Allstate Claims Severity Loss Prediction Report

Ayush Mishra (amm428@pitt.edu) Liping Li (lil112@pitt.edu)

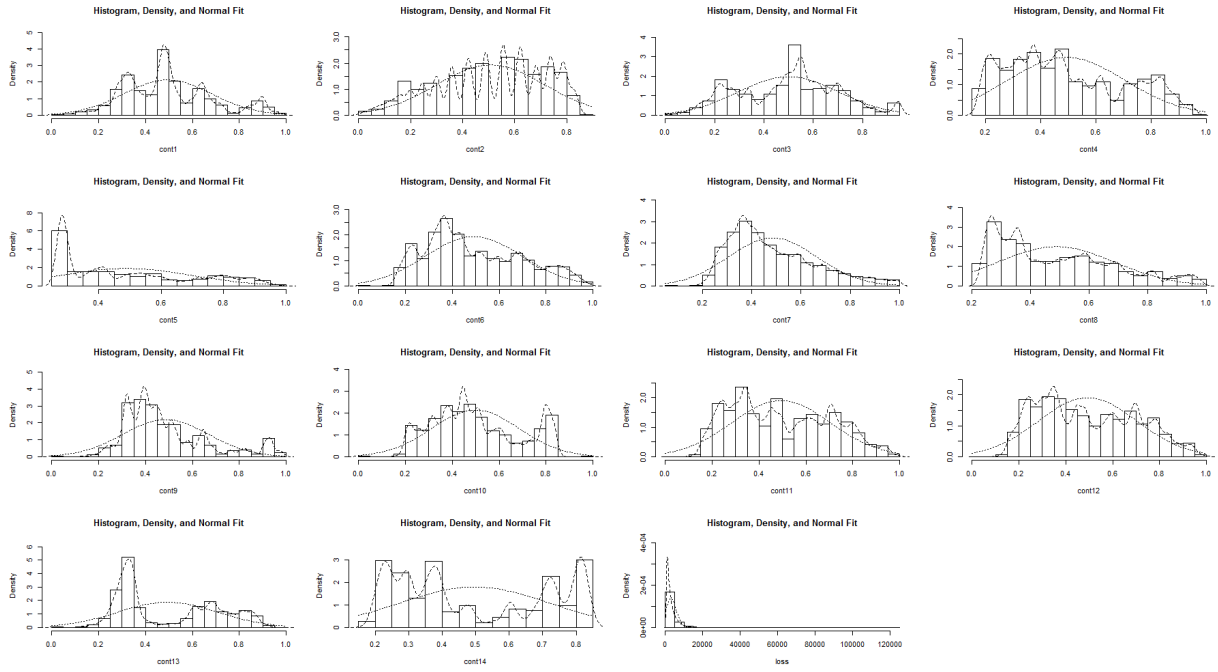
Abstract

This project involves the prediction of a continuous component "loss" incurred by an insurance claim. The features are multiple categorical and continuous variables which perhaps depict the conditions under which individuals are granted insurance claims and determine the value of the insured instance. In this project, we have used the statistical programming language "R" to determine the best methods work with a large dataset which contains a rich combination of categorical and continuous predictors. We describe the Data pre-processing, Model Descriptions, Model performance charts and Ensemble methods as follows.

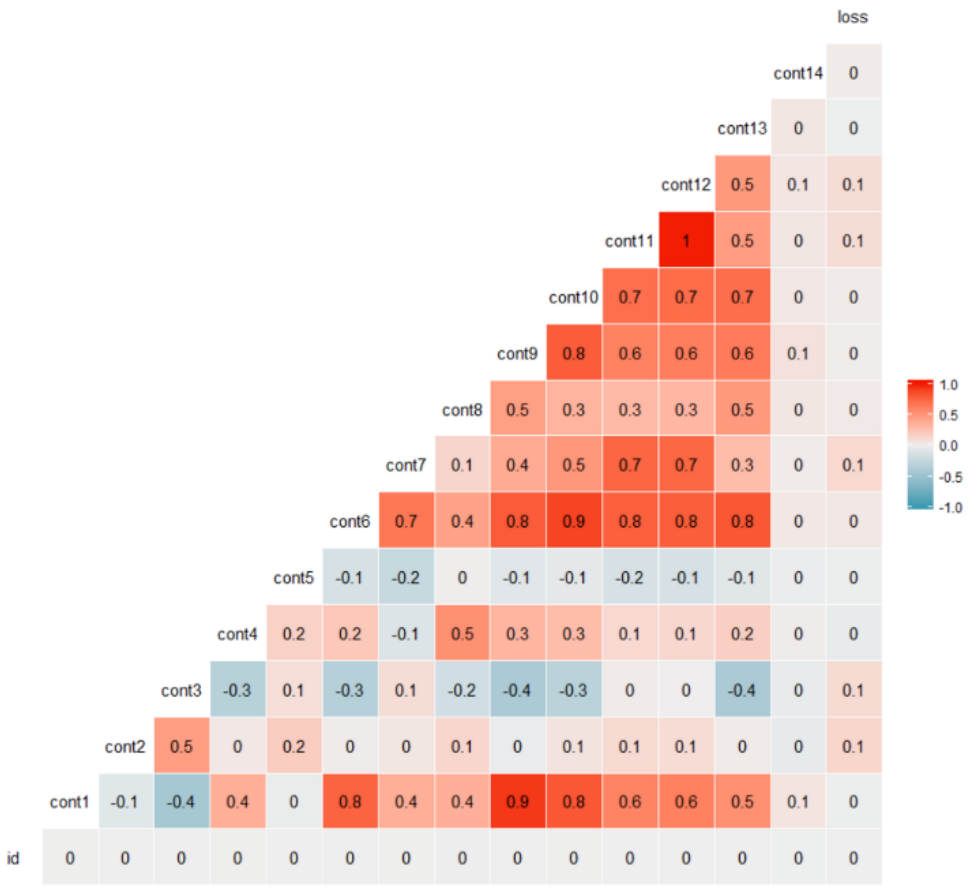
Keyword: Xgboost, neural network, ensemble

1. Data Overview

At first, we import the data and use some visualization method to have an overview of the data.



