1. DEFINITION

This is a Kaggle competition (https://www.kaggle.com/c/allstate-claims-severity). This is about to estimate insurance claims in the very early stages. Due to liability of insurance, it takes a considerable amount of time to settle a claim. It is difficult for an insurance company to know roughly how much the loss would be. In fact, there are not many claims from past experience, an actuary usually approaches two methods. One method is to find out patterns in the claim data and use them to make predictions. This method can be done by fitting parametric models or probability distribution to these claims. However, there is a major difficulty if there are not enough claim data, in other words, the sampling data is not well-representative to the claim population. Another common method is to examine claim data from other insurers or other industry and identify similar claims to the claims on which the actuary would like to make predictions. This method can not avoid the same difficulty as the previous method and can raise another concern about if the industry data is significantly different from the claim population the actuary is tackling.

If data is enough, in addition to fitting the probability distribution, we can build more complicated statistical models to address the claim estimation. This is also the project goal and is a supervised learning problem. The target feature is quantitative and input space includes both quantitative and qualitative features.

Because I want to make a prediction on continuous data, I am concerned about how predictions can be close to real data or the model performance. There are several statistical metrics which can be used to evaluate the model accuracy, including mean absolute error, mean squared error, regression score functions of r^2 and explained variance. In this project, mean absolute error is chosen because it is the simplest measure and can tell how big of an error can be expected on average. The mean absolute error measures the distance between predicted and true values:

$$MAE = \frac{1}{n}\sum_{1}^{n}|y_i - \hat{y}_i| \qquad (1)$$

Where y_i is the actual value and $\widehat{y_i}$ is the predicted value

2. ANALYSIS

Data Exploration

The data are provided in csv format and includes two sets. One set is used for training models, so it contains the target feature 'loss' and the other set is used for testing models. All features have no names and descriptions, and it means that domain knowledge is not required to solve this problem. The training set has 188,318 samples and the test set has 125,546 samples. The number of categorical features (116) is greater than that of numeric features (16 included ‘id’ and ‘loss’ target feature). Name of the categorical features starts with ‘cat’, i.e. cat1, cat2, cat 3, .etc., and name of the numeric features starts with ‘cont’, i.e. cont1, cont2, cont3, .etc.

