 doi:10.20944/preprints202203.0206.v1

*Article*

# Machine Learning in Ratemaking, an Application in Commercial Auto Insurance

**Spencer Matthews [1,*] and Brian Hartman [2]**

[1] University of California - Irvine, Donald Bren School of Information and Computer Science, Department of Statistics, Irvine, CA 92697, USA
[2] Brigham Young University, College of Physical and Mathematical Sciences, Department of Statistics, Provo, UT 84602, USA; hartman@stat.byu.edu
* Correspondence: spencer.matthews@uci.edu

**Abstract:** This paper explores the tuning and results of two-part models on rich datasets provided through the Casualty Actuarial Society (CAS). These data sets include BI (bodily injury), PD (property damage) and COLL (collision) coverage, each documenting policy characteristics and claims across a four year period. The datasets are explored, including summaries of all variables, then the methods for modeling are set forth. Models are tuned and the tuning results are displayed, after which we train the final models and seek to explain select predictions. All of the code will be made available on GitHub. Data was provided by a private insurance carrier to the CAS after anonymizing the data set. This data is available to actuarial researchers for well-defined research projects that have universal benefit to the insurance industry and the public. Our hope is that the methods demonstrated here can be a good foundation for future ratemaking models to be developed and tested more efficiently.

**Keywords:** Ratemaking; Machine Learning; Explainability; Auto Insurance

## 1. Introduction

In recent years, the rise of machine learning models in predictive analytics has led to widespread adoption of these methods. They have proven to be effective modeling techniques in many situations, but have not yet been widely adapted in the auto insurance ratemaking due to the difficulty of explaining the results to regulators (Akinyemi and Leiser 2020). Nevertheless, recent breakthroughs in explainable machine learning potentially allow for these methods to be more widely applied in a ratemaking setting ( Lundberg et al. 2020).

In this paper, we explore the successes and pitfalls of applying machine learning algorithms to a suite of data sets obtained from the Casualty Actuarial Society. First, we explore the data and consider some of the issues that could arise from data integrity. Then, modeling techniques are described, including the algorithms applied and the methods used for model training. Training results are reported and explored, after which final models are trained. Finally, predictions are made on a test set and then explained using some of the newer techniques alluded to above.

We hope that this exploration of machine learning applied to ratemaking stands as a jumping off point for others who wish to bring the increased accuracy of machine learning algorithms into the field. With both the code and data available, it will be easier to compare new methods to the results posted here

## 2. The Data

The data used in this paper was obtained from the CAS. These datasets document the auto policies of a large US-based insurance company across four years, including many variables describing the policy holder, the time the policy was held for, and the number of claims and costs associated with each policy.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 March 2022 doi:10.20944/preprints202203.0206.v1



### 2.1. The Three Data Sets

These policies are split into three different data sets, BI, PD, and COLL. Each data set has columns for the ultimate cost of the policy and the ultimate claim count of the policy, as well as the exposure (or duration) of the policy. There are a significant number of rows with negative and zero exposures. The exact proportions are documented in Table 1.

| Description | Num Records | % | Exposure | Amount | % | Claim Count | % |
|---|---|---|---|---|---|---|---|
| **BI** | 30,342,067 | 100% | 3,830,558 | 634,080,483 | 100.00% | 32,293 | 100.00% |
| Zero exposure | 6,724,652 | 22.16% | - | 6,958,737 | 1.10% | 367 | 1.14% |
| Negative exposure | 3,885,178 | 12.80% | (33) | 10,848,560 | 1.71% | 606 | 1.88% |
| **PD** | 20,201,841 | 100.00% | 2,665,037 | 520,665,847 | 100.00% | 151,842 | 100.00% |
| Zero exposure | 4,138,323 | 20.48% | - | 6,981,221 | 1.34% | 1,898 | 1.25% |
| Negative exposure | 2,590,939 | 12.83% | (129) | 9,330,567 | 1.79% | 2,487 | 1.64% |
| **COLL** | 30,285,873 | 100.00% | 3,835,828 | 443,291,671 | 100.00% | 135,419 | 100.00% |
| Zero exposure | 6,634,314 | 21.91% | - | 5,078,430 | 1.15% | 1,621 | 1.20% |
| Negative exposure | 3,889,473 | 12.84% | (118) | 7,738,811 | 1.75% | 2,291 | 1.69% |

**Table 1.** General Description of the three datasets

### 2.2. The Variables and Values

The data consists of the columns "EARNED_EXPOSURE", "ULTIMATE_AMOUNT", "ULTIMATE_CLAIM_COUNT", and explanatory variables "X_VAR1" through "X_VAR46".

- "EARNED_EXPOSURE" is the amount of time the policy was taken out for.
- "ULTIMATE_AMOUNT" is the claim cost developed to ultimate, in US dollars. Generally the development factors are small in our datasets.
- "ULTIMATE_CLAIM_COUNT" is the number of claims filed for the given policy again developed to ultimate. We rounded these values to the nearest whole number because the development factors were not generally large and we wanted to use discrete models.
- "X_VAR1" through "X_VAR46" are 46 unidentified variables, all of which are discrete (ordinal numeric or text string). We do know that "X_VAR27" is state and "X_VAR41" is year, but do not know the individual levels.

We will explore each of these variables for all three data sets in the sections that follow.

#### 2.2.1. EARNED_EXPOSURE

As mentioned above, EARNED_EXPOSURE is one of the more difficult variables to deal with, as it has many negative and zero values. In our exploration and modeling of the data, we removed rows containing negative or zero values for EARNED_EXPOSURE. A summary of EARNED_EXPOSURE across the three different data sets is given in Table 2.

| Data Set | Minimum | First Quartile | Median | Mean | Third Quartile | Maximum |
|---|---|---|---|---|---|---|
| BI | 0.0001 | 0.0822 | 0.2110 | 0.2385 | 0.3973 | 1.0028 |
| PD | 0.0001 | 0.0822 | 0.2138 | 0.2418 | 0.4028 | 1.9996 |
| COLL | 0.0001 | 0.0822 | 0.2110 | 0.2384 | 0.3973 | 0.9998 |

