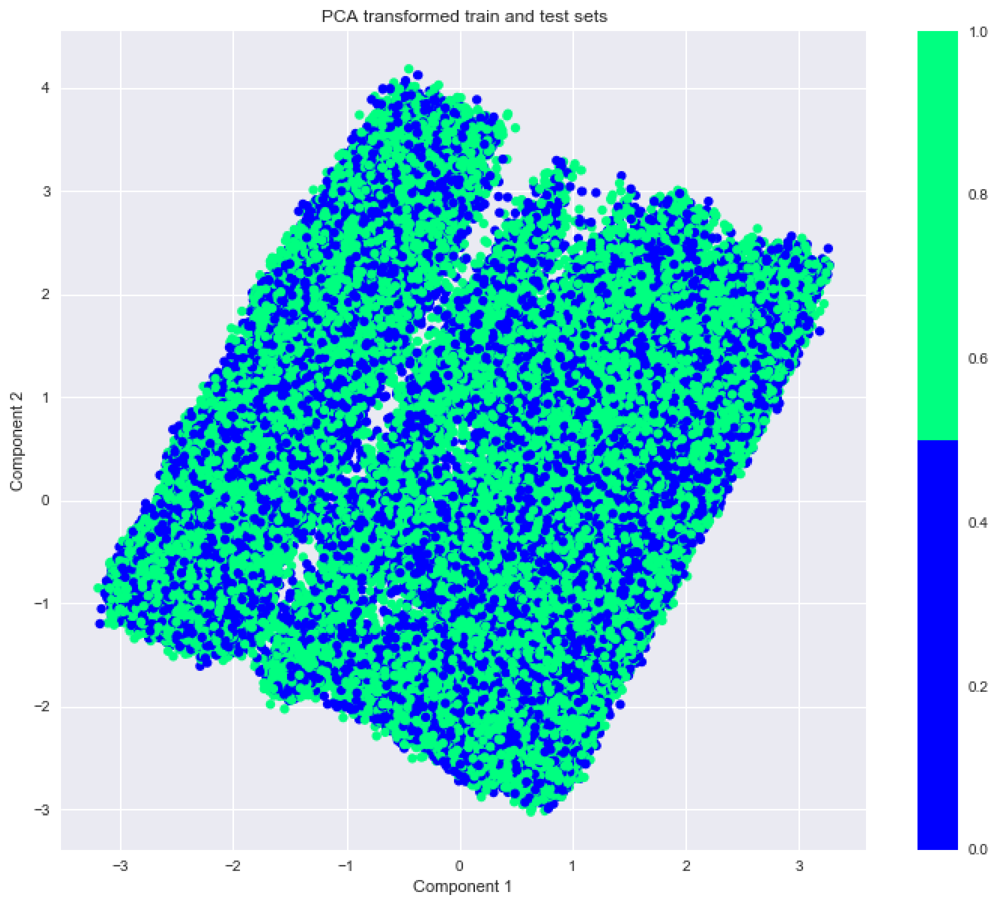
Also, we use compare the first two PCA components of training vs. testing set. In the scatter plot below, the training set and testing set mixed with each other, which is consistent with the AUC of logistic regression.

3.3 3. Algorithms and Techniques

For the regression problem, there are mutiple models that can used to predict continuous variables. I tried several different models, also according to others' experience, the Xgboost and Neural Network performs the best. Also, I used Elastic Net(LASSO and Ridge regression) as a baseline.

Elastic Net

Elastic Net is a regularized method that add both L1(LASSO) and L2(Ridge) regularization


PCA transformed train and test sets
Component 1
Component 2
−3
−2
−1
0
1
2
3
4
0.0
0.2
0.4
0.6
0.8
1.0

title

term to the loss function of linear regression or logistic regression. The more parameters a model have, the more probable overfitting play a rule. Compared the simply a linear regression and logistic regression, LASSO and Ridge regression can be used to decrease overfitting. In most of the situations it performs better than regression without regularization term by trading offa small increase in bias for a large decrease in variance.