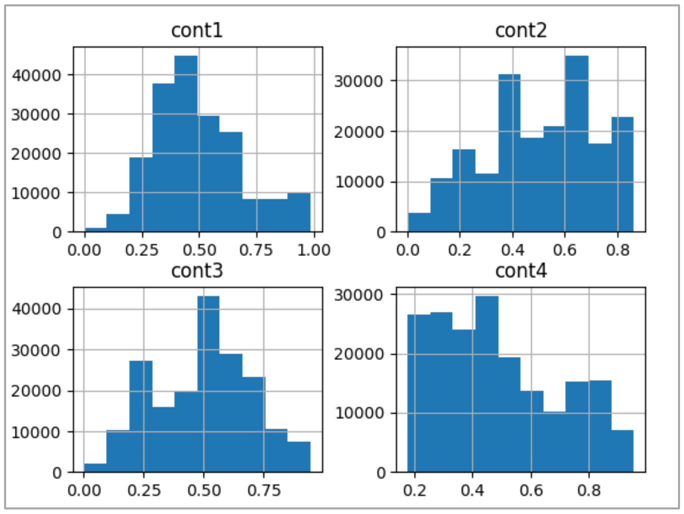

Figure 1: Distribution of cont1 to cont4

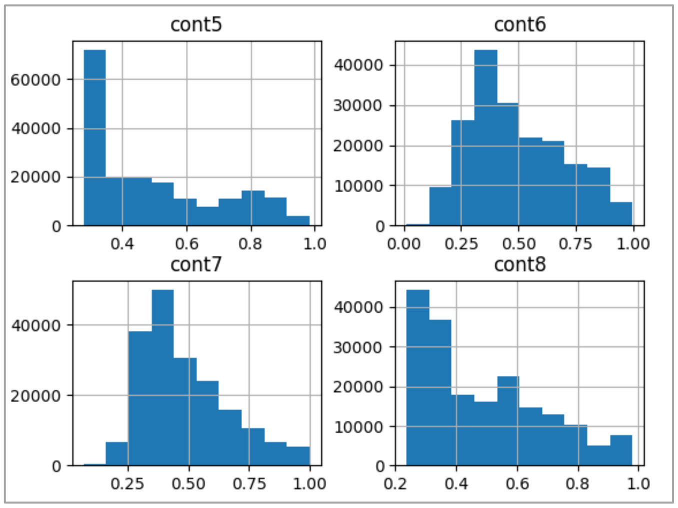

Figure 2: Distribution of cont5 to cont8

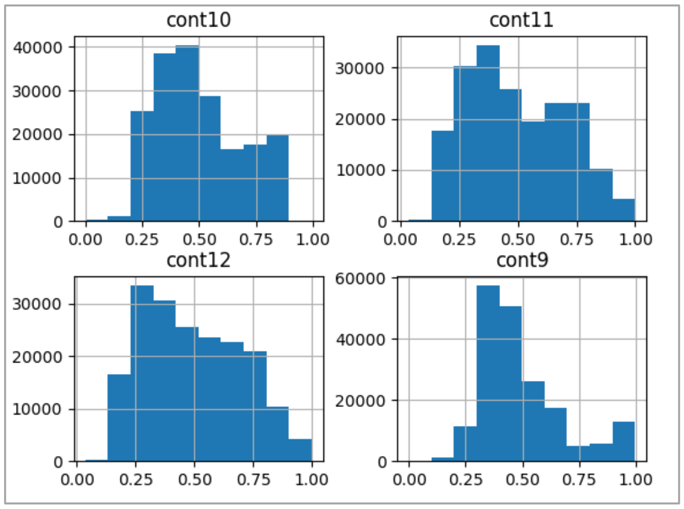

Figure 3: Distribution of cont9 to cont12

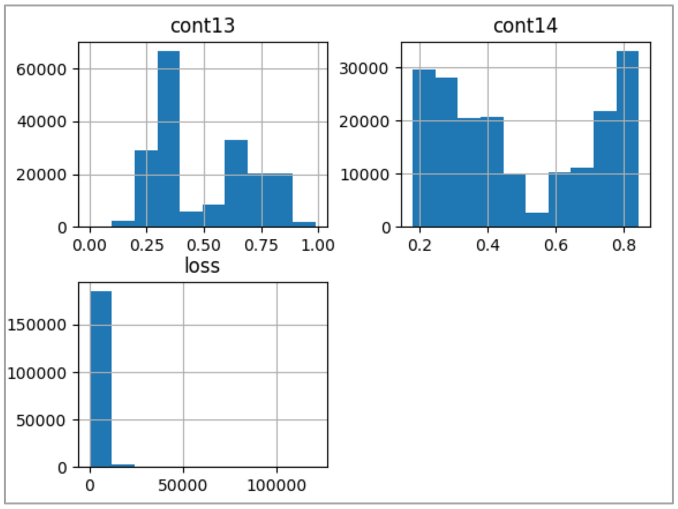

Figure 4: Distribution of cont13, cont14 and loss (target)

Distribution of the numeric features shows unimodal and the values range from 0 to approximately 1 (figures 1 to 4). The features mostly have one mode located between 0.2 and 0.5, except cont2 (~0.6-0.7), cont5 (~0.15-0.25) and cont14 (~0.75-0.85). The mean value is approximately 0.48 to 0.5, but the median value has a wider range of 0.36

to 0.55. Most of them has the median value of 0.44 to 0.46. Spread of distribution of the numeric features is about 0.17 to 0.22. Using skew() function is to calculate skewness of the numeric data and it ranges from -0.31 (for cont12) to 1.072 (for cont9). Skewness value close to 0 indicates the data is normally distributed. In input space, cont2, cont3, cont10 to cont14 have skewness value of less than 0.5. 12 out of 14 numeric features have positive skewness. It means that these features have more weight in the left tail or more values are less than 0.4 or 0.5, except features cont2 (skewness ~ -0.31) and cont3 (skewness ~ -0.01) (figure 1).

Distribution of the target feature 'loss' looks positively skewed with skewness value of ~3.8 and has a wide range from 0.67 to 121012.25. Most values are less than 4000 and the 75% percentile value is 3864.045 and the 50% percentile value is 2115.57. However the higher cost values should not be considered as outliers because these values are possible to happen for insurance claims.

There are 116 categorical features in which cat1 to cat72 have only two labels or levels of A and B (figure 5). Cat73 to cat76 have three levels A, B and C. The other categorical features have more than 3 levels (figure 6). For example, cont116 has 326 different levels and cat110 has 131 levels. The level of A is dominant for cat1 to cat76, cat88 to cat92, cat98, cat102, cat103 and cat114.

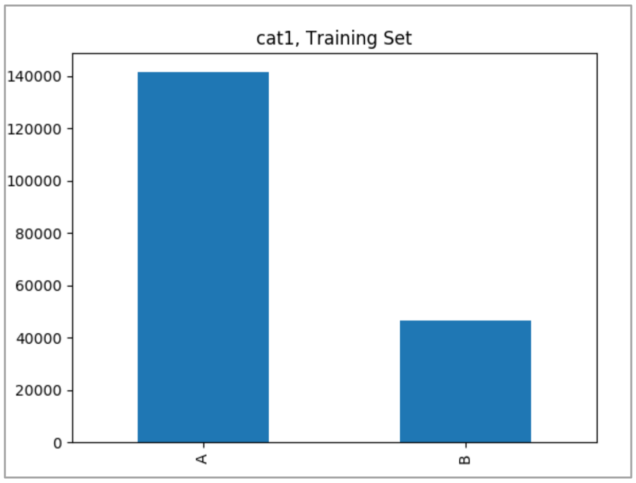

Figure 5: Distribution of cat1 from training set.

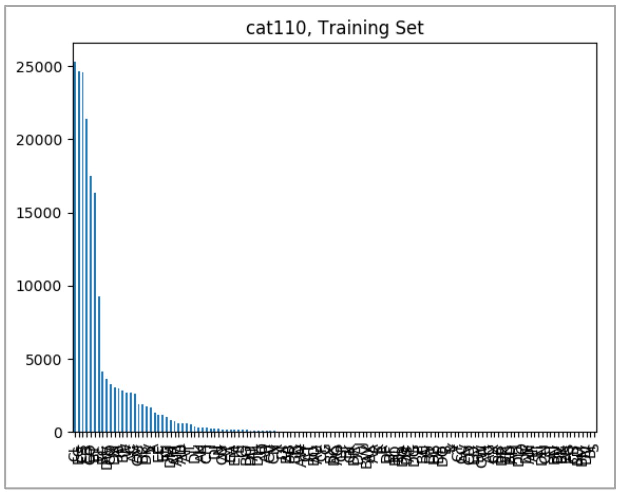

Figure 6: Distribution of cat110 from training set.

Algorithms and Techniques

This is a supervised learning problem and the target is quantitative, so regression algorithms can be applied to solve it. The relationship between input features and the target needs to be examined before experimenting models. Residuals versus the predicted loss is plotted to identify whether there is any discern pattern in the residuals and whether non-linear or linear models are more appropriate. Because the number of observation (188318) is greater than the number of features (404), the problem is not in a high-dimensional setting. However shrinkage approaches can be tried to look at their performance, such as Lasso regression and ridge regression.