**Table 2.** Summary of EARNED_EXPOSURE across the three data sets

#### 2.2.2. ULTIMATE_AMOUNT and ULTIMATE_CLAIM_COUNT

ULTIMATE_AMOUNT and ULTIMATE_CLAIM_COUNT are dealt with jointly, as they are similar in scope, and together can be used to compute the severity (or cost per claim) of a policy. Both these variables are extremely right-skewed due to the fact that such few rows actually have claims. Table 3 shows the percentage of rows in each data set where the ULTIMATE_AMOUNT was greater than zero and Table 4 shows the number of rows for given numbers of claim counts in each data set.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 March 2022 doi:10.20944/preprints202203.0206.v1



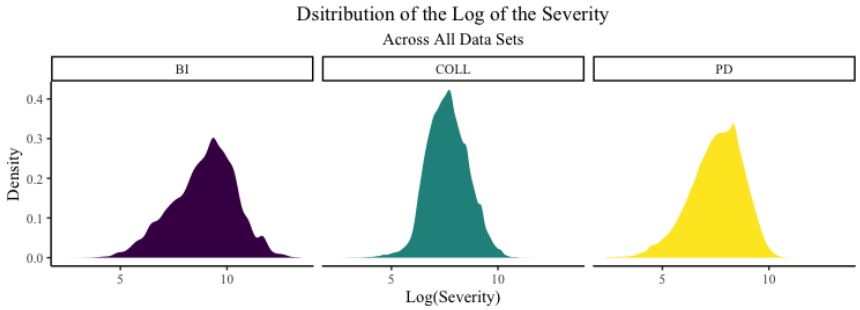


**Figure 1.** Distribution of the log of the severity for all data sets

| Data Set | % of Rows with ULTIMATE_AMOUNT above 0 |
|---|---|
| BI | 0.16% |
| PD | 1.06% |
| COLL | 0.64% |

**Table 3.** Percentages of positive ULTIMATE_AMOUNT in the different data sets

| | **BI** | | **PD** | | **COLL** | |
|---|---|---|---|---|---|---|
| Claim Count | Num Rows | % | Num Rows | % | Num Rows | % |
| 0 | 19,700,470 | 99.8% | 13,329,135 | 98.9% | 19,636,069 | 99.4% |
| 1 | 31,531 | 0.16% | 141,107 | 1.05% | 120,681 | 0.61% |
| 2 | 218 | 0.001% | 2,239 | 0.02% | 5,260 | 0.03% |
| 3+ | 18 | 0.0001% | 98 | 0.001% | 76 | 0.0004% |

**Table 4.** Counts of the rounded values of ULTIMATE_CLAIM_COUNT

A little more insight can be gleaned when looking at the distribution of the severity. The severity is the ULTIMATE_AMOUNT divided by the ULTIMATE_CLAIM_COUNT, only including those rows that had a claim. Summaries for the severity for each data set are given in Table 5 and density plots are shown in Figure 1. Note that the density plots are given in the log scale on the x axis.

| Data Set | Minimum | First Quartile | Median | Mean | Third Quartile | Maximum |
|---|---|---|---|---|---|---|
| BI | 1.1 | 2,842 | 8,821 | 19,231 | 21,300 | 732,953 |
| PD | -9,506 | 856 | 2,095 | 3,468 | 4,553 | 88,494 |
| COLL | -2,850 | 1,061 | 2,027 | 3,294 | 3,979 | 101,203 |

**Table 5.** Summary statistics for the severity for each data set

### 2.2.3. X_VARs

The discrete (and sometimes ordinal) variables X_VAR1 through X_VAR46 are quite varied. Some have many levels while others have few; some are balanced in their distribution between levels but others are not. In Table 6, we give a characterization of all variables including a brief description of each one. More comprehensive data regarding each individual level of every variable can be found in the GitHub repository. See Appendix B for more information.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 March 2022 doi:10.20944/preprints202203.0206.v1



| Variable | Unique Values | Description |
|---|---|---|
| X_VAR1 | 5 | Integer, ranging from 0 to 4 inclusive |
| X_VAR2 | 2 | Binary, 0 or 1. Nearly all observations are 0 so it is thrown out in modeling |
| X_VAR3 | 5 | Integer, ranging from 0 to 4 inclusive |
| X_VAR4 | 31 | Integer, ranging from 1 to 31 inclusive |
| X_VAR5 | 5 | Integer, ranging from 0 to 4 inclusive |
| X_VAR6 | 17 | Integer, ranging from 0 to 16 inclusive |
| X_VAR7 | 17 | Integer, ranging from 0 to 16 inclusive |
| X_VAR8 | 5 | Integer, ranging from 0 to 4 inclusive |
| X_VAR9 | 10 | Integer, ranging from 1 to 10 inclusive |
| X_VAR10 | 2 | Character, "A" or "B" |
| X_VAR11 | 8 | Integer, ranging from 1 to 8 inclusive |
| X_VAR12 | 5 | Integer, ranging from 0 to 4 inclusive |
| X_VAR13 | 4 | Character, "B", "C", "D", or "E" |
| X_VAR14 | 5 | Integer, ranging from 0 to 4 inclusive |
| X_VAR15 | 4 | Integer, ranging from 0 to 3 inclusive |
| X_VAR16 | 9 | Integer, ranging from 1 to 9 inclusive |
| X_VAR17 | 5 | Integer, ranging from 0 to 4 inclusive |
| X_VAR18 | 7 | Character, "A", "B", or "E" through "I" |
| X_VAR19 | 2,062 | A string with the form of an integer between 1 and 2529 inclusive preceded by an "A". Left out of modeling due to the high number of unique values. |
| X_VAR20 | 5 | Integer, ranging from 0 to 4 inclusive |
| X_VAR21 | 17 | Integer, ranging from 0 to 16 inclusive |
| X_VAR22 | 23 | Integer, ranging from 1 to 23 inclusive |
| X_VAR23 | 11 | Character, either a digit 1 through 9 inclusive, the letter "U" or missing |
| X_VAR24 | 11 | Integer, ranging from 1 to 11 inclusive |
| X_VAR25 | 24 | Integer, ranging from 1 to 24 inclusive |
| X_VAR26 | 5 | Integer, ranging from 0 to 4 inclusive |
| X_VAR27 | 48 | String of two letters, alphabetically between "AA" and "BY" inclusive. This variable represents state, but the mapping of the levels to the states is unknown. |
| X_VAR28 | 6 | Integer, ranging from 1 to 6 inclusive |
| X_VAR29 | 5 | Integer, ranging from 0 to 4 inclusive |
| X_VAR30 | 8 | Integer, ranging from 1 to 8 inclusive |
| X_VAR31 | 26 | Integer, ranging from 1 to 26 inclusive |
| X_VAR32 | 5 | Integer, ranging from 0 to 4 inclusive |
| X_VAR33 | 6 | Integer, ranging from 1 to 6 inclusive |
| X_VAR34 | 56,817 | String consisting of the letter "A" followed by a number. Due to the high number of unique levels, it is left out of modeling. |
| X_VAR35 | 5 | Integer, ranging from 0 to 4 inclusive |
| X_VAR36 | 5 | Integer, ranging from 0 to 4 inclusive |
| X_VAR37 | 5 | Integer, ranging from 0 to 4 inclusive |
| X_VAR38 | 31 | A string consisting of the letter "B" followed by an integer between 1 and 31 inclusive |
| X_VAR39 | 16 | Integer, ranging from 1 to 16 inclusive |
| X_VAR40 | 17 | Integer, ranging from 0 to 16 inclusive |
| X_VAR41 | 4 | Integer, ranging from 1 to 4 inclusive. This variable is the year of the policy, but the policies are not evenly dispersed across the 4 levels. |
| X_VAR42 | 5 | Integer, ranging from 0 to 4 inclusive |
| X_VAR43 | 11 | Integer, ranging from 1 to 11 inclusive |
| X_VAR44 | 5 | Integer, ranging from 0 to 4 inclusive |
| X_VAR45 | 17 | Integer, ranging from 0 to 16 inclusive |
| X_VAR46 | 31,064 | String with 5 characters, all numbers and letters. Due to the high number of unique values it is left out of modeling |