Here is the cost function. Note MAE is the metric we used in this problem $MAE = \frac{1}{n}\sum_{i=1}^{\infty} \mid y_i - \hat{y}_i \mid$

LASSO:

$$MAE + \lambda \sum_{j=1}^{p} \mid \beta_j \mid$$

Ridge Regression:

$$MAE + \lambda \sum_{j=1}^{p} \beta_j^2$$

Due to the different fomula, LASSO and Ridge has slightly different properties. Using LASSO, the weights can be shrinked to as small as zero. Thus it can be used to select features. That is, the important features have large weights, while the unimportant ones have very small weights or even zero. Conversely, using Ridge regression the weights are always greater than zero. The performance of LASSO and Ridge regression are quite similar.

The drawback of LASSO is that when there are highly correlated features, it only select one feature and ignore others. Elastic Net can solve this problem. Elastic net includes both LASSO and Ridge regression. It has a hyperparameter you can tune to adjust how much emphasis on LASSO or Ridge regression.

We used LASSO since it trains pretty fast, and had decent performance in many different cases. Also, it has only parameters λ need tune. We used it as our benchmark model.

Xgboost

Boosting tree is an ensemble method. It builds sequential trees, each tree is based on the information of previous tree, thus each tree is a weak learned. Boosting combines those subsequently learned from misclassified training samples to improve the performance of ensemble models. There are different variants of boosting method.

1) AdaBoost

Adaboost is short for Adaptive Boosting. When training each weak learner, Adaboosting uses the complete traning set. For each learner the weights of all training samples are updated based on the mistakes of previous learner. When training the first tree, weights of all samples are equal. When training subsequential learner, we increase the weights for the misclassified samples, and lower the weights for the correctly classified samples. Also based on the error rate of each learner, we computed a weight for learner. The weight for learner would be used to compute(update) the weight vector for all training samples entry for next learner. For more details please check the book Python Machine Learning

Eventually compute the final prediction using the formula below, which is a weighed combination of weak learners.

$$\hat{y} = (\sum_{j=1}^{m}(\alpha_j \times predict(C_j, X)) > 0)$$

2) Gradient Boosting

Gradient Boosting is one of the best machine learning for tabular data. Different from Adaboost where the weak learner on all training set, in Gradient Boosting weak learner only trains on the residuals of previous learner. For regression with square loss, residual actually is negative gradient, so fit learner on residual is updating the model using gradient descent. That's why we call it Gradient Boosting.

In Adaboost, the "shortcomings" are identified by high-weight data points; while in Gradient Boosting, the "shortcomings" are identified by gradients. Both high-weight data points and gradients tell us how to improve the model. For more details, please check this NEU lecture note. Adaboost emphasizing misclassified points by changing the weights of training set, and keep the target constant. Gradient Boosting emphasizing misclassified points by changing the target to residuals for next training procedure (the misclassified points have larger residual). Both ways would improve the model to predict misclassified points (or points with large residual) correctly.

In Adaboost, the weak learner is a decision stump(tree with depth one), while the depth of tree in Gradient Boosting is a parameter need be tuned. But essentially the learner could be any classifier, like decision tree, logistic regression, SVM, etc.

3) XGBoost

XGboost is a optimized distributed gradient boosting library which implements machine learning algorithms under Gradient Boosting framework. Compared to the regular Gradient Boosting tree implementation, it has several advantages:

- Regularization.

Standard Gradient Boosting has no regularization. XGBoosting added regularization in cost function, we can tune `lambda` (L2 regulariztion) and `alpha`(L1 regularization) to reduce overfitting.