There are different techniques to evaluate performance of single models in statistics, for example a single train-test set split, k-fold cross-validation, leave one out cross-validation and repeated random test-train splits. In the project, k-fold cross-validation is selected because it has less variance and more reliable estimate of the performance. It works by splitting the dataset into k folds. The model is trained on k-1 folds and tested on the other fold. This process is repeated k times and produces k different model scores. We can summarize using mean and standard deviation values. The scores are compared between single models and provide score baseline for improving model performance later. The score or metric for this problem is the mean absolute error which is sum of the absolute differences between predictions and actual values. This score gives an idea of the magnitude of the error, but the direction. The mean absolute error has unit of the target feature.

Next, the ensembles are applied to improve accuracy. Three most popular approaches for combining results from different models are bagging, boosting and voting. Bagging and boosting are from the same-type models and voting is from different type models. Bagging is to build multiple models from different subsamples of the training set. Boosting is to build multiple models each of which learns to improve the prediction accuracy of a previous model in the model sequences. Voting is to build the multiple models and take advantages of simple statistics to make predictions. In the project, bagging and boosting models are selected to learn the data.

3. METHODOLOGY

Data Preprocessing

Collinearity can exist between numeric features and implies that the information that a feature provides about the target or response is redundant in the presence of other features. A simple way to detect collinearity is to look at the correlation matrix of the features. Pearson correlations are estimated and correlation value greater than 0.8 is considered as significant. The results show that there are six strong relationships between cont1 and cont9 (r = 0.93), between cont1 and cont10 (r = 0.81), between cont6 and cont10 (r = 0.88), and between cont11 and cont12 (r = 0.99) (figure 7). It is possible for collinearity to happen between more than two features and the correlation matrix does not reveal it. Two simple methods can be used to solve the collinearity problem. The first one is to remove one of the problematic features. The second is to combine the collinear features into a single feature. For this project, the first method is applied and consequently cont9, cont10, cont11 and cont13 are removed from the training and test sets.

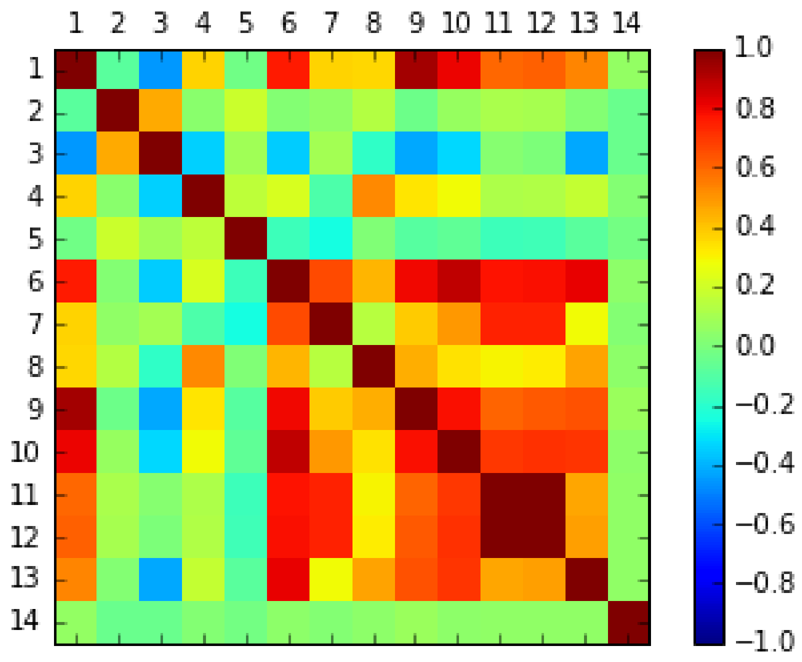

Figure 7: Correlation matrix for the numeric features from the training dataset

Another problem is a skew. Many machine learning algorithms assume a Gaussian distribution for numeric features. Therefore a feature with skewed distribution should be treated. After removing highly correlated features, the skewness value of the numeric features is pretty small <1. The target feature ‘loss’ is highly positive skewed and log function is used to transform the target. To avoid the log of zero value, a random constant, 200 is added to the original target before using log function. The transformed target appears to vary normally in figure 8.

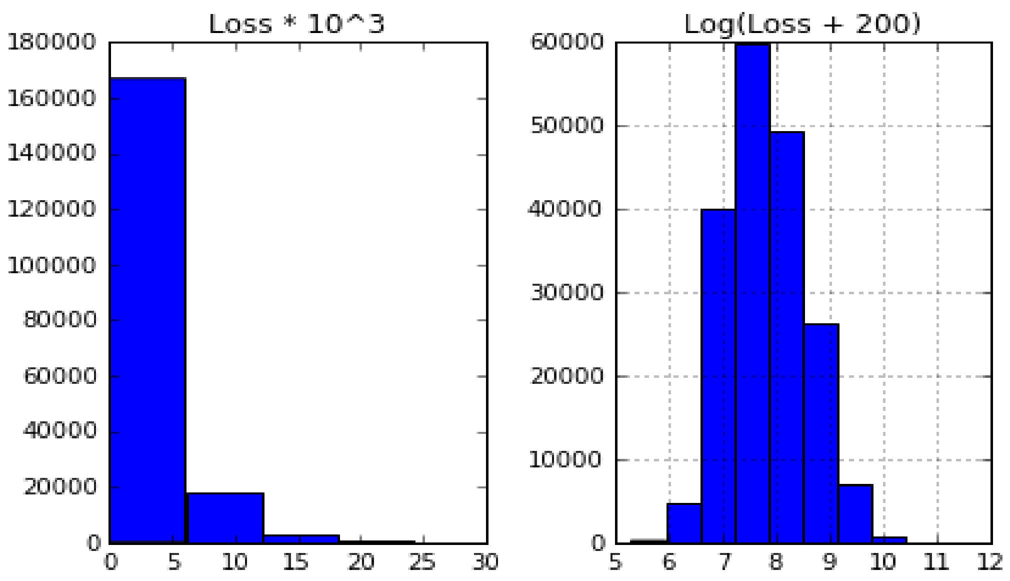

Figure 8: Original (left) and transformed (right) target feature 'loss' in the traing set.