**Table 6.** Characteristics of the predictor variables

 doi:10.20944/preprints202203.0206.v1



## 3. Modeling Techniques

In order to model the data, we use a two-part model where the first part predicts the frequency of claims and the second part predicts the severity, or cost per claim. These models are tuned separately, on the same set test data. Here we discuss the data preparation that occurred before model tuning, as well as the methodology we used to tune the different models.

### *3.1. Data Preparation*

The initial data came in 21 separate files. There are 8 files for the BI data, 5 for the PD data, and 8 for the COLL data. In order to work with this data as three separate datasets, we combined these files in three distinct files. During this process we filtered out any records where EARNED_EXPOSURE was less than or equal to zero, and we also rounded ULTIMATE_CLAIM_COUNT to the nearest integer. Furthermore, all rows with an ULTIMATE_CLAIM_COUNT greater than three were set to three.

Once the data was cleaned, we needed to create the train, validation, and test sets of data for all three data sets. This was done using the {caret} package function createDataPartition in R, which gives a stratified sample along a variable so as to make sure the proportions are similar in all three data sets (Kuhn 2008). We used a split of 50% training data, 25% validation data, and 25% test data for all three data sets.

Due to the imbalanced nature of the data sets, we also wanted to ensure that the model was not extremely biased towards predicting zero. In order to accomplish this, we re-balanced the training set so that it was perfectly balanced between the four classes (0 claims, 1 claim, 2 claims, 3+ claims). This required a combination of up-sampling and down-sampling, where the 0 claim class was down-sampled to 3 million observations and the 1, 2, and 3 claim classes were up-sampled to have 3 million observations each. Using bootstrap sampling (through the sample_n function in the {dplyr} package) we were able to create training sets that were balanced in terms of frequency, with 12 million rows (Wickham et al. 2018).

During the modeling stages, we used the X_VARs as the predictors and the severity or ULTIMATE_CLAIM_COUNT as the response (depending on which part of the model we were tuning). However, we dropped four of the X_VARs because they caused issues in tuning. X_VAR2 was often constant across the entire training set, and so was not useful for predicting. X_VAR19, X_VAR34, and X_VAR46 contained thousands of unique levels as categorical predictors and were dropped due to the model complexity caused by the number of levels. Of the variables that remained, those with only numbers as levels were treated as ordinal (a regular numeric predictor) and those with characters defining categories were treated as categorical predictors.

### *3.2. Model Tuning*

Once the training, validation, and testing data were obtained, we needed to tune the models. We worked in the h2o framework inside of R to rapidly tune the models (H2O.ai 2021). The basic layout for both parts was to loop over a predefined tuning grid where the parameters were set to different values. In each loop, use the balanced training data to train the model, and then use the validation data to compute metrics on that model. In each loop we also computed the model predictions on the test data set, and saved them to a prediction data frame. After storing the model tuning parameters and resulting metrics as a row of a data frame, the algorithm continued to the next set of tuning parameters. This process was repeated for all tuning parameters we defined, for all model types and across all three data sets.

#### 3.2.1. Frequency Model Tuning

To model claim frequency, we used the multinomial family of models with our four possible classes (0 claims, 1 claims, 2 claims, and 3 claims). We first created a baseline generalized linear model, to ensure that the predictions from the machine learning

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 March 2022 doi:10.20944/preprints202203.0206.v1



methods improved upon its accuracy. Then, models were developed using a random forest algorithm, a gradient boosted forest algorithm, and a deep learning algorithm (all as implemented in H2O).

Table 7 shows the tuning parameters used for the random forest models, Table 8 shows the tuning parameters used for the gradient boosted forest models, and Table 9 shows the tuning parameters used for the deep learning models.

| Model Parameter | Possible Values |
|---|---|
| ntrees | 100 |
| max_depth | 20, 15, 10, 7, 5 |
| min_split_improvement | 0.001, 0.01 |
| mtries | 20, 7 |
| histogram_type | "RoundRobin" |
| sample_rate | 0.632 |
| categorical_encoding | "EnumLimited" |
| col_sample_rate_per_tree | 0.8 |
| seed | 16 |

**Table 7.** Tuning Parameters for Random Forest Frequency Models

| Model Parameter | Possible Values |
|---|---|
| ntrees | 300, 500, 1000 |
| max_depth | 1, 2, 3, 5, 7, 10 |
| learn_rate | 0.001, 0.0001 |
| min_split_improvement | 0.0001 |
| distribution | "multinomial" |
| sample_rate | 0.632 |
| nbins_cats | 56 |
| categorical_encoding | "Eigen" |
| col_sample_rate_per_tree | 0.8 |
| seed | 16 |