Histogram Representation of continuous variables



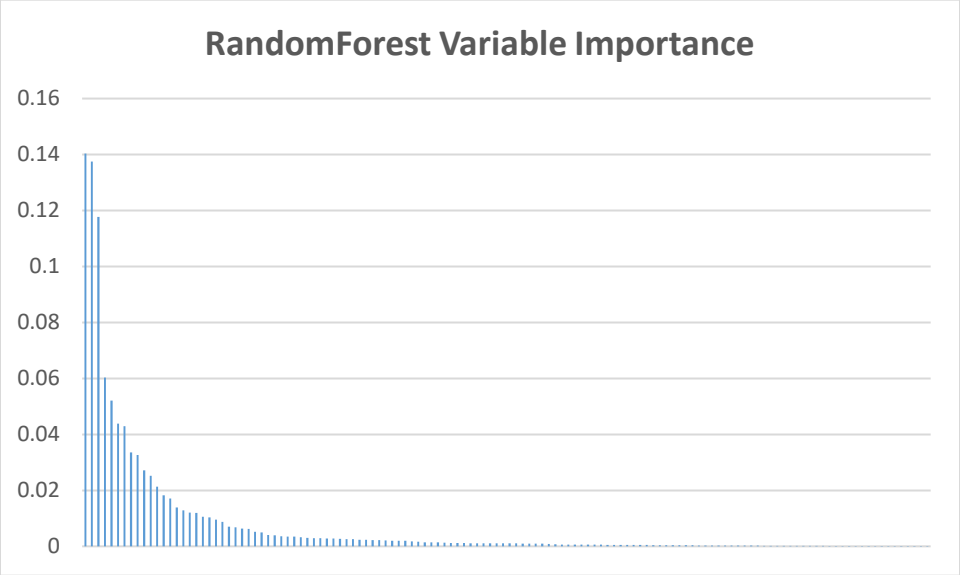
Correlation Heatmap of continuous variables

- There are many high correlations among variables.
- Some continuous variables are not normally distributed.
- Dependent variable “loss” has smooth and continuous outlier on right side.
- 116 categorical variables combined with 14 continuous variables and over 400,000 records together make it impossible for common methods in R to compute.

2. Data Preprocessing

- **Feature Selection**

  Variable Importance: variable importance rank from random forest model – We used the random forest model to recognize and rank implicitly the most important features and then go on to select a subset of the top half of the most important ones.



  Collinear Deduction: eliminate collinear (high correlation index) pairs – We used the regular correlation function for continuous variables and Chi Squared test to determine the probability of the Null Hypothesis of independence of categorical variables. A probability value of lower than 0.05 was considered an indication of independence.

- **Categorical Variable Encoding**

  Sparse matrix

  One-hot – Regular sorting of categorical variables in alphabetical order and converting them into integers

