Allstate Insurance Claims Severity: A Machine Learning Approach

Rajeeva Gaur
SUNet ID: rajeevag

Jeff Pickelman
SUNet ID: pattern

Hongyi Wang
SUNet ID: hongyiw

I. INTRODUCTION

The insurance industry has massive scale, and serves an important function by helping individuals and companies manage risk. Efficiencies in insurance claim severity analysis can help keep rates lower for consumers, and provide targeted assistance to better serve them.

This project is inspired by a dataset release by Allstate[6], an US-based insurance company. The goal of the project is to explore a variety of machine learning methods and find the best one to predict the repair cost (or loss) for a given insurance claim.

The dataset has 130 features - 116 are categorical, and 14 are real-valued features. There is a loss associated with each training example. There are 188,318 training examples. We partitioned the given dataset into training, cross validation, and test sets.

II. METHOD

We chose to explore four supervised learning methods: Linear Regression, Random Forests, Gradient Boosted Trees, and Support Vector Regression. Shared across all three methods, we are using the Scikit-learn library[2] for our model creation and tuning.

The categorical data needed to be converted into a format such that the learning methods can make use of it. For this we implemented One Hot Encoding.

Which error metric to use was non-trivial: we could choose either to optimize for the lowest average claim cost, or to optimize to minimize large errors. To reduce the variance in claims costs for an insurance company, we chose to minimize large errors. Thus our error metric is Root Mean Squared Error (RMSE), which was used to evaluate results across all our implementations.

The provided test set does not include the loss target regression value, so we plan to partition our training examples into three sets: training set (80%), cross validation set (10%), and test set (10%). Our goal is to minimize the error on predicted loss for the test set.

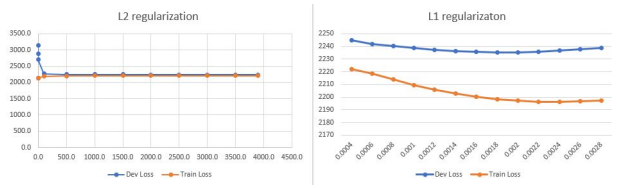

Fig. 1. L_1 and L_2 Regularization

III. LINEAR REGRESSION

A. Overview

Our goal is to predict the cost of repair as close to the actual cost as possible. Linear Regression-based methods provide a real number as an estimate which is used as a predicted cost of repair and also used for computing RMSE to determine model parameters. The cost function for linear regression is the residual sum of squares between actual cost and estimated cost.

B. Models Training and Selection

In this section we determine the best Linear Regression model and parameters. We used the Scikit-learn library[2] for models tuning and selection. In our first three experiments, we used linear regression with no regularization, with L_1 regularization, and with L_2 regularization. We achieved the results as shown in TABLE I, where λ_{opt} was chosen to give the best result on the cross validation set; Fig. 1 shows graphs for regression models with L_1 and L_2 regularization.

TABLE I
LINEAR REGRESSION ERROR

Regularization	λ_{opt}	RMSE (CV)	RMSE (Train)
(none)	(N/A)	3470.8	2143.0
L_2	3100	2242.4	2210.4
L_1	0.018	2235.3	2198.5
0.9*L_1 + 0.1*L_2	0.002	2235.6	2200.0
0.8*L_1 + 0.2*L_2	0.022	2235.8	2200.4

Notice that L_2 regularization has very high value for λ_{opt} which significantly reduces the parameter weights. As a result we focus on L_1 regularization and Elastic

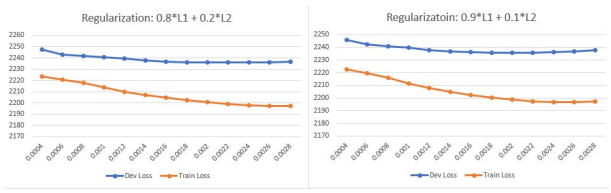

Fig. 2. Elastic Net Regularization

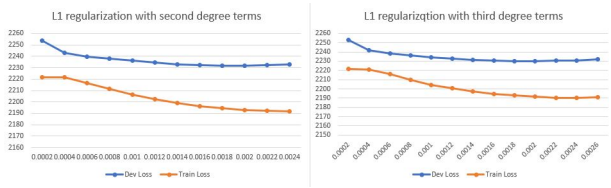

Fig. 3. L_1 Regularization with higher degree terms

Net next as potential models. Graphs in figure 2 correspond to Elastic Nets, and their minimum errors are given in TABLE I

TABLE II

L_1 REGULARIZATION WITH HIGHER DEGREE TERMS

L_1 Regularization	λ_{opt}	RMSE (CV)	RMSE (Train)
L_1	0.018	2235.3	2198.5
2nd degree terms	0.002	2231.7	2192.8
3rd degree terms	0.002	2230.4	2190.5