**Table 8.** Tuning Parameters for Gradient Boosted Forest Frequency Models

| Model Parameter | Possible Values |
|---|---|
| activation | "Tanh" |
| hidden | 100, [100, 100], [200, 200], [100, 100, 100] |
| adaptive_rate | FALSE |
| rate | 0.1, 0.01, 0.005, 0.001, .0005 |
| rate_decay | 0.5 |
| momentum_start | 0.5 |
| momentum_stable | 0.99 |
| input_dropout_ratio | 0.1 |
| initial_weight_distribution | "Normal" |
| initial_weight_scale | 1 |
| loss | "Automatic" |
| distribution | "multinomial" |
| stopping_metric | "logloss" |
| stopping_tolerance | 0.001 |
| categorical_encoding | "EnumLimited" |
| seed | 16 |
| mini_batch_size | 100 |

**Table 9.** Tuning Parameters for Deep Learning Frequency Models

 doi:10.20944/preprints202203.0206.v1



3.2.2. Severity Model Tuning

Severity model tuning proceeded in a similar way to the frequency model tuning. First, a baseline model was created using a simple linear model to predict the severity. Then, random forest, gradient boosted, and deep learning models were tuned in H2O to predict the severity. All these models were regression models, trying to predict the numerical value of severity. Because the severity is very right-skewed, we also tuned the models to predict the log of the severity with each model type and for each tuning parameter combination. This helped improve the predictive power of the models in many cases.

The severity models tuned much more quickly than the frequency models because before tuning the models we dropped all duplicate rows from the training data set and filtered where ULTIMATE_AMOUNT was greater than zero. This was possible because severity was not unbalanced, so we did not need to up-sample or down-sample any part of the data. Dropping duplicate rows allowed us to essentially undo the up-sampling that was done to all rows with a claim.

Table 10 lists the tuning parameters used for the severity random forest models, Table 11 lists the tuning parameters used for the severity gradient boosted forest models, and Table 12 lists the tuning parameters used for the severity deep learning models.

| Model Parameter | Possible Values |
|---|---|
| ntrees | 100, 200 |
| max_depth | 3, 5, 7, 10, 15, 20, 30 |
| min_split_improvement | 0.01, 0.001, 0.0001 |
| mtries | -1, 20, 7 |
| histogram_type | “UniformAdaptive”, “RoundRobin” |
| sample_rate | 0.632 |
| categorical_encoding | “EnumLimited” |
| col_sample_rate_per_tree | 0.8 |
| seed | 16 |

**Table 10.** Tuning Parameters for Random Forest Severity Models

| Model Parameter | Possible Values |
|---|---|
| ntrees | 300, 500, 1000 |
| max_depth | 1, 2, 3, 5, 7, 10 |
| learn_rate | 0.001, 0.0001 |
| min_split_improvement | 0.0001 |
| distribution | “gaussian”, “gamma”, “laplace”*, “huber”* |
| sample_rate | 0.632 |
| nbins_cats | 56 |
| categorical_encoding | “Eigen” |
| col_sample_rate_per_tree | 0.8 |
| seed | 16 |

**Table 11.** Tuning Parameters for Gradient Boosted Forest Severity Models

*Laplace and Huber distributions only used when not predicting the log of the severity

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 March 2022 doi:10.20944/preprints202203.0206.v1



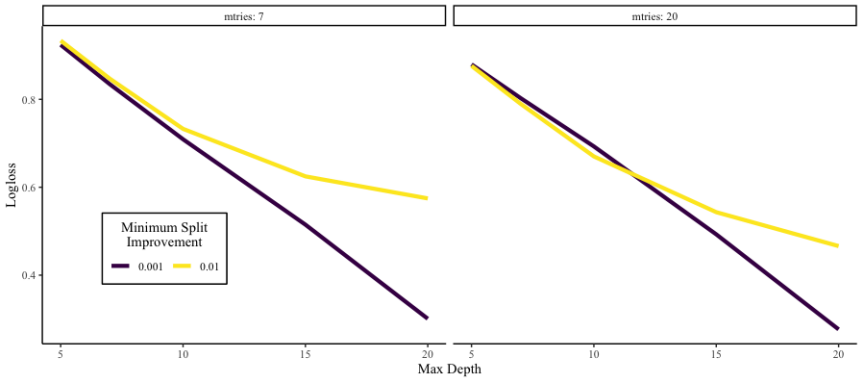


**Figure 2.** Visualization of the results from tuning random forests to predict claim frequency on the BI data

| Model Parameter | Possible Values |
|---|---|
| activation | "Tanh" |
| hidden | 100, [100, 100], [200, 200], [100, 100, 100] |
| adaptive_rate | FALSE |
| rate | 0.01, 0.001, 0.0001, 0.00001 |
| rate_decay | 0.5 |
| momentum_start | 0.5 |
| momentum_stable | 0.99 |
| input_dropout_ratio | 0.1 |
| initial_weight_distribution | "Normal" |
| initial_weight_scale | 1 |
| loss | "Automatic" |
| distribution | "gaussian", "laplace" |
| stopping_metric | "MAE" |
| stopping_tolerance | 0.001 |
| categorical_encoding | "EnumLimited" |
| seed | 16 |
| mini_batch_size | 50 |

**Table 12.** Tuning Parameters for Deep Learning Severity Models

## 4. Tuning Results

After the tuning process was complete across all three datasets for all model types and parameter combinations, we set to work visualizing and summarizing the tuning results. In this section we will explain the tuning results for each data set and each part of the model, note that the metrics shown in these parts are calculated on the validation data set. We will then look at the results when the predictions are combined and the metrics of the overall two-part model are computed on the test data set.

### *4.1. BI Data*

The models on the BI data performed the poorest out of the three datasets, which is understandable. Bodily Injury claims are the least common, and when they do occur they have the largest range of severity. This leads to a lot of noise in the models and overall poorer predictive performance.