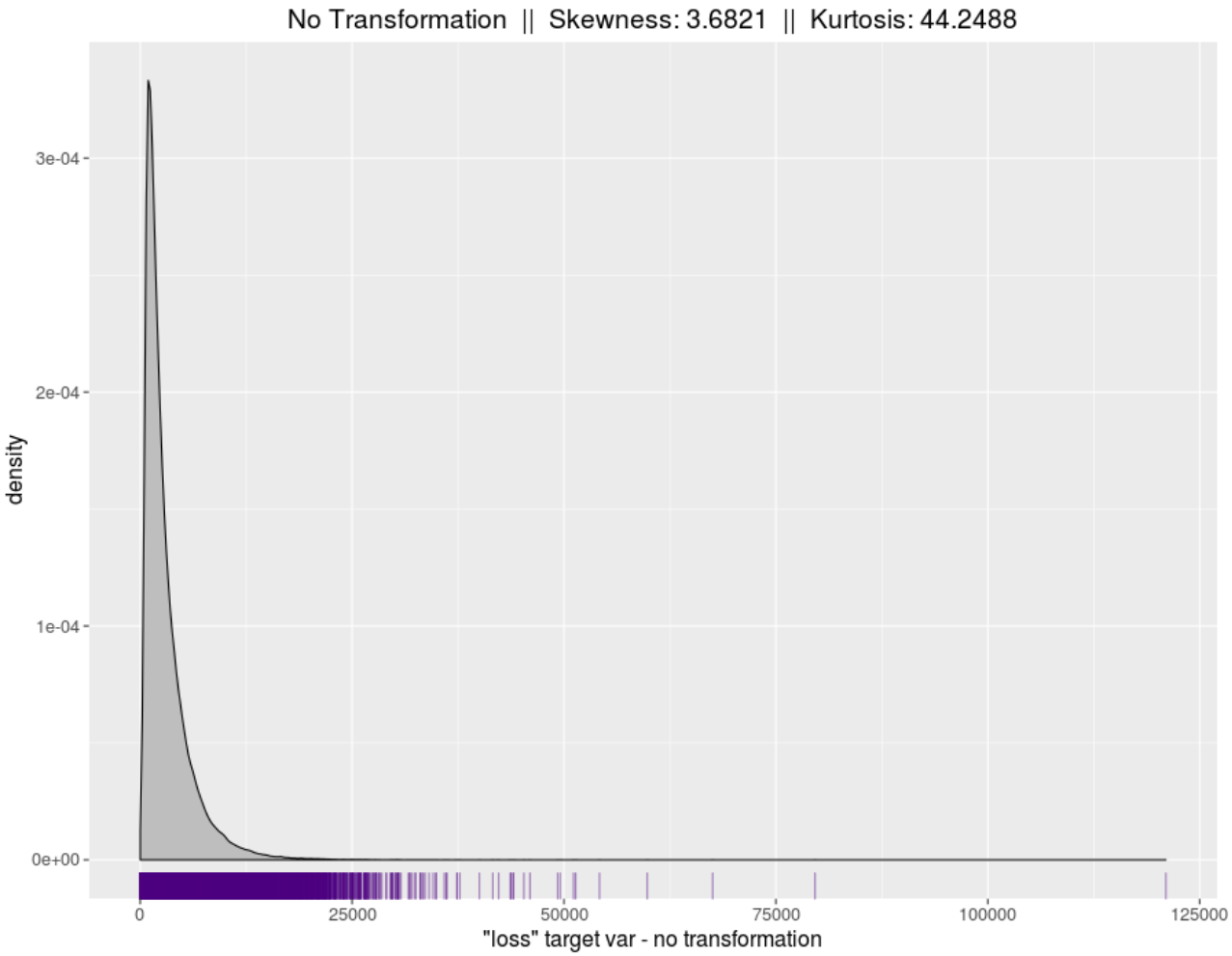
- **Continuous Variable Transform**

  Methods are embedded in different model packages.