72 out of 116 categorical features have two levels of A and B, most of them face level imbalance. If one level highly dominates the other level for a categorical feature, the information that provides to the target is not important and the feature can be removed. I try different thresholds for occurrence frequency percentage of two-level categorical features, including 95% and 99%. The number of categorical features can drop considerably, and as a result the total number of features in the input space reduces. The 99% threshold can reduce the number of categorical features from 116 to 85. There are some features having many levels some of which rarely occur in the data. Another technique is applied to process them. I combine all levels with very small occurrence frequency from a feature into one level called 'rare'. Two thresholds of occurrence frequency are tried, including 1% and 5%, and the 1% threshold is chosen for the project. Figures 9 and 10 show difference in distribution between original and processed cat105.

There is a problem of level inconsistency for categorical features between the training and test datasets. After processing categorical features, there is only one feature showing this problem. Cat116 has 26 levels in the training set but only 25 levels in the test set (figures 11 and 12). The simple solution is to remove the extra level from the training set. The last step is to convert all 85 categorical features from both the training and the test sets to dummies features using get_dummies() function and the total number of features is 404.

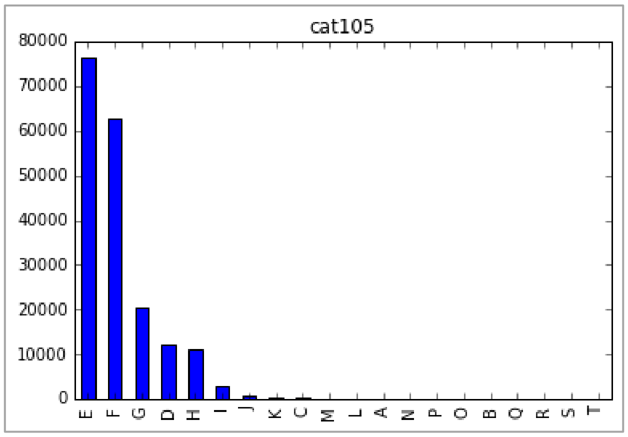

Figure 9: Original ‘cat105’ distribution.

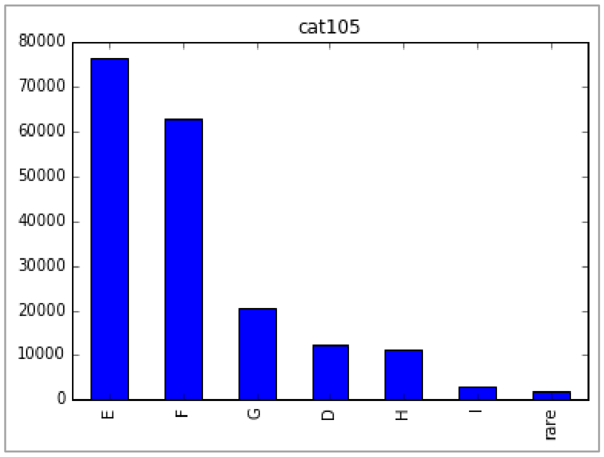

Figure 10: Modified ‘cat105’ distribution.

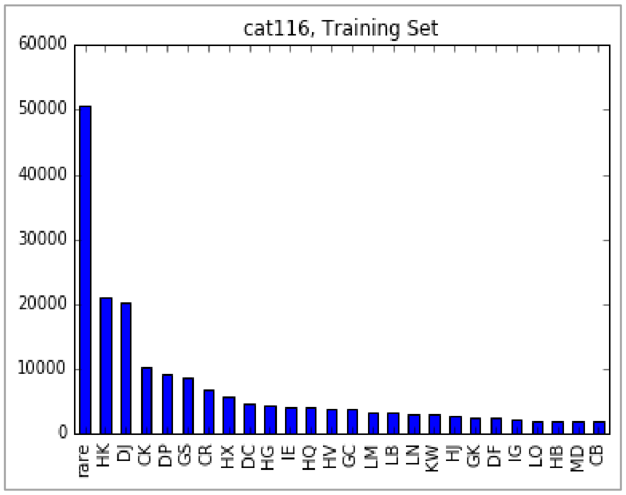

Figure 11: Distribution of ‘cat116’ in the training set.

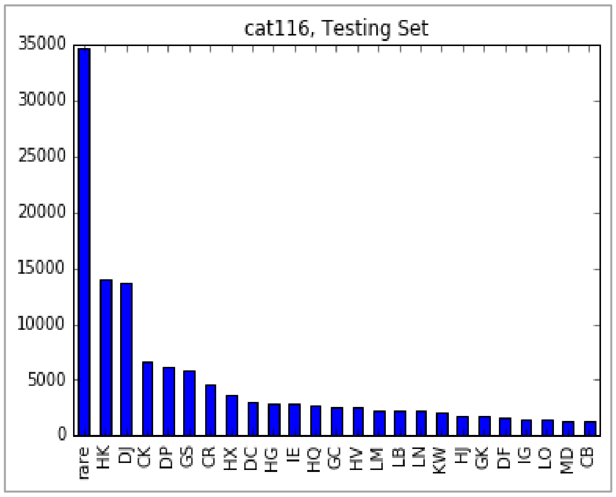

Figure 12: Distribution of ‘cat116’ in the test set.

Implementation

The data are too dense to examine whether discern pattern exists in plot of residuals and predicted values. A simpler version of this plot is drawn from a subset of the training observation (1/20) (not shown here), and it does not really show a pattern in the residuals.