#### 4.1.1. Frequency Models

The most accurate frequency model was a deep neural network, but the neural networks also had the largest range of accuracy. Random Forests performed best on average across all tuning parameters.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 March 2022 doi:10.20944/preprints202203.0206.v1



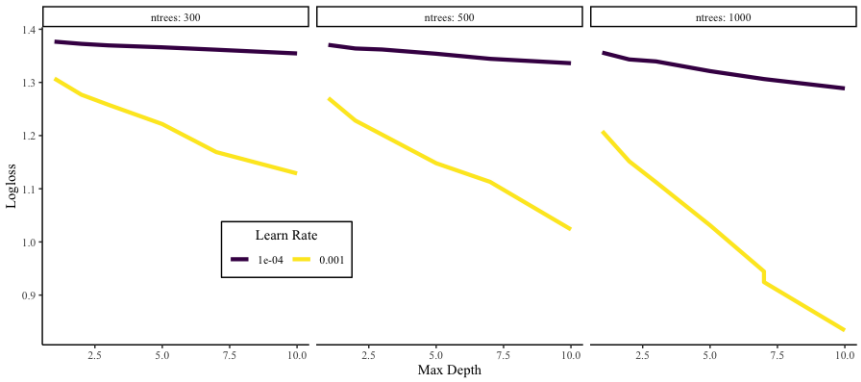


**Figure 3.** Visualization of the results from tuning gradient boosted forests to predict claim frequency on the BI data

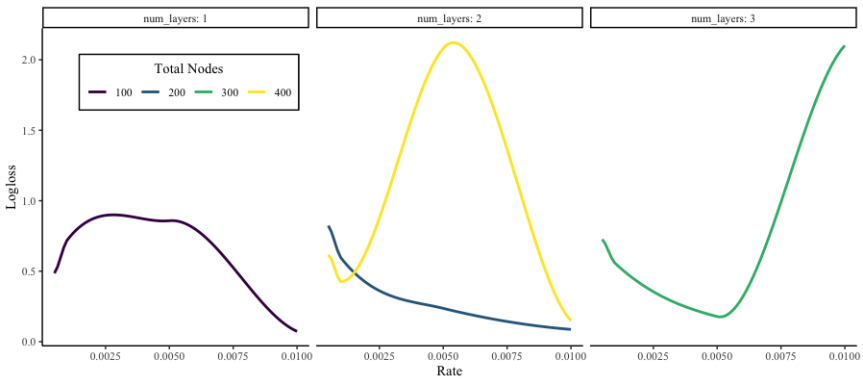


**Figure 4.** Visualization of the results from tuning deep learning models to predict claim frequency on the BI data

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 March 2022 doi:10.20944/preprints202203.0206.v1



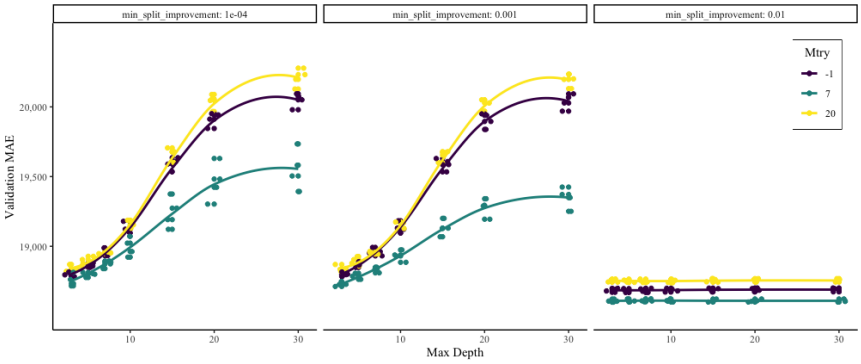


**Figure 5.** Visualization of the results from tuning random forests to predict severity on the BI data

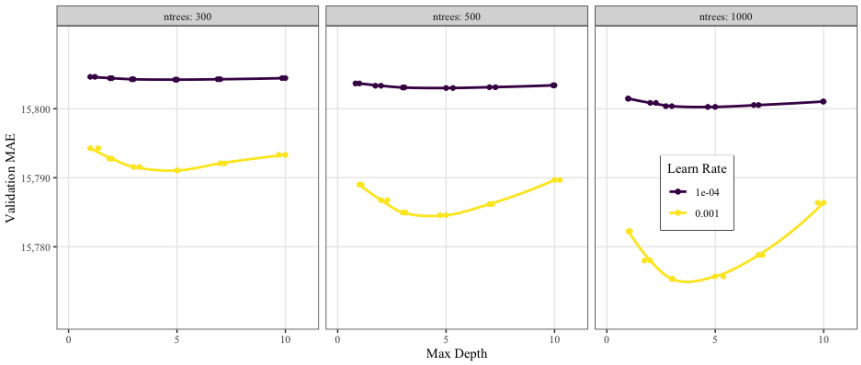


**Figure 6.** Visualization of the results from tuning gradient boosted forests to predict severity on the BI data (note that only models with the distribution tuning parameter set to "Laplace" are shown since their MAE was much less than the other models)

#### 4.1.2. Severity Models

In contract, neural networks performed quite poorly on severity for the BI data. Instead, we found that gradient boosted models worked best, especially those with a smaller learn rate. Nonetheless, the best MAE was well over 15,000 which is less than desirable.

### *4.2. PD Data*

The PD and COLL datasets were more conducive to modeling and running those datasets through the tuning process resulted in more accurate models, in terms of logloss and MAE. The PD dataset has the most claims out of the three datasets, and the least number of records. These properties allow greater relationships to be drawn between covariates and response and result in more accurate models.

#### 4.2.1. Frequency Models

However, most of the increase in accuracy comes from the severity part of the model. The frequency model tuning results are similar to those seen in the BI data, with the random forest models being most consistent and hovering between .4 and 1. The gradient boosted models performed quite poorly in predicting frequency on the PD dataset, and there were some difficulties in tuning the neural networks that prohibited the entire tuning grid from being run.