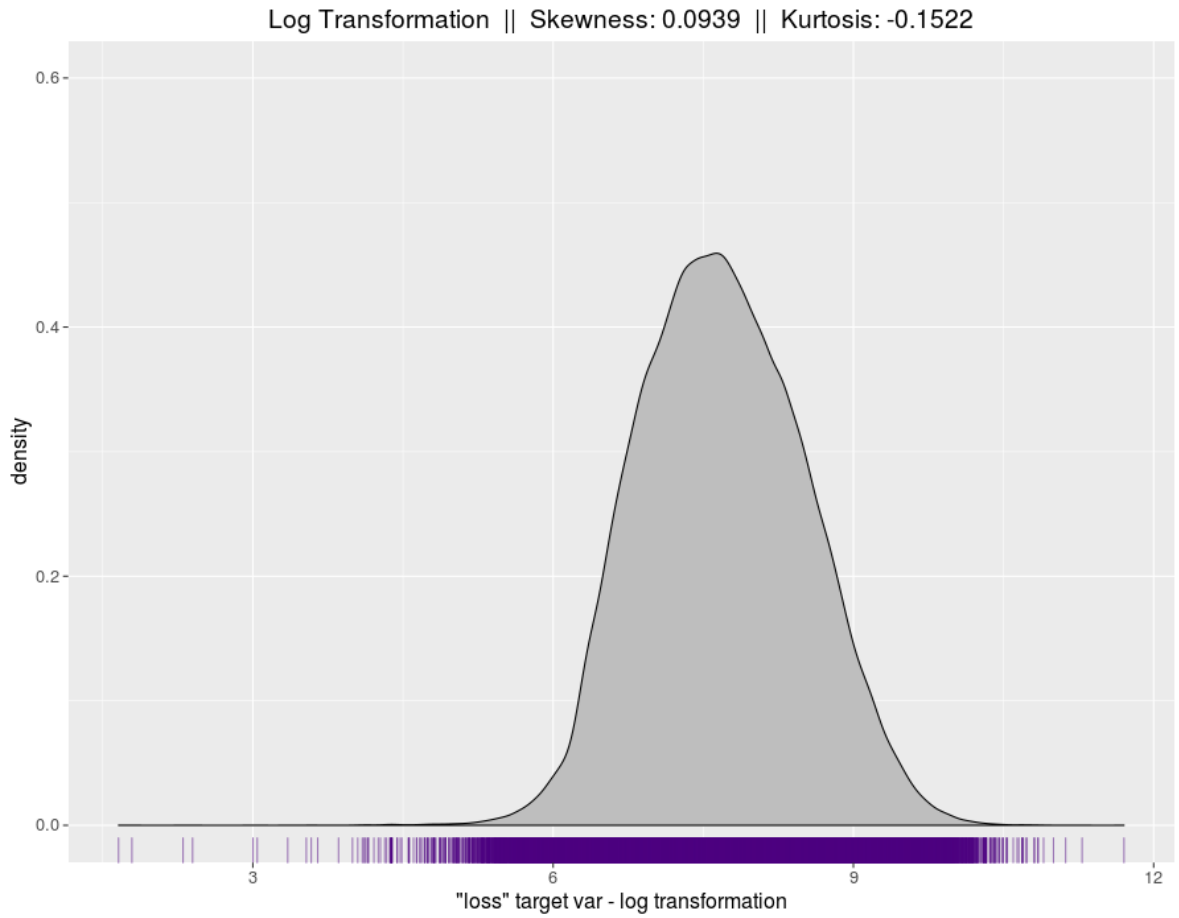
- **Dependent Variable**

  Embedded Distribution: Gaussian, Laplace &Quantile

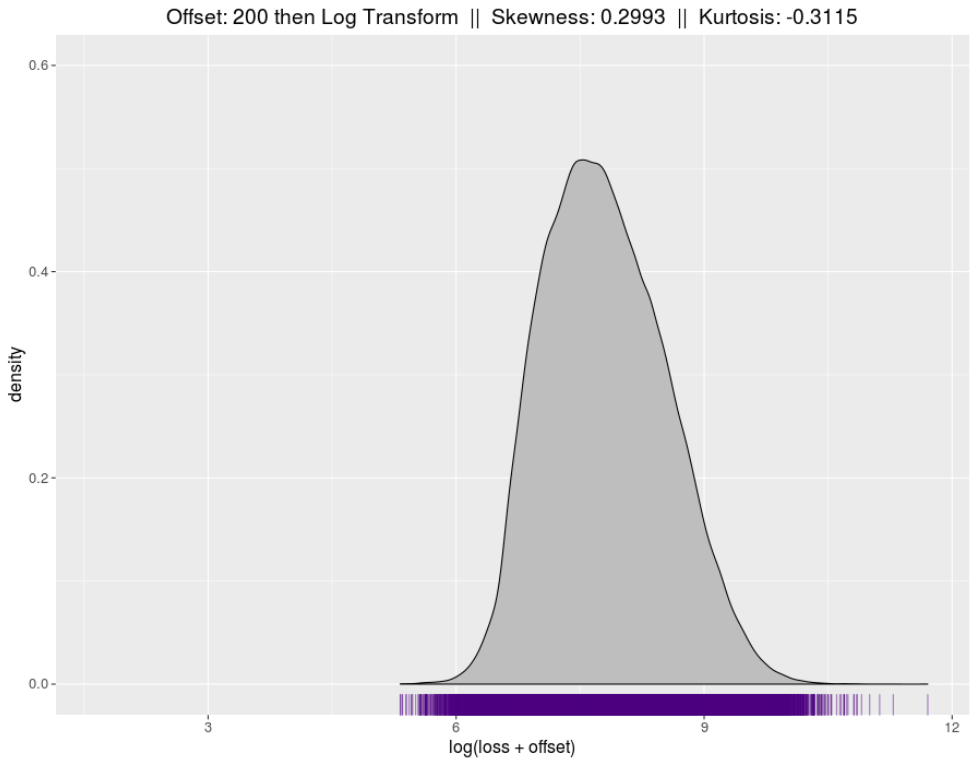
  Shift Introduction (Log Loss transformation):



As we see above, we have a highly skewed response variable with a "long tail "of outliers. This is what leads us towards more unpredictability in our model, notwithstanding that in hindsight, leaving a large scope for improvement. To tap into this plausibility, we Log Transform the "Loss "variable:



The purple bar at the bottom still has some stragglers (indicating long tail outliers). A shift before taking log transform helps cut off the long tail and allow for the data to be more snugly concentrated around the mean, hence aiding in a better prediction:

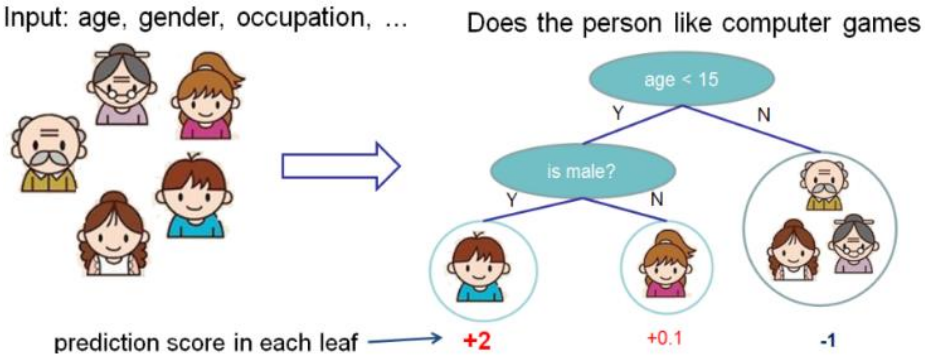


The correct way to choose the offset of course would be to test the model through cross validating different shift values. This could be accomplished using a cross-validation model, computing MAE values for a range of **SHIFT** values. The idea would be to select the trained model with the smallest MAE value and select the corresponding shift offset to proceed with further enhancements.