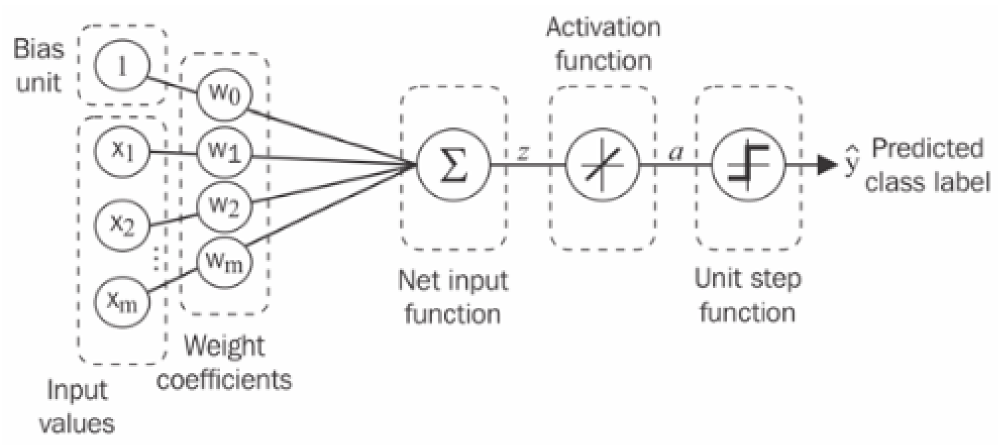
- Parallel and distributed implementation.

Boosting has sequential trees where each tree is build based on the information of previous tree, so it cannot be parallized at tree granularity. XGBoost implemented parallel computation at feature granularity. When splitting a node, we need choose the feature and the split point of that feature which maximize information gain. That is, at each node we need sort all features. This can be done by parallel and distributed computation.

XGBoost works very good for both classification and regression. For most of the tabular data, XGBoosting could achieve the best performance through fine tuning parameters. In this case, most of the categorical only two or a few unique values, which can be handled by XBGoost easily.

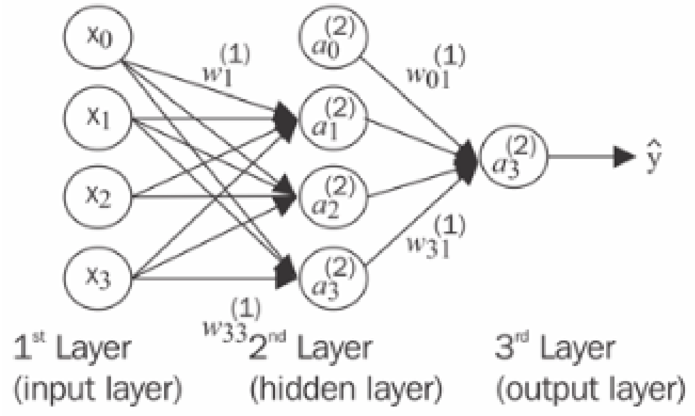
Neural Network

Deep learning(deep neural network) gets great success in machine learning. We also build feedforward and fully-connect neural network by Keras. Here is a simple single-layer neural network and a multi-layer neural network (image is from the book [Python Machine Learning]):



In the single-layer neural network, X is the input values, W is the weight coefficients, ϕ is activation fucntion. According to forward propagation, the output equals $\phi(w^T x)$. The weights are the parameterd would be learnt through back propagation. We can see if the activation function is simply $y = x$, then the network actually is equivalent to linear regression; if the activation function $\phi(z) = \frac{1}{1+e^{-z}}$, then the network is equivalent to logistic regression.

We can make the network more complicated by add more hidden layers like the image below. It gives the network power to learn more complicated patterns from data.



In this case a shallow fully connnected network also have a big performance. Since Neural Network and Gradient Boosting are different types of models, we can improve the final prediction via model ensemble/stacking.

3.4 4. Benchmark

I use the simple and straight forward model LASSO as benchmark. I build the model and tuned the parameter on training set, and submit the predicted result of testing set and got a Kaggle's public leaderboard score.

R has a very good library `glmnet` which can do cross validation to tune the regularization parameter λ in LASSO and Rridge regression. It provides a default list of parameters and returns the best one through cross validation. According to my experience, it performs better than the `ElasticNet` in Python Sklearn.