It is clear from TABLE I that L_1 regularization produces the best result. Next, we add second degree $(x + x^2)$ and third degree $(x + x^2 + x^3)$ terms with L_1 regularization, where x is the value of the continuous features. From figure 3 and TABLE II we conclude that the model with the third degree terms gives the best result. (Due to time limitations and since there are only fourteen continuous features we didn't add terms beyond third degree.) Once we determined that L_1 regularization with third order terms gives the best result, we investigated if increasing the size of the input will improve the error on the cross validation set. In the Fig. 4 we chose 7 input sizes starting from 100000 examples and incrementing it by 10000 examples in each iteration. For each input size, we ran L_1 regularization with third degree terms. We chose the smallest cross validation error and corresponding training error for each set of training input size. The graph in figure 4 shows that beyond 110000 examples, with the increase in the training set size the training error is decreasing; however cross validation error - excluding some small variations possibly due to small

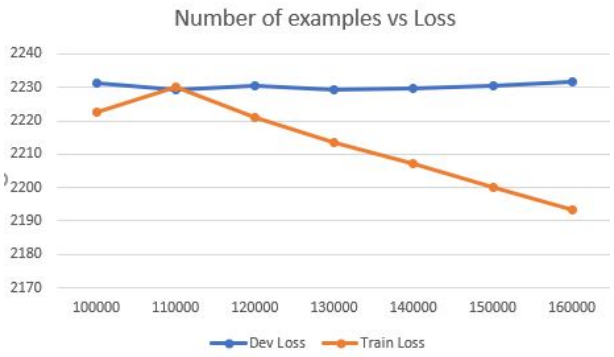

Fig. 4. Errors with training size

changes in parameters - stays nearly the same. We conclude that a training size $\geq$ 110000 examples is a reasonable size for the cross validation errors to be nearly optimal.

C. Results

We considered several models for linear regressions and tuned them to get the best possible RMSE on the cross validation set. The best RMSE on cross validation set was 2230.4 for the linear regression model with third degree terms and with L_1 regularization. Also, we determined that RMSE on cross validation set doesn't change after training examples $\geq$ 110000.

IV. RANDOM FORESTS

A. Overview of Random Forest

We now discuss our analysis using a Random Forest learning algorithm[1] for this regression task. While normal decision trees can quickly overfit and learn the peculiarities of the training data, random forests utilize feature bagging such that each forest considers a randomly chosen subset of the available data. The outputs of each of the forests are then averaged, resulting in a smoothed output that allows for variance to be minimized.

B. Initial Parameter Tuning

We used the Scikit-learn library[2], and it's RandomForestRegressor class. The inherent tree structure of a Random Forest yields many tunable parameters which deal with tree depth, leafs (weighting, samples, number), splits, etc. These proved useful in tuning the model to minimize the RMSE.

It is computationally infeasible to try even a moderate subset of the permutations of parameters. Thus we first looked at six parameters individually to get a general sense for how they affect the loss output.

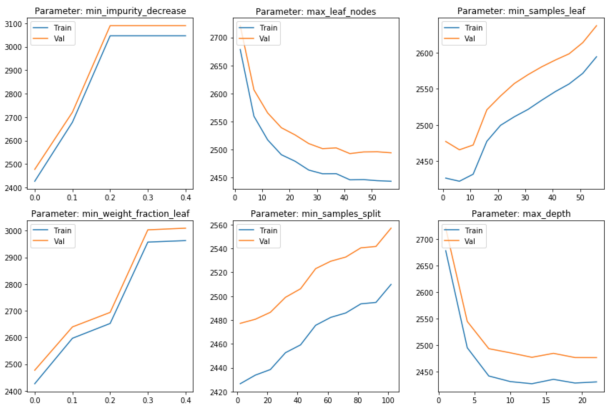

Fig. 5. Random Forest: Parameter Tuning

Figure 5 shows the training and validation loss for each of these six parameters with varying values.

Many trends can be gleaned from these charts. Firstly, both `min_impurity_decrease` and `min_weight_fraction_leaf` show no benefit from being increased, so we fix them both to zero. Similarly, `min_samples_split` should also remain at it's lowest allowable value of two. Both `max_leaf_nodes` and `max_depth` show the opposite trend, indicating they should be set reasonably high to capture the descriptive power in the dataset. Finally, `min_samples_leaf` shows high variance for even small changes in the parameter value. After further experimentation, a low value near the default value of one proved optimal.

C. Grid Search

From the initial tuning, we identified `max_depth` as a parameter with a non-obvious optimal value, and thus warranted further examination. Additionally, large values of `max_depth` led to overfitting of the training set, so we want to be careful to avoid this high variance situation. We chose `max_depth` as one of the two parameters in a grid search.