#### 4.2.2. Severity Models

The severity models for the PD dataset did much better than on the BI dataset, with MAEs in the 2,500 range. The most accurate model was a gradient boosted forest, although neural networks based on the Laplace distribution also performed very well.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 March 2022 doi:10.20944/preprints202203.0206.v1



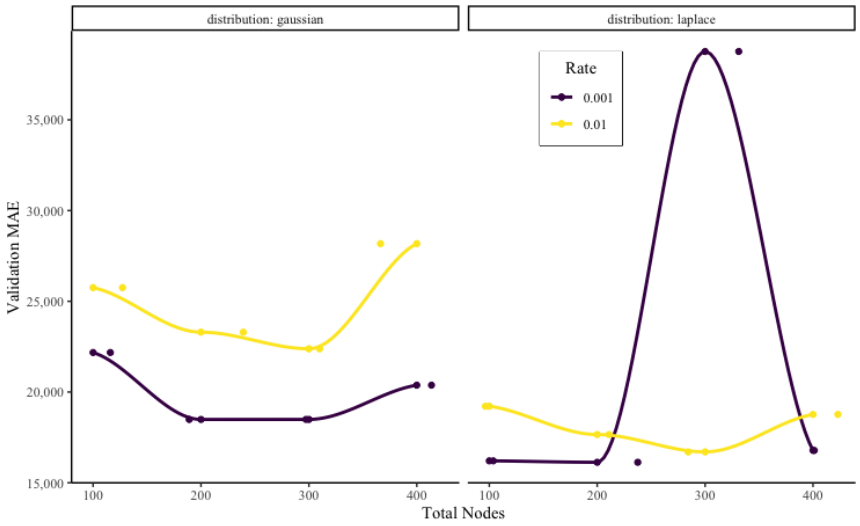


**Figure 7.** Visualization of the results from tuning deep learning models to predict severity on the BI data

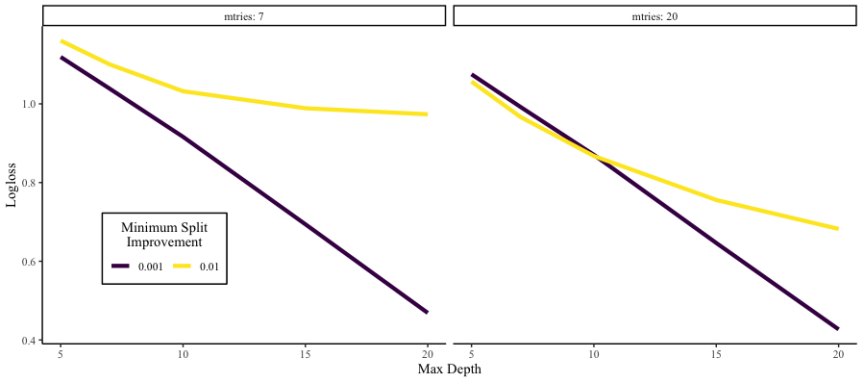


**Figure 8.** Visualization of the results from tuning random forests to predict claim frequency on the PD data

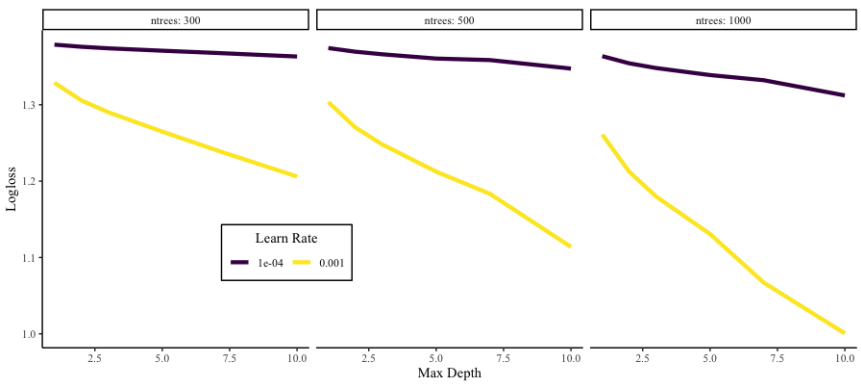


**Figure 9.** Visualization of the results from tuning gradient boosted forests to predict claim frequency on the PD data

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 March 2022 doi:10.20944/preprints202203.0206.v1



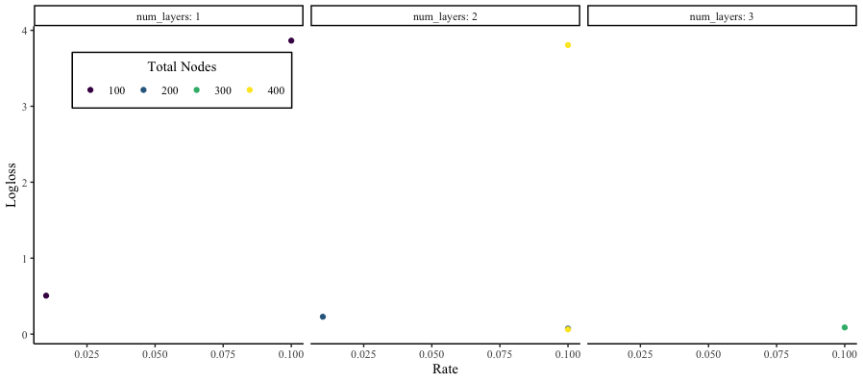


**Figure 10.** Visualization of the results from tuning deep learning models to predict claim frequency on the PD data

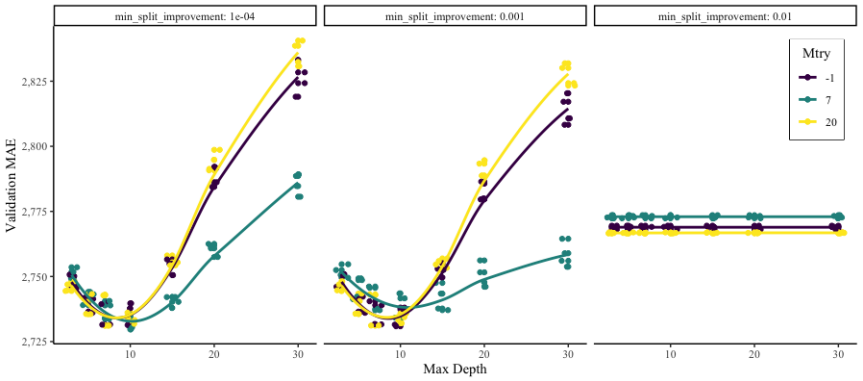


**Figure 11.** Visualization of the results from tuning random forests to predict severity on the PD data

The model tuning in these plots shows how the tuning grid did a sufficient job of exploring the parameter space to find at least a local minimum in many cases.

### *4.3. COLL Data*

Again, the COLL data was more adept for modeling than the BI data. These models are the most accurate, and the tuning results give us the best metrics.

#### 4.3.1. Frequency Models

Both the random forest models and the neural network models performed very well in predicting the frequency of claims across the dataset. The gradient boosted models again performed poorly.