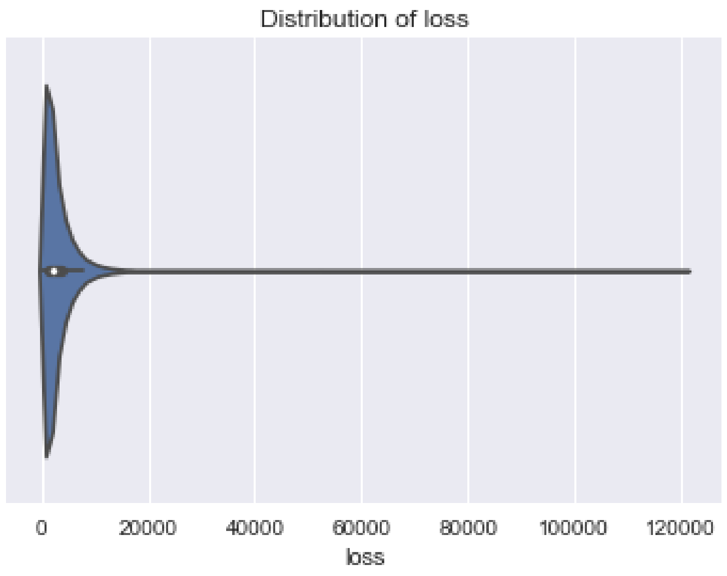
I did 3-fold cross validation on the training set, and got 1234.18 on Kaggle's public leaderboard. Please check the script `lasso_benchmark.R`.

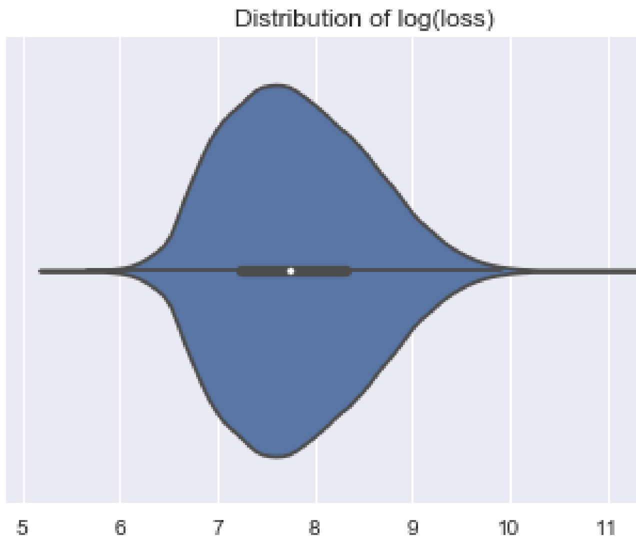
4 Part 3. Methodology

4.1 1. Data Preprocessing

Target variable transformation

The skewness of target variable is 3.79. It's quite non-normal distributed. The logarithmic transformation made the target looks normal, and got a better predict performance than the original target. Also, according to the discussion on Kaggle, take a shift on the target before logarithmic transformation helps a little. The shift can be regarded as a hyperparameter. Here I set shift to be 200, which means add 200 to all target value before transformation. Note when compute prediction, we need minus 200 after exponential transformation.





Catgorical variable

Theoretically speaking, the categorical should be processed by one-hot encoding, we can do this in `pandas.get_dummies`. When the order of categorical variable value doesn't matter, one-hot encoding stores all the information of the original categorical variables.

However, in our data set some categorical variables have hundreds of different values, thus one-hot encoding would generate a very sparse matrix. There is another encoding method `pandas.factorize` which encodes categorical variable into integer values. Boosting cannot handle categorical variable with too many different values, thus `pandas.factorize` (or `sklearn.LabelEncoder`) is a good choice. Check this discussion on Kaggle why XGBoost works good for this encoding. For Neural Network, we still create dummies variable for categorical variables.

Continuous variable

Although all the continuous variables are within 0 and 1, we still used `StandardScaler` of sklearn to standardize the data (remove the mean and divide by standard deviation)

4.2 2. Implementation and Refinement

LASSO

The first model is the LASSO implemented in R as benchmark. The library has its own cross validation to choose the best regularization parameter `lambda`. So it's straight forward and no need to create folds by myself. For the benchmark model, we only did the basic data preprocessing without feature engineering. For the script please check `lasso_benchmark.R`. The benchmark score of this model on Kaggle leaderboard is 1234.18.

XGBoost

For XGBoost, we first build a model with experienced default parameters. The purpose the XGBoost starter is to have a look at how well the performance of XGBoost could have. The quick starter got the score of 1124.25 on Kaggle public leaderboard, which is even better than the fined tuned LASSO. We would like to continue tuning the XGBoost model.