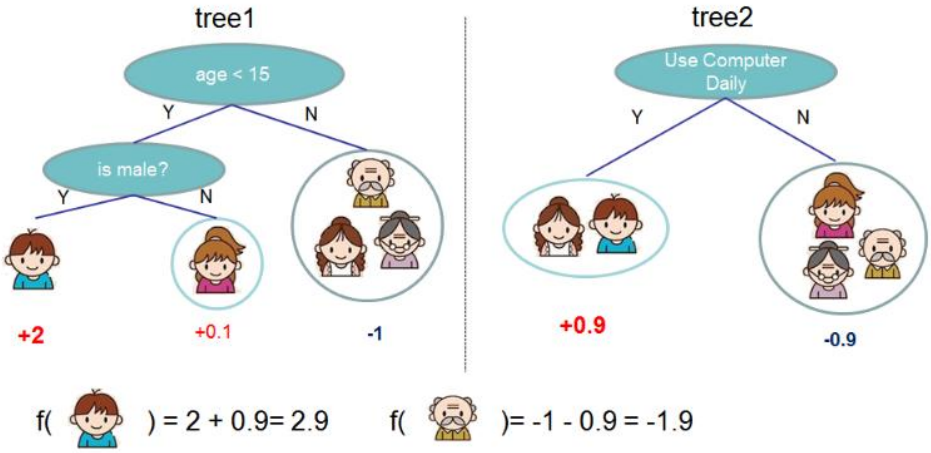
3. MODELS

■ XGBOOST

This stands for Extreme Gradient boosted trees. This model, proposed in the paper Greedy Function Approximation: A Gradient Boosting Machine, by Friedman. This is a tree based ensemble which uses a set of Classification and Regression trees (CART). For ex:
The above classifies members of a family into different leaves and assigns them a score corresponding to each leaf. This is different from Decision trees since D-TREES only have decision values at the leaf nodes. CART, on the other hand has a real score attached to each of the leaves which give us richer interpretations that go beyond classification.



Usually, a single tree is not strong enough to be used for practical purposes, hence, we use the so-called tree ensemble, that has the capability to sum the prediction results of multiple trees together.



Here is an example of the tree ensemble of two trees. The prediction scores of each individual tree are summed up to get the final score. If you look at the example, an important fact is that the two trees try to complement each other. The XGBoost algorithm uses intuitive objective functions very similar to Random Forests (actually they are exactly the same, the only difference comes in the way we train the model). It uses Additive training algorithm to rank the tree structures and then select best tree structures out of the given models (specified as parameters to the given trainer).

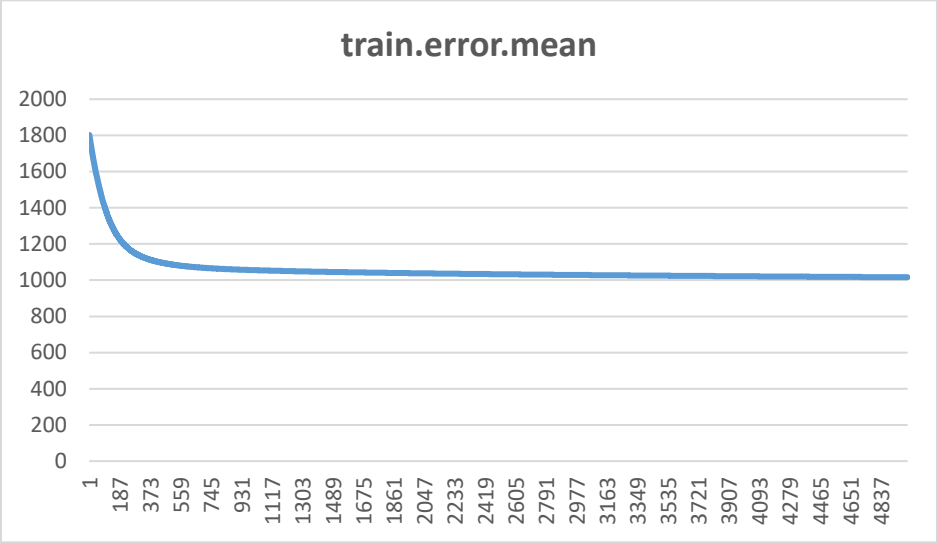
As far as the application goes, we used the "xgboost" package in R. After a lot of testing and consistently getting low MAE scores – 1150-1170, we were able to identify certain parameters that helped enhance our test results:

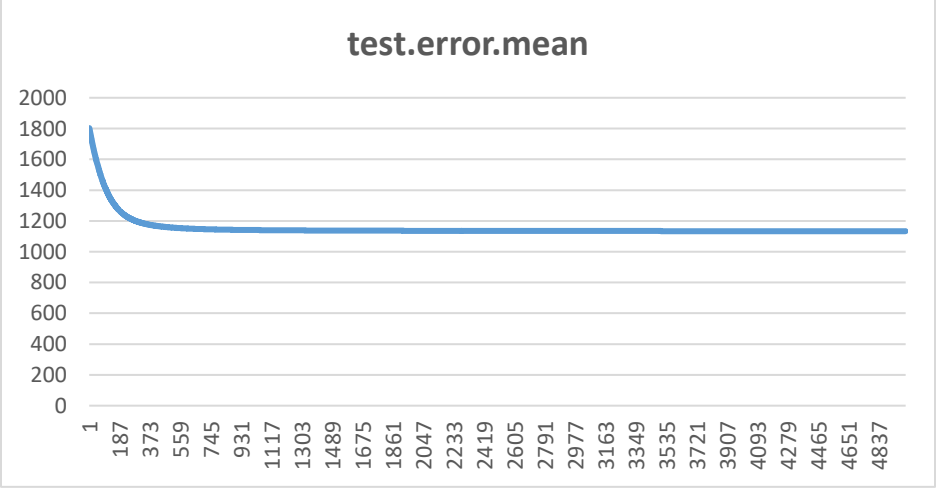
```
xgb_params = list(
  seed = 0,
  colsample_bytree = 0.5,
  subsample = 0.8,
  eta = 0.01,
  objective = 'reg:linear',
  max_depth = 12,
  alpha = 1,
  gamma = 2,
  min_child_weight = 1,
  base_score = 7.76
)
```

colsample_bytree	Subsamples ratio of columns when constructing each tree
subsample	Subsample ratio of the training instance. Setting it to 0.5 means that xgboost randomly collected half of the data instances to grow trees and this prevents overfitting. We use it as = 0.8, since we already subsample the columns to half the size.
eta	Control the learning rate: scale the contribution of each tree by a factor of 0 < eta < 1 when it is added to the current approximation. Used to prevent overfitting by making the boosting process more conservative
objective	Specify the learning task and the corresponding learning objective. "reg:linear" - linear regression
max_depth	maximum depth of a tree
alpha	L1 regularization term on weights. (there is no L1 reg on bias because it is not important)
gamma	minimum loss reduction required to make a further partition on a leaf node of the tree.
min_child_weight	minimum sum of instance weight(hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight, then the building process will give up further partitioning.
base_score	the initial prediction score of all instances, global bias.