Therefore, both linear and non-linear single models are tried in the project. Linear models are linear regression, Lasso regression and ridge regression. Linear regression is an example of parametric approach that assumes a linear functional form for f(X), where X is feature input. It is easy to fit and estimate a number of coefficients. The coefficients are simply interpreted and significance tests can be performed easily. But it has some disadvantages related to the functional assumption, therefore it performs poorly if the true relationship is far from the assumed function form. Lasso and ridge regressions generally are used for high-dimensional data, and have shrinkage penalties (l_1 for Lasso regression and l_2 for ridge regression) that can considerably reduce variance of coefficient estimates. A tuning parameter $\lambda \geq 0$ serves to control the relative impact of bias and variance on the regression coefficient estimates. The default $\lambda = 1$ is used in this project.

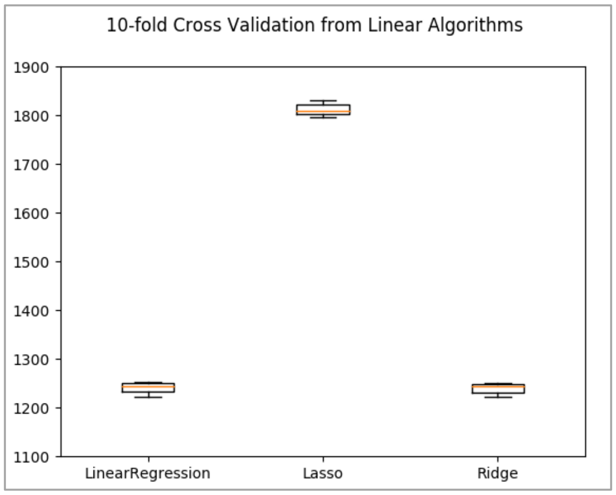

Figure 13: Boxplots of cross-validation errors (10 folds) for linear models.

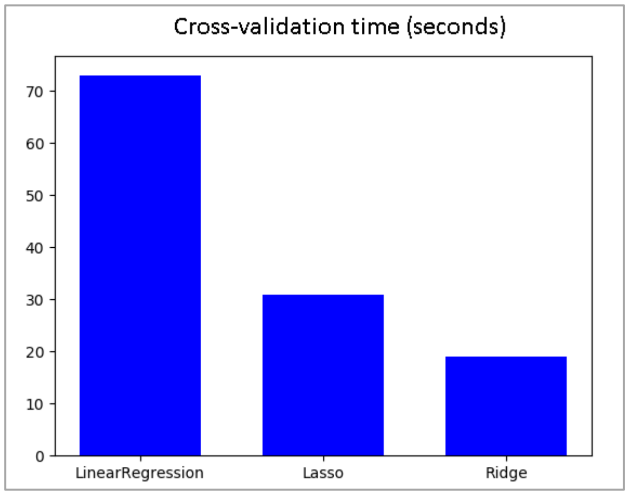

Figure 14: Cross-validation time for linear models.

Cross-validation errors with 10 folds are compared between three linear models. Linear regression and ridge regression have quite similar errors (mean absolute error is 1241.32 and 1240.54 respectively), and Lasso regression performs worse with an error of ~ 1812 (figure 13). These results are not surprising because ridge regression is very similar to linear regression with a shrinkage penalty and includes all features and associated coefficients. Lasso regression is different from the others. Lasso regression can shrink some coefficient estimates to be equal to zero. In other words, Lasso regression can do feature selection and the feature input can lose some information which can be good for fitting model and making predictions. Standard deviation for Lasso regression (~12) is greater than the other models (~9.8). Running time of linear models has order of 10 seconds (figure 14).

In contrast, non-parametric approaches do not make any strong assumptions about the functional form of f(X) and are free to learn from the training data. Three common models are K-nearest neighbors regression, decision tree and support vector regression which are examined with default parameters. These models are to build non-linear and complex relationship between the features and the target. For K-nearest neighbors

(KNN) regression, given a K value and a prediction point x_0, the KNN identifies the K training points that are closest to x_0 and estimates prediction as a mean value of responses of the K closest points. KNN regression is configured with default parameters: K = 5, metric = 'minkowski', p = 2. Regression trees are preferred because of interpretability and visualization, however their accuracy may be not as good as other regression approaches. The regression trees use default parameters, except criterion is set to mean absolute error. Although support vector regression has three versions SVR, NuSVR and LinearSVR, SVR is selected in order to use non-linear kernel with default kernel of 'rbf' and degree of 3.

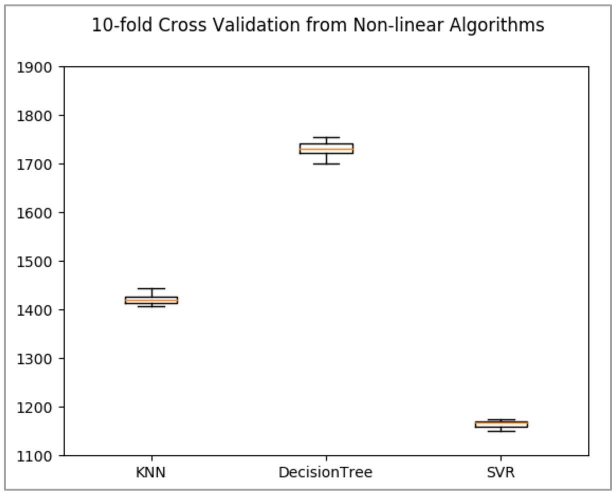

Figure 15: Boxplots of cross-validation errors (10 folds) for non-linear models.