#### 4.3.2. Severity Models

The severity models on the COLL data had the lowest MAE out of all model tuning across the three datasets. And again, neural networks and gradient boosted both performed strongly, while random forest lagged behind. This is likely due to the inability to specify a target distribution when training a random forest, whereas we were able to specify a heavy-tailed distribution such as the Laplace distribution for the gradient boosted and neural network models.

### *4.4. Two-Part Model*

In additional to looking at the model tuning results for each individual part (frequency and severity) of the two-part model, we also compared the predictions of the two-part model against the test dataset. For each frequency and severity model, we predicted the test dataset, and saved those predictions. Once all model tuning was

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 March 2022 doi:10.20944/preprints202203.0206.v1



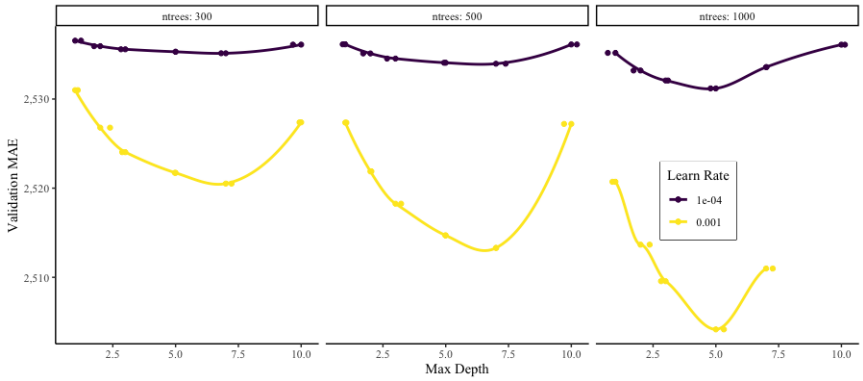


**Figure 12.** Visualization of the results from tuning gradient boosted forests to predict severity on the PD data (note that only models with the distribution tuning parameter set to "Laplace" are shown since their MAE was much less than the other models)

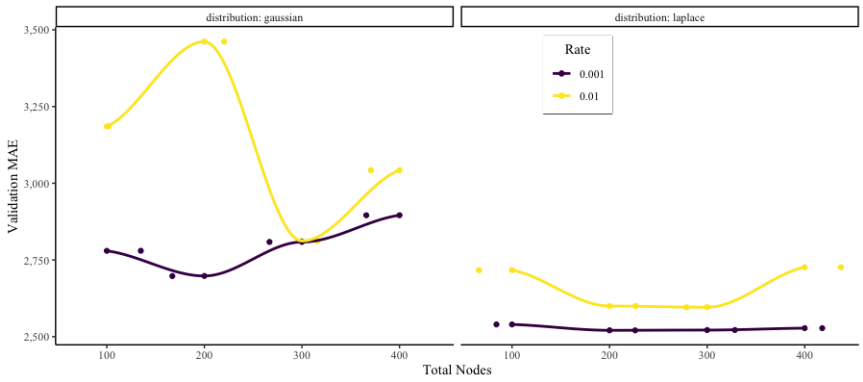


**Figure 13.** Visualization of the results from tuning deep learning models to predict severity on the PD data

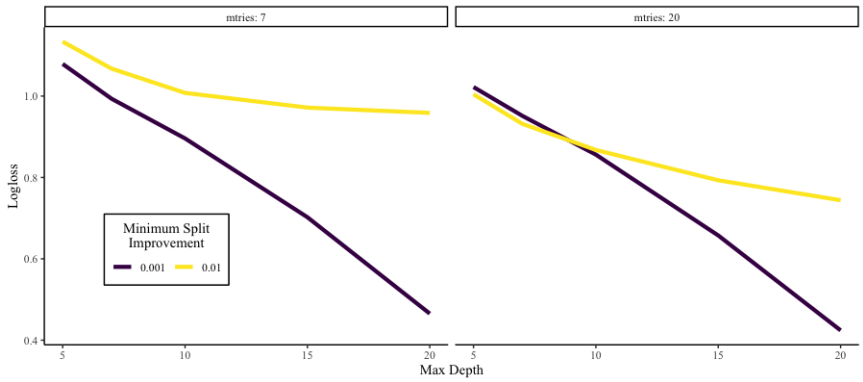


**Figure 14.** Visualization of the results from tuning random forests to predict claim frequency on the COLL data

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 March 2022 doi:10.20944/preprints202203.0206.v1



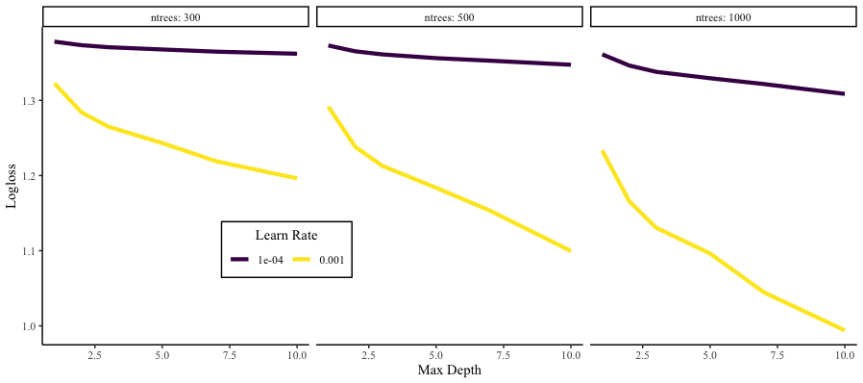


**Figure 15.** Visualization of the results from tuning gradient boosted forests to predict claim frequency on the COLL data