These parameters actually helped reduce the train error by 10-15 MAE points on individually trained models. We then proceeded with cross validation of our model. The

advantage of using cross validation is that we get to observe the train error as opposed to test error. We ran our model for 5000 rounds in order to observe the best rounds when it came to average train error and average test error for each round. The following are the plots for corresponding results:

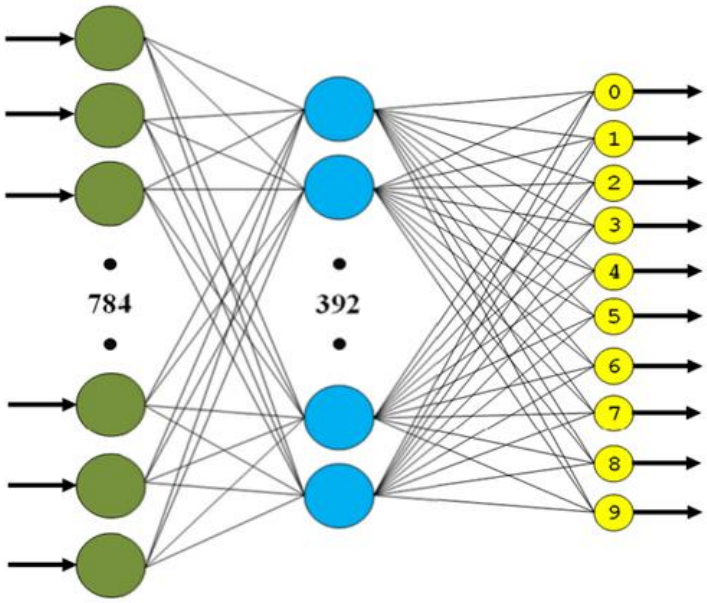




We see a constant decrease in the training and test errors which tend to flatten out at around a 1000 rounds after which the error reduction is minimal. We select the best round of cross validation i.e. the round for which the train MAE came out to be the lowest. We used this to train our model which gave us an absolute MAE of 1133 on the “loss” target variable. This was our best performing model using XGBoost

■ NEURAL NETWORK

It is called “deeplearning” algorithm in h2o package. you could have two sets of neurons: ones that receive an input signal and ones that send an output signal. When the input layer receives an input, it passes on a modified version of the input to the next layer. In a deep network, there are many layers between the input and output (and the layers are not made of neurons but it can help to think of it that way), allowing the algorithm to use multiple processing layers, composed of multiple linear and non-linear transformations.
(source: https://en.wikipedia.org/wiki/Deep_learning#Artificial_neural_networks)



A fast search tool: h2o.grid

One of major shortage for neural network method is difficulty in searching appropriate hidden layers and nodes. h2o package enables hyper parameters for setting parameter lists and compute result for each combination.

Since using only the first half of variables in the importance list didn’t influence train error. I used only half variables to do grid search with h2o deep learning algorithm.

Example of grid search code:

```
search_criteria = list(strategy = "RandomDiscrete",
                       max_models = 100, stopping_metric = "AUTO",
                       stopping_rounds = 5, seed = 101)

nn.grid <- h2o.grid(algorithm = "deeplearning",
                    grid_id = "dl_grid",
                    x = features,
                    y = response,
                   use_all_factor_levels = T,
                    training_frame = train_xf_sp,
                    validation_frame = valid_xf_sp,
                    standardize=T,
                    nesterov_accelerated_gradient=T,
                   #diagnostics=T,
                    hyper_params = hyper_params,
                    search_criteria = search_criteria )
```

```
## Construct hyper-parameter space
hidden.opt= list(c(30,30),c(30,20),c(30,10),c(20,10),
                 c(20,20,10),c(12,6),
                 c(30,30,10),c(40,20))
# distribution.opt=c("laplace","quantile","huber")
#activation.opt=c("Rectifier","Maxout")
activation.opt=c("Maxout","Rectifier")
#distribution.opt=c("laplace","quantile")
#ncg.opt=c(T,F)
epochs.opt=c(8,10,20,30)

hyper_params = list( hidden = hidden.opt,
                     #distribution=distribution.opt,
                     activation=activation.opt,
                     #nesterov_accelerated_gradient=ncg.opt
                     #loss=loss.opt,
                     #stopping_metric=stopmetric.opt
                     epochs=epochs.opt
                     )