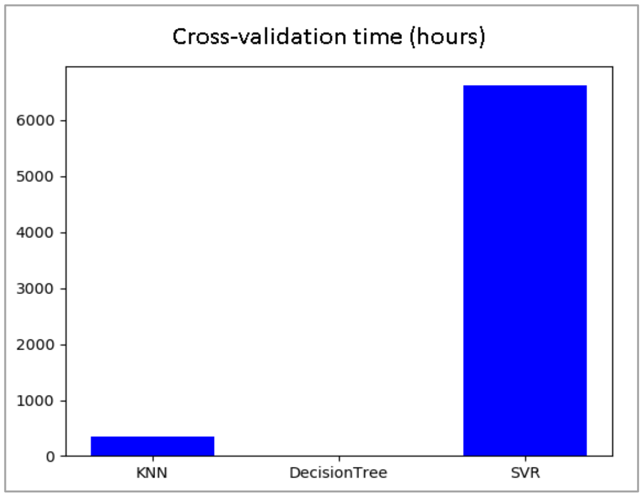

Figure 16: Cross-validation time for non-linear models (in hours).

The results show that the decision tree model has more variance and higher bias (mean error ~1730) than KNN (mean error ~ 1422) and SVR models (mean error ~ 1165) (figures 15 and 16). Support vector regression outperforms KNN and decision trees as well as ridge regression, however it takes longer time to finish training, ~ 4.5 days.

Tree ensemble approaches are powerful and very widely-used. The approaches have several advantages such as learning high-order interaction between the features, invariant to scaling of inputs, and scalability in industry. Two approaches (including random forests and boosted trees) are fitted on the training data with default parameters. These models are compared to one another and the better one is selected for tuning parameters on the training set and making predictions for claim 'loss'.

Random forests provide a modified version of bagged trees. In bagging, a number of trees are built on bootstrapped training subsamples, but in random forests, a random subset of features m are chosen from space of n features as split candidates, each time a split is considered. For this insurance problem, m is chosen to be equal to n.

Boosted trees are grown sequentially, each tree is grown using information from the previous tree and is fitted on a dataset. For each observation i in the training set, the associated prediction y_i is estimated through T times from t = 0 to T (equation 2).

$$\widehat{y_i^0} = 0;\ \widehat{y_i^1} = f_1(x_i) + \widehat{y_i^0}\ ;\ \ \widehat{y_i^2} = f_2(x_i) + \widehat{y_i^1}\ ;\widehat{y_i^t} = f_t(x_i) + \widehat{y_i^{t-1}} \quad (2)$$

One of boosted model is extreme gradient boosted trees with XGBoost library. This model is applied here because it is popularly powerful and performs well in many competitions. It has high flexibility for regularization and optimization of loss function.

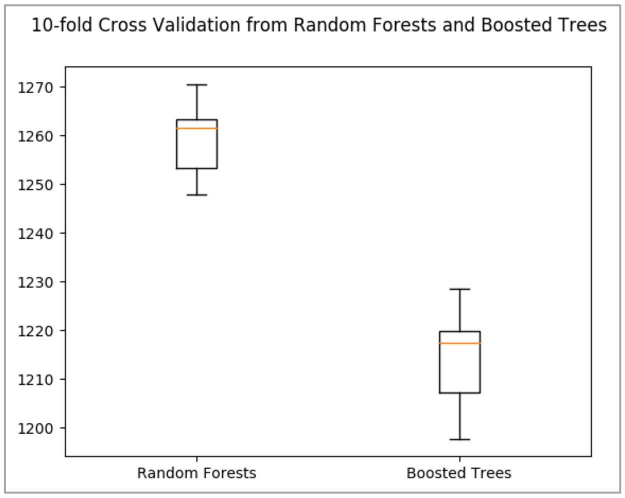

Figure 17: Boxplots of cross-validation errors (10 folds) for random forests and gradient boosted trees.

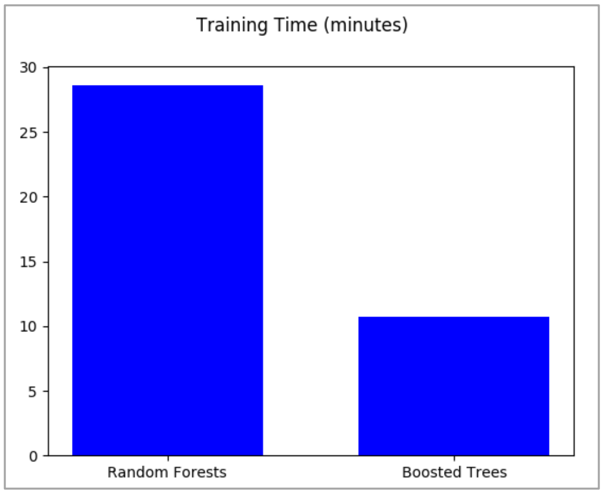

Figure 18: Cross-validation time for non-linear models (in minutes).

The ensemble results show that XGBoost model outperform random forests. The XGBoost model can reduce the averaged cross-validation error by 3.7% (~45.85 loss unit) compared to random forests (figure 17). Standard deviation is about 9.3 for boosted trees and 7.2 for random forests. The running time for 10-fold cross-validation is in minutes for both tree ensembles and random forests take longer time ~ 29 minutes (figure 18).