For the starter XGBoost model, the parameters were by experience. `eta = 0.01`, `gamma = 1`, `min_child_weight = 1`, `colsample_bytree=0.5`, `subsample = 0.8`, `max_depth = 12`, `alpha = 1`.

Before we begin to tune XGBoost, we did feature engineering based on the information got from the XGBoost starter. Based on the feature importance returned by XGBoost starter, we selected the top important categorical features and created their interactions, and combined these new-generated feature with the original data. We selected 35 categorical variables, and the two-way interaction has $C(35, 2) = 595$ new combinations. It took some time to executpe on my laoptop.

```
cat80,cat87,cat57,cat12,cat79,cat10,cat7,cat89,cat2,cat72,
cat81,cat11,cat1,cat13,cat9,cat3,cat16,cat90,cat23,cat36,
cat73,cat103,cat40,cat28,cat111,cat6,cat76,cat50,cat5,
cat4,cat14,cat38,cat24,cat82,cat25
```

Creating feature interactions took a long time, so I combined these created feature with original training and testing set, and stored them as another two files. The final XGBoost model would read the processed dataset instead of original ones to save time. For the script for feature interaction please check `create_new_features.ipynb`.

Usually there are several parameters need be tuned:

```
max_depth
min_child_weight
gamma
subsample
colsample_bytree
```

The Analytics Vidhya Blog provided a methodology for XGBoost model parameter tuning, which sequentially tune the parameters one by one. For each time it tunes one parameters and keeps the other one fixed. It's a greedy method to find parameter combinations that performs good. Compared to grid search, it saves much time. A disadvantage of this method is that it cannot guarantee the tuning is always moving towards the global optima, especially when the scores of different candidates of one parameter are quite close. At this time, it's hard to decide which value to choose and move to tune next parameter. I tried this method and tuned the parameters mamually, but the predicted score is even worse than the default parameters, so I didn't go deeper in this method.

We keep tuning the parameters using another method: Bayesian Optimization. Bayesian Optimization is used for optimizing a function which you cannot write the derivative of the hyper-parameters. The object function is regarded as a black-box. We think the gaussian process is a distribution of functions. Intuitively speaking, Bayesian Optimization choose the next parameter combination to tune based on the performance information of previous parameter combination. It saves much time and number of iterations than grid search.

For the Bayesian Optimization parameter tuning process, check the file `xgb_bayes.ipynb`.

I set the range of several parameters need tune: `min_child_weight: (1, 10)`, `max_depth: (10, 15)`, `gamma: (1, 10)`. The rest parameters keep the same as the original model. Note in Bayesian Optimization you can set `init_points` (randomly chosen points to sample before Bayesian Optimization fitting the Gaussian Process) and `n_iter` (total number of times the Bayesian Optimization is to repeated). These two parameters determines how many times the parameters would change.

The Bayesian Optimization tuned parameters are `min_child_weight: 9.6712`, `max_depth: 13`, `gamma': 2.3452`. They are tuned on training data set by 3-folds cross validation. Re-run the XGBoost with the tuned parameters, the score on Kaggle public leaderboard is 1117.48.

Besides the parameter tuning, there are still other methods would probably increase the score. When we are doing cross validation, for the model trained on each fold, we use this model to predict the test set. For k-folds cross validation, there are k models with the same parameters but trained on different k-1 folds of the data. Take the average of those k models' prediction on test set as the final prediction. This method takes advantage of k fold models we created during cross validation. We not only use cross validation to check measure the performance of the model, but also utilize the diverse model to decrease the variance hence increase final score.

Similarily, another method is bagging which bags the training set and re-run the whole model training process for each bag. The final prediction is the average of prediction of each bag. Due to the limited computation power of the laptop, I just did 5-folds cross validation with no bagging, the average of the 5 models' prediction on test set got a score of 1113.69 on Kaggle leaderboard. For the XGBoost final model, please check `xgboost_final_model.ipynb`.

Neural Network on Keras

I constructed a 4-layer fully-connect feedforward neural network in Keras. The network structure is very simple. Please ignore the number of Param as it may be different if you did different feature engineering or select different number of features to feed the model.