```

Example of grid search output:

```
> print(grid)
H2O Grid Details
================

Grid ID: dl_grid
Used hyper parameters:
  -  activation
  -  epochs
  -  hidden
Number of models: 64
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by decreasing mae
  activation epochs      hidden        model_ids               mae
1  Rectifier    8.0 [I@5e93a1ee dl_grid_model_53 1244.5317852621902
2  Rectifier   10.0 [I@50dfbb90 dl_grid_model_35 1239.1688490792126
3  Rectifier   20.0 [I@3c943f99 dl_grid_model_11 1233.8996339298797
4     Maxout   10.0 [I@349d1dfa dl_grid_model_16 1223.4893478987765
5     Maxout    8.0  [I@8fd8580  dl_grid_model_7 1221.2354947038077

---
   activation epochs      hidden        model_ids               mae
59  Rectifier    8.0 [I@4f000608 dl_grid_model_39 1187.8570615818749
60     Maxout    8.0 [I@2d1c69d5  dl_grid_model_1 1185.9674672089327
61  Rectifier   20.0 [I@33d9dc4e dl_grid_model_60 1184.4503405838987
62     Maxout   30.0 [I@37b1c4a3  dl_grid_model_3 1184.3851468372347
63  Rectifier   20.0 [I@68b2a534 dl_grid_model_55 1182.9675318659638
64  Rectifier   20.0 [I@73e4d5a7 dl_grid_model_36 1181.3270944618348
```

These models can be sorted by validation index, like “mae”, “rmse” and so on. But I consider both validation MAE and train MAE.

Example of model selecting:

```
> h2o.getModel(grid@model_ids[[38]])
Model Details:
==============
  H2ORegressionModel: deeplearning
Model ID:  dl_grid_model_13
Status of Neuron Layers: predicting loss, regression, laplace distribution, Absolute
loss, 63,571 weights/biases, 759.9 KB, 2,224,985 training samples, mini-batch size 1
  layer units   type dropout      l1      l2 mean_rate rate_rms
1     1  1048  Input  0.00 %
2     2    30 Maxout  0.00 % 0.000000 0.000000  0.149932 0.278115
3     3    10 Maxout  0.00 % 0.000000 0.000000  0.068734 0.047986
4     4     1 Linear        0.000000 0.000000  0.001114 0.000600
  momentum mean_weight weight_rms mean_bias bias_rms
1
2 0.000000   -0.002682   0.106078  0.313456 0.500682
```

```
3 0.000000   -0.016960   0.194846 -0.107391 0.572861
4 0.000000    0.016542   0.030111 -0.193949 0.000000
  H2ORegressionMetrics: deeplearning
** Reported on training data. **
** Metrics reported on temporary training frame with 9997 samples **
MSE:  4572038
RMSE:  2138.232
MAE:  1120.788
RMSLE:  0.5445744
Mean Residual Deviance :  1120.788
  H2ORegressionMetrics: deeplearning
** Reported on validation data. **
** Metrics reported on full validation frame **
  MSE:  4198245RMSE:  2048.962MAE:  1172.869RMSLE:  0.5570899Mean Residual Deviance :
1172.869
```

Train selected models

I trained 5 deep learning models have a reported train MAE 1114~1120, and their averaged prediction the full train dataset MAE is around 1120. Averaged test dataset prediction get error of 1136 on Kaggle.

Parameter setting:

use_all_factor_levels	Use all the categorical variable levels, default encoding as enum
standardize	Standardize continuous variable. Though I feel continuous variable already be standardized, this procedure still has slight impact
activation option	Rectifier seems to be faster. Maxout seems to work better with less hidden nodes setting.
epochs	How many times the dataset should be iterated.

distribution option	Laplace is double exponential distribution. Money issues often follow exponential distribution. Quantile distribution is to cut groups in terms of quantiles. In general, both Laplace and quantile distribution get better result in deep learning models compared to Gaussian.
Nesterov accelerated gradient	Adjust momentum automatically

Limitation

1. Overfitting

h2o.deeplearning has a high variation between train mae and valid mae. And also, the train mae is computed based on a subsample. So even I tried to decrease the reported mae, the actual mae on full train dataset will probably be 30+ greater. This seems a common problem for deep neural network. Some suggests dropout activation to counter overfitting, but the result with h2o was even worse with dropout activation option.

2. Epochs tracing

Epoch represents how many times the dataset should be iterated, having a best value for each model. But h2o does not provide function to track the train error change curve according to epoch number automatically.

3. Naïve selection of hidden units

It's impossible to search all the possible hidden units in with limited time and resource. Deep learning computation takes 2 mins for models with half variable, epochs less than 10, no cross validation, hidden units less than 30. But there seems no shortcut to find the best setting except try-and-error. Though h2o provide an advanced grid search, finding the optimal hidden units is still tough.

4. Reproducible

The result is reproducible only when CPU thread is set to 1 and use the same sampling seed. In order to decrease computing time, you need to use all threads. Therefore, the result seems unstable.

4. Model Construct & Comparison

Model	Package	data/features	Parameter	MAE
Gradient Boosted Model	h2o	all	default	1214
NeuralNet	h2o	all	use_all_factor_levels = T, standardize=T,distribution = 'laplace', hidden=c(10,5),epochs=5, diagnostics=T	1160
NeuralNet	h2o	first half important variables	use_all_factor_levels = T, standardize=T,distribution = 'laplace', hidden=c(12,6),epochs=5, diagnostics=T	1144
NeuralNet	h2o	first half important variables	use_all_factor_levels = T, standardize=T, activation = "Maxout", distribution = "laplace", hidden=c(12,6), epochs=16, nesterov_accelerated_gradient=T, seed=101,	1123 on its own report, 1147 on full train dataset
NeuralNet	h2o	first half important variables & logged "loss"	use_all_factor_levels = T, standardize=T, hidden=c(14,7),epochs=5, diagnostics=T	1161
Gradient boosted trees	XGboost	All	seed = 0, colsample_bytree = 0.7, subsample = 0.7, eta = 0.075, objective = 'reg:linear', max_depth = 6, num_parallel_tree = 1, min_child_weight = 1, base_score = 7	1149
Gradient boosted	XGboost	Reduced with Chi Squared	seed = 0, colsample_bytree = 0.7,	1183