One final parameter that hasn't been discussed yet, but has a large effect on the outcome of the Random Forest model is `max_features`. We chose this as the second parameter for the grid search. Note that after one-hot encoding, there were over 1000 features, so we tried values which spanned this range.

Each subplot in figure 6 uses a different value for the maximum tree depth, and within each subplot is the loss versus the maximum number of features used in a forest.

It is immediately clear that increasing `max_depth` beyond eight leads to the overfitting we wanted to avoid, as the validation and training errors diverge after that point. Also, using a large number of feature yields better loss values, and has no indication of leading to overfitting.

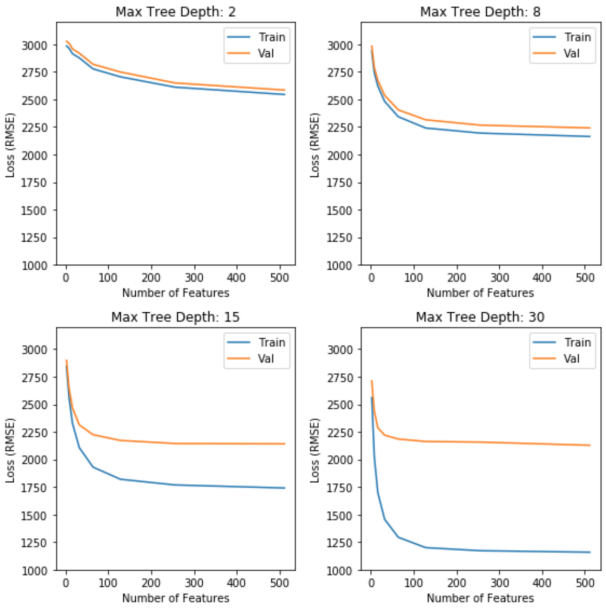

Fig. 6. Random Forest: Grid Search

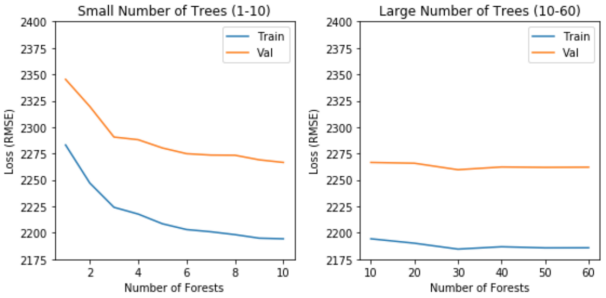

Fig. 7. Random Forest: Loss by Number of Trees

D. Results

The final variable is the number of trees to use. In all our testing, at no point did increasing the number of trees lead to a higher loss. This is likely because we took care to not overfit our model, and thus any random forest was a collection of some number of well-fit trees. Thus the number of trees (`n_estimators`) isn't so much a parameter, as a tradeoff between training time and decreased loss.

Figure 7 shows the loss achieved at various numbers of trees in the forest.

Notably, the loss tapered off beyond 30-50 trees,

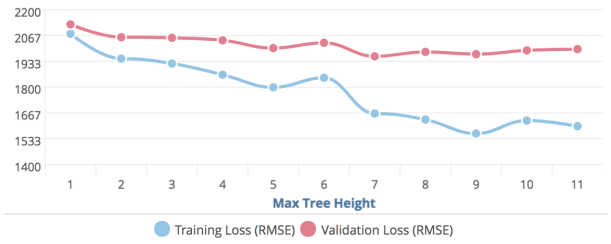

Fig. 8. Gradient Boosted Tree Loss

meaning values for `n_estimators` beyond this would yield small gains at the cost of ever increasing run time. Using 50 trees, we achieved a final RMSE of 2198.25.

V. Gradient Boosted Trees

A. Overview

Another non-linear model we tried was gradient boosted decision trees. Similar to random forest, it is a an ensemble of tree models. Boosted Trees are capable models which are able to fit the training data extremely well, and - given care during training to avoid over fitting - generally perform extremely well.

B. Model Training

Although gradient boosted trees are a relatively simple model, there are various parameters to tune to yield an optimal result. These parameters and tuning methods are discussed in this section.

The parameters that makes the most impact are max tree depth, and min child weight. The max tree depth parameter controls how deep a tree can grow, and represents the complexity of the model. A more complex model with high tree depth can fit the training data extremely well, but such model overfits the training set and performs poorly on dev and test sets.

Figure 8 shows the impact of this parameter, where training error continues to decrease with increasing tree depth, dev error starts to go up past a certain value as the model starts to overfit the training set. Clearly, the optimal tree height is 7. Min child weight also serves to control model complexity by limiting the nodes that can be split further. With these 2 main parameters, we performed grid search to find the optimal value.

