Predicting Claim Severity for Allstate Insurance

Machine Learning Project

Sanya Goyal, Kian Kamyab, Kimberly Keiter, Sam Koshy (L), Samantha Mazzeo

22th November, 2017

Executive Summary

Allstate Insurance tasked us with creating a model to predict the continuous variable 'loss' which quantifies the severity of a claim to the insurance company. We attempted various machine learning models but found the XGBoost model to be the best fit for the data. After comparing two XGBoost models made using some data preparation and without any data preparation, we found the the best fit to be the model without any data preparation since it had the lowest mean absolute error (MAE). This model performed the best on the validation set with an MAE of 1142.

Data Preparation

The Allstate data set initially consisted of 132 variables: 116 categorical , 14 continuous, an ID, and the target, 'loss'. There were no labels or descriptions associated with the 130 variables possible predictors. As a result of this, we decided to reduce the dimensionality. Although a lot of the models we tested can take all the parameters in the dataset, we wanted to reduce dimensionality to help make simplistic, less computationally expensive models. Data preparation was not used for our champion model (XGBoost2).

Categorical

We split the 116 categorical variables into four categories:

Category	Number of Levels	Number of Variables	Number of Dimensions Before PCA	Number of Dimensions After PCA
Binary	2	72	144	25
Low	3 - 4	16	60	30
Mid	5 - 9	11	76	50
High	10+	17		

We ran principal component analysis (PCA) on binary, low, and mid categories. For each category, we chose the number of components that explained approximately 90% of the variance. For the high category, we treated those variables as continuous variables.

Continuous

For the 14 continuous variables, we created a correlation matrix to see the relationships between them. From this, we removed four continuous variables:

- Cont 9
- Cont 10
- Cont 12
- Cont 13

These four variables were all at least 81% correlated with another continuous variable in the dataset.

After we set up our data, we split the training set into an 70% training set and a 30% validation set. We then ran our models on the training set, found our 'best' models, and validated on the validation set.

Models

XGBoost

We fit two XGBoost models to the data, one without any data preprocessing and one with data preprocessing. We found that the XGBoost model without any data preprocessing resulted in our lowest MAE of 1142 on the validation data set of over 54,000 observations. We chose this as our final model to make predictions.

The XGBoost model's hyper-parameters were tuned to increase model performance. We decided to use an eta value of 0.1, instead of the default 0.3, to make the model robust to overfitting by making the boosting process more conservative. Additionally, we changed the default of the minimum sums of the weights of all observations required in a child from 1 to 1.5, and we increased the max depth of each tree from 6 to 7. Our conservative tuning of this parameter was meant to prevent overfitting, as at higher levels of max depth the model could learn patterns very specific to the training data set. Lastly, we changed lambda to be 1.5, instead of 1, to make the model even more robust to overfitting.

The target variable "loss" was log-transformed to account for its extreme right-skewness. This transformation drastically improved our model performance, reducing MAE from 1380 to 1142.

We manually appended fictitious data (101 for id variable, "B" in all categorical variables, continuous variables can have any numeric data) to the test data so that a sparse data frame could be created for variables with only one level. We later removed this fictitious prediction in the final predicted output.

Random Forest

We fit a random forest, however, we did not choose this as our final model because there is no interpretability associated with it. We initially looked at it for variable importance, however, ultimately did not use it because PCA was able to reduce dimensionality of the variables instead. Random forests are prone to overfitting so we believed PCA was a better route for dimension reduction.

Ridge/Lasso Regression

We did not use ridge or lasso regression for reasons similar to random forests. Ridge regression can help with the problem of overfitting, while lasso regression is good for variable selection. However, we did not find either beneficial for the final model. We chose to adjust the lambda value in the XGBoost model to help with regularization and avoid the issue of overfitting.

Neural Network

We decided to not use a neural network as our final model as it is computationally expensive, and prone to overfitting the data. In addition, we could not fit a neural network that had an MAE competitive enough with the XGBoost model to keep it for consideration.

Appendix

Code.

```
# Importing the dataset
as.data = read.csv('C:/Users/Sam Koshy/Desktop/MSA - Fall/502/Fall 3/Machine
Learning/HW/allstate_train.csv')
as.test=read.csv('C:/Users/Sam Koshy/Desktop/MSA - Fall/502/Fall 3/Machine
Learning/HW/allstate_test.csv')

# Splitting the dataset into the Training set and Validation set
library(caTools)
set.seed(414)
split = sample.split(as.data, SplitRatio = 0.7)
as.train = subset(as.data, split == TRUE)
as.val = subset(as.data, split == FALSE)
library(Matrix)
sparse_train = sparse.model.matrix(loss ~ . -loss, data=as.train)
sparse_val = sparse.model.matrix(loss ~ . -loss, data=as.val)
hist(as.train$loss, breaks=10000)

#Transforming Target
loss1=log(as.train$loss)
hist(loss1, breaks=10000)

# Fitting XGBoost to the Training set
library(xgboost)
library(readr)
library(stringr)
library(caret)
library(car)

Sevierity = xgboost(data = sparse_train, label = loss1, booster= "gbtree",
                    objective="reg:linear", eta = 0.1, gamma = 0, nround=200,
                    max_depth = 7, subsample = 1, colsample_bytree = 1, verbose = 0,
                    min_child_weight=1.5, lambda=1.5, alpha=0,
                    eval_metric = "mae")
ptrain = predict(Sevierity, sparse_train)
error.train=MAE(exp(ptrain),as.train$loss)
pvalid = predict(Sevierity, sparse_val)
error.val=MAE(exp(pvalid),as.val$loss)
error.val

#Refitting with entire dataset
sparse_data = sparse.model.matrix(loss~ . -loss, data=as.data)
```

```
loss2=log(as.data$loss)
Sevierity1 = xgboost(data = sparse_data, label = loss2, booster= "gbtree",
                     objective="reg:linear", eta = 0.1, gamma = 0, nround=200,
                     max_depth = 7, subsample = 1, colsample_bytree = 1, verbose = 0,
                     min_child_weight=1.5, lambda=1.5, alpha=0,
                     eval_metric = "mae")
pdata = predict(Sevierity1, sparse_data)
error.data=MAE(exp(pdata),as.data$loss)

sparse_test = sparse.model.matrix(~ ., data=as.test)
ptest= predict(Sevierity1, sparse_test)
loss= exp(ptest)
write.csv(loss, file = "C:/Users/Sam Koshy/Desktop/MSA - Fall/502/Fall 3/Machine
Learning/HW/orange1.csv")
```