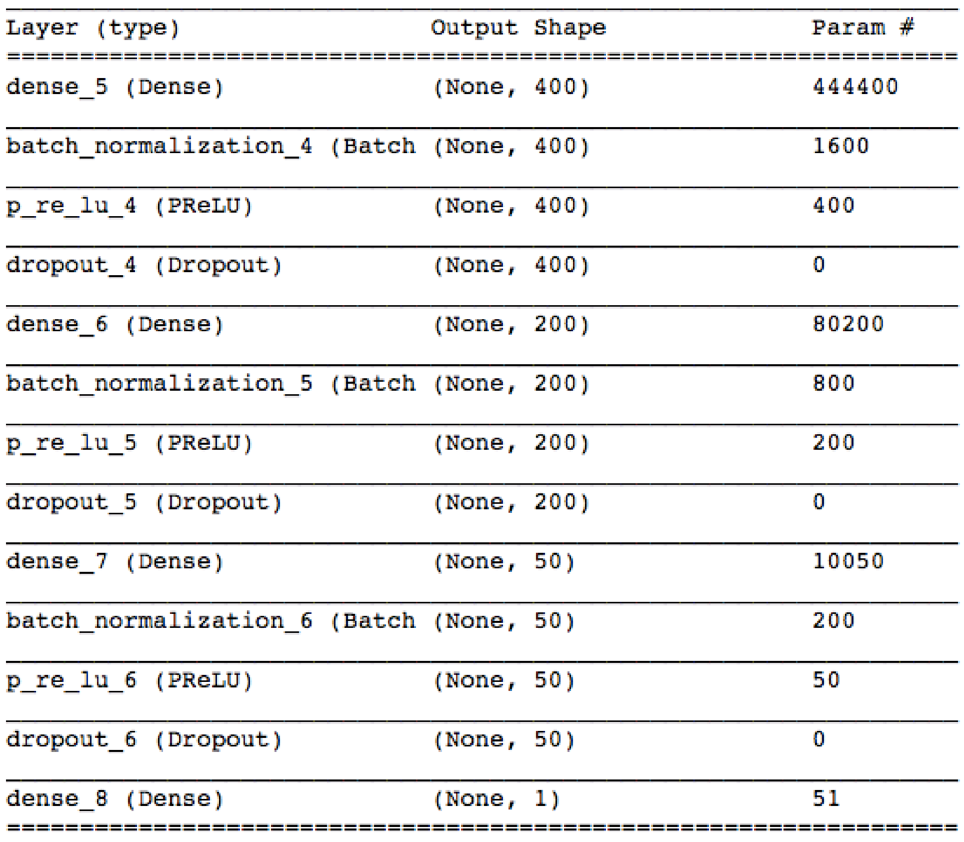
I added drop out to decrese overfitting, and batch normalization to reduce internal covariate shift in neural network. We have three hidden layers, each of them has 400, 200 and 50 nodes repectively. For each hidden layer, I added dropout to decrease overfitting, 0.4, 0.2 and 0.2 for each hidden layer. Note dropout is also hyperparameter need be tuned, here we just set the empirical numbers. I used activation function `PRelu` (Parametric Relu), which doesn't saturate during the learning compared to `sigmoid` and even `Relu`.

As the data is pretty large, we fit data into Keras by `fit_generator` and generate batch whose `batch_size=128`. We set `nb_epoch=100`, `sample_per_epoch` to be the row size of training set. That is, for each epoch we learn 128 (`batch_size`) data entries till all training set has been finished. Then we begin next epoch, and repeat it for 100 (`n_epoch`) times. Note the weight updates for each batch.

We used the optimizer `Adadelta` with the default learning rate `lr=1`. There are different optimizers to choose in Keras.

The performance of Neural Network is pretty good for this data set. The 5-folds and 1-bag model got score of 1117.28 on Kaggle leaderboard. The methodology of parameter tunning is smiliar to that of XGBoost. Since our structure already got a good score, and training Neural Network need pretty long time, I didn't tune parameters of Neural Network in this project. For the final Nerual Network script, please check `keras_script.py`.