Refinement

XGBoost model has a defined objective function as below:

$$obj = \sum_{i=1}^{n} l\left(y_i, \widehat{y_i^t}\right) + \sum_{i=1}^{t} \Omega(f_i) \quad (3)$$

Where the first term is training loss function and the second term is regularization. The main task is how to optimize this objective function. Boosting method is additive. It means that adding a new tree at one time to improve the objective function. The objective function for XGBoost model (xgboost library: http://xgboost.readthedocs.io/en/latest/model.html) is required during training the model. The Taylor expansion is applied to the loss function up to the second order and the objective function is in equations 4 and 5.

$$obj^t = \sum_{i=1}^{n}\left(l(y_i, \widehat{y_i^{t-1}}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right) + \sum_{i=1}^{t} \Omega(f_i) \qquad (4)$$

$$= \sum_{i=1}^{n}\left(g_i f_t(x_i) + \tfrac{1}{2} h_i f_t^2(x_i)\right) + \sum_{i=1}^{t} \Omega(f_i) + \text{constant} \qquad (5)$$

Where $g_i = \partial_{\widehat{y_i^{t-1}}} l(y_i, \widehat{y_i^{t-1}})$ and $h_i = \partial^2_{\widehat{y_i^{t-1}}} l(y_i, \widehat{y_i^{t-1}})$

XGBoost trees are highly flexible and allow users to define custom optimization objectives and evaluation criteria. A simplified custom objective function is used here to provide the first (g_i) and second-order (h_i) derivatives in equation 4 as below (from Kaggle forum https://www.kaggle.com/dmi3kno/allstate-claims-severity/farons-xgb-starter-with-custom-objective). The first order derivative is a function of difference of true value and predicted value of the target, and the second order derivative is a function of squared difference of true value and predicted value of the target.

$$g_i = C \times \frac{y_i - \widehat{y_i^{t-1}}}{abs\left(y_i - \widehat{y_i^{t-1}}\right) + C} \qquad (6)$$

$$h_i = \frac{C^2}{\left(abs\left(y_i - \widehat{y_i^{t-1}}\right) + C\right)^2} \qquad (7)$$

Where C is a constant which influences how smooth the loss function is. C is initially set to 0.7 for this project.

Tree-based models are chosen for the booster at each iteration. Five out of twelve booster parameters are focused in order to improve the model performance in this project (Table 1). Before tuning the XGBoost model, XGBoost.cv() function is run in order to provide the best number of boosting iteration for a single run and it is 1542. GridSearchCV() function from sklearn.grid_search is applied to tune on max_depth and min_child_weight, learning_rate and subsample, and then gamma.

Table 1: Parameters and values for tuning the XGBoost model.

Parameters	**Higher value**	**Values**
Max_depth: maximum depth of a tree	More flexible	4, 6, 8, 10

Min_child_weight: minimum sum of instance weight of a child for further partition	More conservative	1, 3, 5, 7
Subsample: subset of training set for fitting trees	More flexible	0.7, 0.8, 0.9, 1
Learning rate: for updating new features at each boosting step	More conservative	0.01, 0.05, 0.1
Gamma: minimum loss reduction required for further partition on a leaf node of the tree	More conservative	0, 0.1, 0.2, 0.3, 0.4

4. RESULTS

Model Evaluation and Validation

The grid search run with 5-fold cross-validation shows that maximum depth of tree highly impacts on the xgboost model performance (figure 19). The mean absolute error for the training data considerably reduces by ~4.6% (~56 loss unit) when changing max_depth from 10 to 4. Otherwise the minimum sum of child weights does not change the error much, ~ 0.1 to 0.2%. The grid scores indicate the first parameter combination performs the best (max_depth = 4 and min_child_weight = 1). The associated error is 1147.4 and is much better than the mean error (~ 1214) obtained from the 10-fold cross-validation with default setting for xgboost model (figure 17).

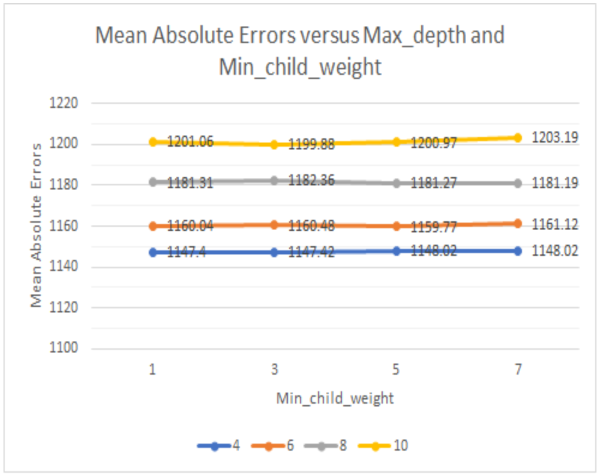

Figure 19: Mean absolute error with changing max_depth and min_child_weight.

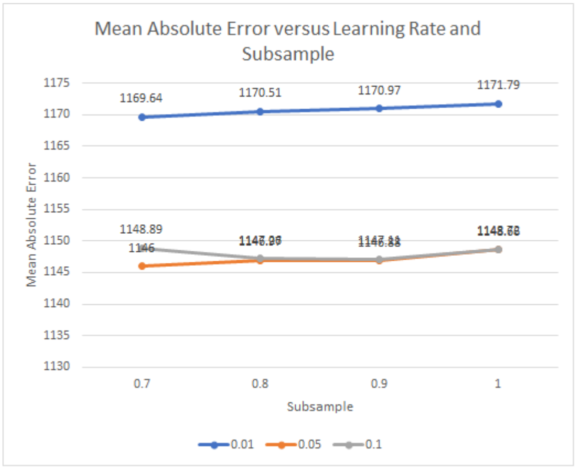

Figure 20: Mean absolute error with changing learning rate and subsample.

Next, the minimum loss reduction required to make a split, 'gamma', is optimized. Five values of gamma: 0, 0.1, 0.2, 0.3 and 0.4 are tried and it seems not to help reduce the training error (error ~ 1147.26 for gamma = 0.4). Therefore, the learning rate and subsample parameters are optimized and see whether the obtained error can be better than 1147.26. Decreasing the learning rate to 0.01 makes the model worsen with errors ~ 1170. The learning rates of 0.05 and 0.1 result in similar results (~1146 to 1148) with different subsample values. The best error is 1146 from the combination of learning rate = 0.05 and subsample = 0.7. A decrease of ~1.26 loss unit or ~0.11% is obtained by tuning the learning rate and subsample (figure 20). The best set of parameters for XGBoost model is : max_depth = 4, min_child_weight = 1, gamma = 0.4, subsample = 0.7 and learning rate = 0.05. This is applied to the test data and the obtained error is 1124.39 on public score board.