trees		test. Down to almost half of the features	subsample = 0.7, eta = 0.075, objective = 'reg:linear', max_depth = 6, num_parallel_tree = 1, min_child_weight = 1, base_score = 7	
Gradient boosted trees	XGboost	All (shift = 200)	seed = 0, colsample_bytree = 0.5, subsample = 0.8, eta = 0.01, objective = 'reg:linear', max_depth = 12, alpha = 1, gamma = 2, min_child_weight = 1, base_score = 7.76	1133
Gradient boosted trees	XGboost	Chi Sq reduced dataset. (shift = 200)	seed = 0, colsample_bytree = 0.5, subsample = 0.8, eta = 0.01, objective = 'reg:linear', max_depth = 12, alpha = 1, gamma = 2, min_child_weight = 1, base_score = 7.76	1150
Gradient boosted trees	XGboost	Random Forest importance reduced dataset (shift = 200)	seed = 0, colsample_bytree = 0.5, subsample = 0.8, eta = 0.01, objective = 'reg:linear', max_depth = 12, alpha = 1, gamma = 2, min_child_weight = 1, base_score = 7.76	1135

Some of the better performing models are highlighted as having the blue MAEs.

5. ENSEMBLE Models

The thing about using individual models is that there is only so much we can do about optimizing the performance. There is a limit to parameter optimization and regularization. There is also a huge chance of overfitting the data due to high amount of reinforced learning with some advanced models such as deep nets. These issues directed us towards exploring Ensemble Methods to gain better results.

Bagging method

First method we used was to ensemble the results of various xgboost models. The idea to do this sparks from the bagging method of ensembles. The idea was to use our cross validation results and identify a few best performing rounds, i.e. for those cross validation rounds which gave the lowest mean train error. After identifying these rounds, we trained our models again only specific to these best few rounds:

```
#For submission - test data
predictions <- foreach(m=1:iterations,.combine=cbind) %do% {
  gbdt = xgb.train(xgb_params, dtrain, nrounds = as.integer(best_nrounds[m]/0.8),verbose= 2 )
  exp(predict(gbdt,dtest)) - SHIFT
}
```

The above code snippet, let us reduce each round to 80% of its original size just to avoid overfit and generate some randomness. The "foreach" library helps store the results of each iteration "m" into "predictions[,m]". Once the list is populated over the iterations, the predictions data frame contains, in each column, the results of each iteration (1:m). We can then use the "rowMeans" function to take the average prediction score for each row in the predictions data frame and this results in a simple ensemble of results from the xgboost model.

Result of such ensembles:

Method	Parameters	Train MAE	Test Error
XGB ensemble 1	Rounds:3156,4847,2583	995	1115
XGB ensemble 2	Rounds: 4995,4996,4997,4998,4999	972	1112
NN Ensemble 1		315	1135

NN Averaged 1	hidden=c(30,30,10),epochs=50;
	hidden=c(20,10),epochs=12;
	hidden=c(30,20,10),epochs=34;
	hidden=c(30,10),epochs=10;
	hidden=c(12,6),epochs=14;
		1120	1136

XGB+NN Ensemble of Ensembles	Used XGB ensemble 1 and NN ensemble 1 in a weighted ratio 9:1 in favor of the XGB ensemble	1002	1119

The results were staggeringly good as compared to the performance of single models.

Ensemble of ensembles

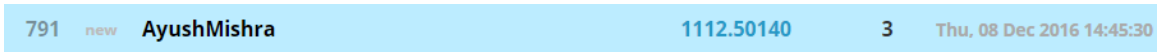
The second method is the ensemble of ensembles. The Neural Network models learned the model so fully that the train MAE went down as low as 315 but there were larger Test Errors ~ 1135. In order to improve the performance of this model, I used the ensembled XGB model to be combined with the NN ensemble. To this end, I applied something called the weighted ratios ensemble method. This is a method where in, we combine the data by assigning a constant weight which giver a higher weightage to the contribution of one of the models in the ensemble as compared to the other one. As a general formula: (A*x+B*y)/(x+y)

This allows the contribution ratio of models A and B in the ensemble to be x:y which is more convenient than having to take a simple average, since it allows for the choice of amount of contribution to the output which is necessary if you have some bad models but could still

```
ensemble_xgb_NN_submission <- (nn_pred_test+xgb_pred_test*9)/10
```

prove essential under certain circumstances:

This resulted in an MAE score of 1117 which was a considerable improvement over the previous models (but still lower than pure XGB ensemble, since overfitting in NN causes loss of information with the test set). Best Ensemble Submission:



6. Conclusion

This project made us realize the predictive potential of even weaker models. Ensembles pave the way for decreasing the complexity of individual models as well as improve performance by way of incorporating larger number of models with relative ease. In the long run, this would overcome any individually exceptional model since every individual model has its limitations as it adapts to the data in question in a certain way which somewhat inhibits its learning from being ubiquitous and it is also important for individual models to remain fairly simple and not to make things overtly complex. Thus, an ensemble provides for incorporating the goodness of multiple models without the baggage of complexity.

It has been a wonderful learning experience working on this project. Further work remains to make use of predictive parameter ensembles which ensemble features in the dataset before we make any predictions, the formulation of which was fairly complex and beyond scope but we intend to keep it as future work.