The same as XGBoost, due to computation power I didn't implement mutiple bags. But according to experience bagging would prabably increase the score. For the academic purpose we focus on methodology more rather than the final score, so I didn't invest too much time to do



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 400)               444400
_________________________________________________________________
batch_normalization_4 (Batch (None, 400)               1600
_________________________________________________________________
p_re_lu_4 (PReLU)            (None, 400)               400
_________________________________________________________________
dropout_4 (Dropout)          (None, 400)               0
_________________________________________________________________
dense_6 (Dense)              (None, 200)               80200
_________________________________________________________________
batch_normalization_5 (Batch (None, 200)               800
_________________________________________________________________
p_re_lu_5 (PReLU)            (None, 200)               200
_________________________________________________________________
dropout_5 (Dropout)          (None, 200)               0
_________________________________________________________________
dense_7 (Dense)              (None, 50)                10050
_________________________________________________________________
batch_normalization_6 (Batch (None, 50)                200
_________________________________________________________________
p_re_lu_6 (PReLU)            (None, 50)                50
_________________________________________________________________
dropout_6 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 51
=================================================================
```

title

multiple bagging.

Ensemble of XGBoost and Neural Network

Different models may learn different hidden pattern of the data, therefore Ensemble / Stacking is a promsing way to combine information from individual models. If individual models are different and their performance is pretty close, then ensemble/stacking would probably outperfrom each of the individual model.

Stacking usually have two levels. First level is the base (each single) model. We train the models on training set and predict the Out-of-Fold for the training set. E.g. divide training set into 5 folds, each time train model on 4 of the 5 fold, and predict the rest 1 fold. Out-of-Fold predict guarantees that the data doesn't leak to the model. For each different model, compute their Out-of-Fold prediction, and add those prediction of all models you want to stack as new features to the training set.

In second level, we train another classifier (e.g. XGBoost) on the new train (original training set combined with Out-of-Fold prediction as new features). The second level model could learn the useful information from the first level models.

Ensemble is simpler compared to Stacking. It's majority vote or weighted sum of prediction of each individual model. Since stacking need much time to compute Out-of-Fold prediction, I simply did ensemble on the best two kinds of models I trained. There are fives base models, 2 XGBoost and 3 Neural Network. Within these five models, the top three best models were weighted 0.25, the rest two were weighted 0.15. We should note the weights are also hyperparameters can be tuned. Here our weights got higher score than eaqual vote. The weighted sum of several XGBoost and Keras Neural Network prediction got a higher score of 1108.12 on Kaggle leaderboard, which validates ensemble do outperform individual models in some cases. Check `vote_temp.py` for the simple ensemble script.

5 Part 4. Results

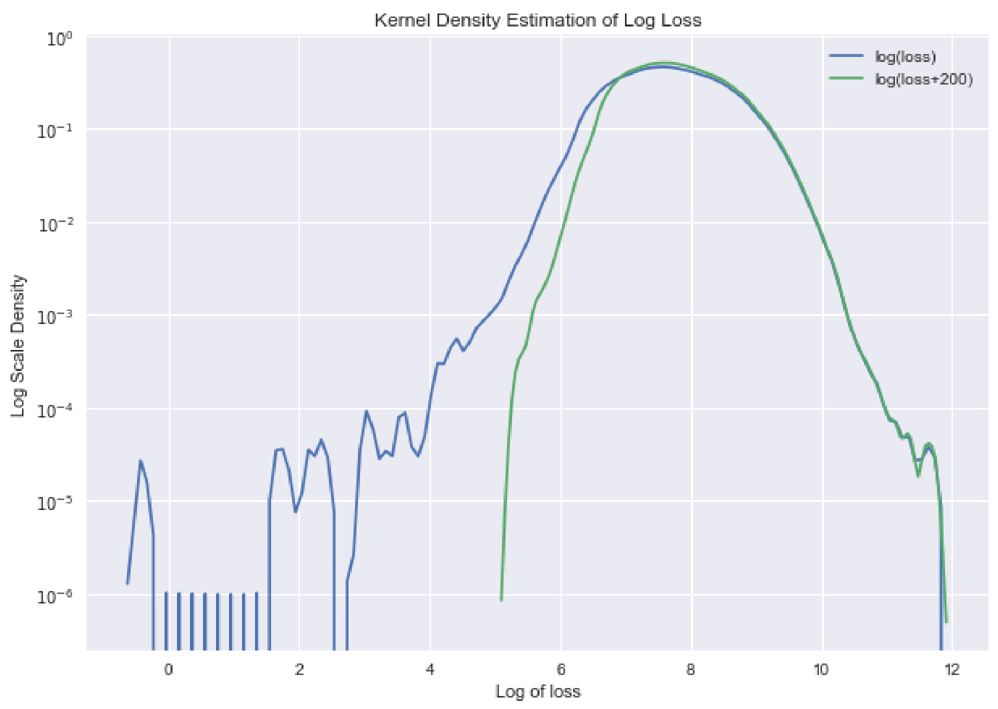
5.1 1. Model Evaluation and Validation