There are other parameters to tune for gradient boosted tree, including gamma, subsample, lambda and alpha. After fixing the 2 main parameters, we they performed another grid search on pairs of parameters. This produced the final parameters that we used in our model. We had to do 3 tiers of pair-wise search to reduce training time, because training these parameters together yields a $O(n^6)$ complexity, where n is the resolution of the search. This approximates the optimal value and yields reasonable training time.

C. Results

With optimal parameter values, using $max\_tree\_depth = 7$ and $min\_child\_weight = 4$, we are able to achieve an RMSE of 1959.3.

VI. Support Vector Regression

Another non-linear model we explored was Support Vector Regression. It is similar to the SVM algorithm discussed in class, but for a regression task. It allows us to explore higher dimensional feature space using different kernels. Unsurprisingly, SVR performed better than linear regression.

TABLE III
SVR Results

Kernel	RMSE (CV)	RMSE (Training)
Linear	2248	2279
Polynomial	2407	2398
RBF	2162	2129
Sigmoid	2203	2184

One challenge of training an SVR model is the long training time, due to the complexity of the model. This makes it harder to experiment, and inhibits the ability to rapidly iterate. Training a single SVM model takes more than 12 hours using our entire dataset, which makes it infeasible to fully tune and optimize. The results shown are using a subset of the training examples and a subset of the dev examples.

VII. Model Comparison

The final model comparison can be seen in figure 9. Gradient boosted tree had the best performance out of all of the models by a substantial margin.

VIII. Analysis and Insight

A. Tree Methods: Comparison

Our RandomForest (RF) and GradientBoostedTree (GBT) learning methods yielded similar results, with GBT having an edge. Both of these are ensemble methods that that train many models and get results for each, but they take different approaches to aggregating their results.

RF makes use of bootstrap aggregation (bagging), which is simply choosing (with replacement) a subset

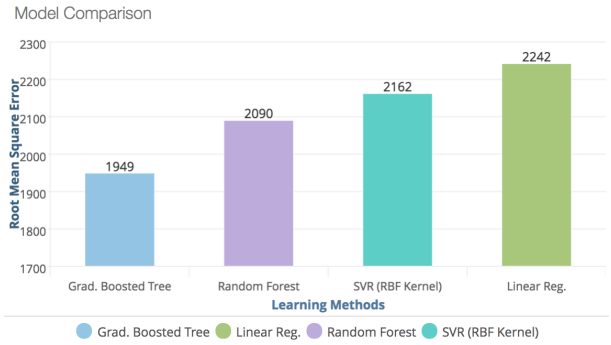

Fig. 9. Final Model Comparison

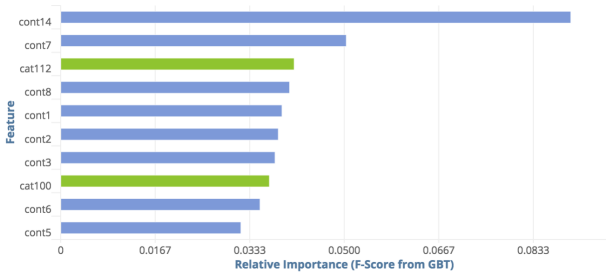

Fig. 10. Relative Feature Importance

of the data. GBT takes this a step further by weighting certain data points (boosting) such that they have a larger effect on the resulting model.

Whenever a data point is misclassified in a GBT, subsequent runs will weight that point higher such that it gets emphasized. This sequential learning gives GBT a clear advantage over RF for our claims regression problem, and helps to show why it achieved the best loss out of all the learning methods we implemented.

B. Ensemble Attempt

We took the output of the Linear Regression model and fed it in as an additional feature to the Gradient Boosted Tree (GBT). Surprisingly, this resulted in an increase in the loss, with an RMSE of 1969.84. Our intuition as to why this was the case follows.

The GBT was the best tree method (compared to RandomForest). Additionally, GBT seems to have been able to capture all of the descriptive power of the data. Therefore using the output of linear regression as an additional feature did not improve the performance of GBT.

C. Feature Importance

Figure 10 shows the relative feature importance using F-Score from the decision trees. Unsurprisingly the continuous features yielded the most information. Eight of the top 10 features by F-score were continuous (seen in blue).

Since the features are anonymized, we unfortunately can't determine what real-world measures they map to!

REFERENCES

[1] Ho, Tin Kam (1995). Random Decision Forests. Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 1416 August 1995. pp. 278282.

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

[3] Jain, Aarshay. Complete Guide to Parameter Tuning in XGBoost, 2016.

[4] Brownlee, Jason. A Gentle Introduction to XGBoost for Applied Machine Learning, 2016.

[5] Welling, Max. Support Vector Regression , Department of Computer Science University of Toronto.

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