These final XGBoost model results are obtained with customized objective function in which constant C is initially set to 0.7. We need to change value of the constant C and examine how the model performs with 10-fold cross validation. In this project, a range of C values from 0.3 to 1.5 is tested. Test errors are smaller than cross-validation errors. The cross-validation error varies from 1145.192 to 1147.46 (figure 21). The cross-validation errors do not have a clear pattern and reach minimum value at C = 0.9. Like the cross-validation errors, the test errors have a minimum value at C = 0.9. The test errors decreases from C = 0.3 to C = 0.9 and then increases at C = 1.1 and C = 1.3. At C = 0.9, the best cross-validation error is 1145.192 and the best test error is 1122.481. By blending the predictions for the test sets with different values of C, the test error can be reduced by 0.6 loss unit (error ~1121.89).

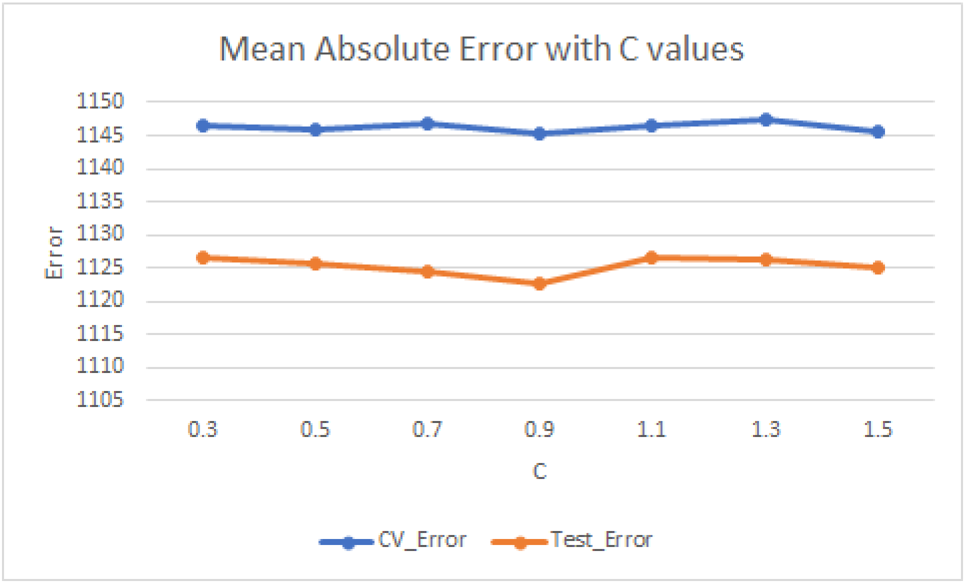

Figure 21: Mean absolute error with changing C in loss function.

Justification

In methodology section, single models are carried out and the best single-model error is chosen as a benchmark for this problem. The best score is obtained from support vector regression ~ 1165. Ensemble models generally can beat the single models by combining the predictions of multiple base learners. Two popular ensemble methods are considered in this projects, including random forests and boosted trees with XGBoost library. XGBoost trees outperform random forests in terms of both model error and running time. Extreme gradient boosted trees run three times faster and have an error 45.89 loss unit smaller than random forests. Due to constraints of time and computer resources, the XGBoost model are trained with several parameters which play an important role in controlling overfitting. Maximum depth of the trees and minimum splitting weight of a child have higher impacts on the overall model performance than other selected parameters. The cross validation errors dramatically reduce from 1213.83 to 1146 with C = 0.7 and to 1145.192 with C = 0.9 after tuning learner parameters. The results from the final XGBoost model are significantly improved and already beat the error benchmark of 1165.

5. CONCLUSION

This project is to model the claim data to predict severity of an insurance claim in term of loss. It is interesting that detailed descriptions are not provided, and data itself have to speak up. Domain knowledge in insurance field is not required, however the dataset itself can show typical characteristics of insurance field. Insurance policy is a contract between the insured and insurer. Insurance policy is designed to meet specific needs and has many legal clauses. This may be a reason why the dataset have more categorical features than numeric features. However this also causes some difficulties in working on the dataset, because multiples labels or levels can exist in each categorical feature. Nearly 2/3 categorical features have only two labels A and B, and A dominates B. If one label strongly dominates the other in a feature, this feature is redundant for modeling the target 'loss'. Another difficulty is related to categorical features with many different labels. Treating these features can cause a big difference in the number of features after transformation. Also, labels in some features are not consistent between the training and test sets that needs to be dealt well. For numeric features, highly correlated features are removed and skewed target is log-transformed before fitting models.

Single and ensemble models are applied to data. Support vector regression and extreme gradient boosted trees work best, however extreme gradient boosted trees are selected because of speed and flexible tuning and obtain the cross-validation error of 1145.192 and test error of 1122.481. Support vector regression is quite powerful algorithm, so if having time, this model should be examined further.

Another improvement can be done with feature engineering. Different thresholds for grouping labels for multiple-label categorical features can be tested or even more no grouping techniques. Features can be selected further by using a single model and examining their importance scores. In addition to evaluating performance of the final XGBoost model with more values of C constant, we also can try with different functions for estimating the first order (gradient) and second (hessian) order derivatives for optimizing the loss function of the XGBoost model, for example log likelihood loss, Cauchy loss function.

Moreover, the voting ensemble approach can be applied to this problem. Different model types for supervised learning can be fitted on data and predictions can be made by averaging with weights among those models. Another powerful model which can be applied to this problem is neural networks. Three popular libraries include Theano, TensorFlow and Keras.