We used MAE as the metric to evaluation the prediction score. When validate the model, I did k-fold cross validation where computed the average score of each fold prediction as the final score on the training set. For the XGBoost model, we tuned the parameter through Bayes Optimization, and also improved the score by taking average of the model of each fold. The final score on Kaggle leaderboard is 1113.69. For Neural Network in Keras, we still validated the model performance by cross validation. The score on leaderboard is 1117.28. By simple ensemble multiple models of these two method, the score on leaderboard is 1108.12.

In this project, we chose MAE as the metric which is resistent to extreme values(outliers) in the data set. For each single model, we did cross validation to tune parameters thus the tuned parameters are more accurate. Also, I computed the average of those k models' prediction on test set as the final prediction. This decreases the variance of the model, and makes it more robust to the data.

5.2 2. Justification

The benchmark of logistic regression has score of 1234.18. We can see as one of the most common used models, LASSO doesn't perform very good is this problem. Compared to benchmark, XGBoost with empirical parameter values got a score of 1124.25, which is far more better than LASSO. By tuning parameters and k-fold average, we can even increase the score significantly.



title

For Neural Network, it also got a decent score with a simple structure. If we spend more time tuning parameters and structure of Neural Network, it could be even better.

We can conclude our two models XGBoost and Neural Network outform the benchmark model LASSO. However, the advantage of LASSO is that the training is pretty fast. Acutally is many cases, LASSO / Rridge Regression performs pretty well. Also in the case where the data set is too large and advanced model like Neural Network need train for a long time, LASSO would be a good choice considering the balance of efficiency and performance.

6 Part 5. Conclusion

6.1 1. Free-Form Visualization

Remember during the data preprocess we add 200 shift on the target before taking logarithmic transformation. Why this shift helps for the predction? Let's visualize it.

Please note the y axis is set as log scale density in order to show the data points whose value and density are very small. We can see after the transformation the plot become more symmetric, and removed the impact of the small-value and small-density points (kind of outliers). According to the result, this transformation do increased the score. We can also regard the shift as a hyperparameter and tune it if there is enough time.

6.2 2. Reflection

This project is quite meaningful as we built a model prototype that a insurance company can use to increase revenue. For me, I learnt a lot through this complete whole pipeline, including data preprocessing, feature engineering, modeling, parameter tuning, ensemble / stacking, even target transforming and loss function customizing.

The most difficult part is the parameter tuning. This is indeed a trivial process which need a long time but cannot guarantee it's moving to the right direction. Probably you just get stuck at a local optima and missed the global one. However, when tuning parameters, both experience and a methodology are important, it would help you to reach better score.

The most exciting part is ensemble and stacking. It showed although a single model may not be the best, ensemble and stacking could combine the good information from individual model and cancel off the bad information. For competition, ensemble and stacking is necessary if you want to get a better score.

6.3 3. Improvement

One possible improvement is the feature engineering. I select the important categorical variables and compute their interactions as new features. This is based on the assumption that the interactions of important features would also be important. Actually we can select more features and compute their two-way or three-way interactions, and use another classifier to select the important one. The library `Xgbfir` can help construct feature interactions and compute importance through XGBoost.

Another improvement is stacking. Build two-layer stacking model, add the out-of-fold prediction as new features. By fine tuning the parameters, stacking would have better performance than ensemble.