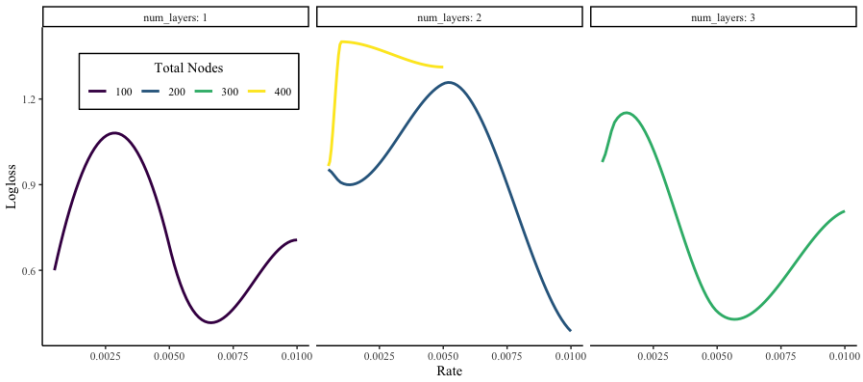


**Figure 16.** Visualization of the results from tuning deep learning models to predict claim frequency on the COLL data

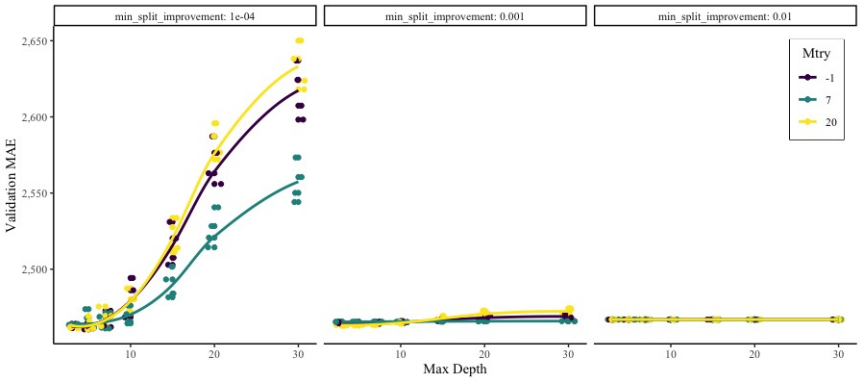


**Figure 17.** Visualization of the results from tuning random forests to predict severity on the COLL data

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 March 2022 doi:10.20944/preprints202203.0206.v1



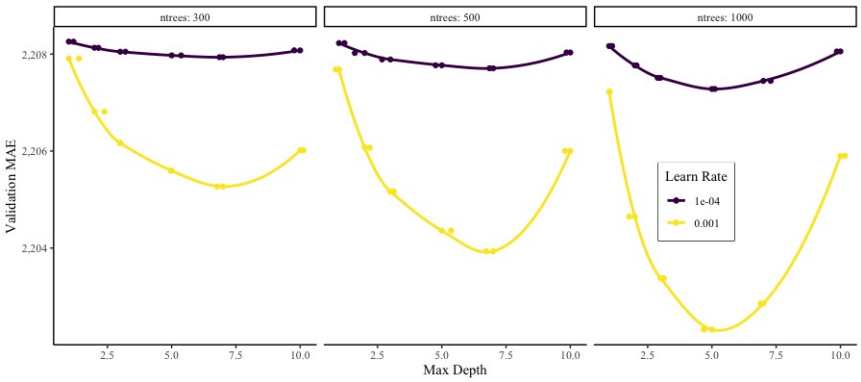


**Figure 18.** Visualization of the results from tuning gradient boosted forests to predict severity on the COLL data (note that only models with the distribution tuning parameter set to "Laplace" are shown since their MAE was much less than the other models)

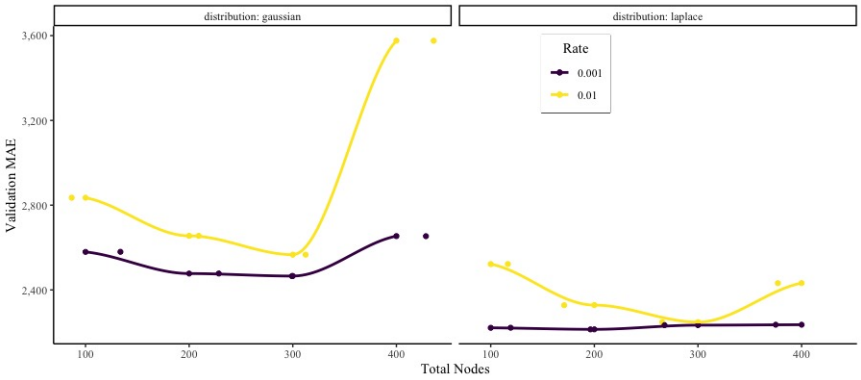


**Figure 19.** Visualization of the results from tuning deep learning models to predict severity on the COLL data

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 March 2022 doi:10.20944/preprints202203.0206.v1



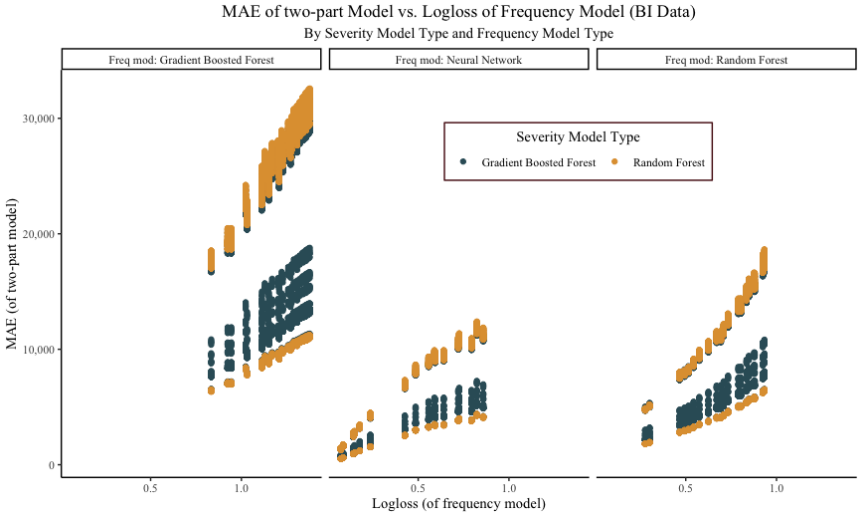


**Figure 20.** Comparison of the logloss of the frequency model part to the MAE of the overall two-part model for the BI data

finished, we were able to compare two-part model predictions to the actual value of the test set. To do this, we computed the expected value as predicted by the two-part model for each policy in the test set for every combination of frequency and severity model.

In general, the better the frequency predictions and the better the severity predictions, the better the overall two-part model. However, the best individual models did not constitute the best two-part model. This is likely due to the fact that the individual model parts were measured using the validation dataset, while the overall two-part models were measured with the test dataset.

The results of the two-part models can be compared with the performance of the individual model parts and then visualized. One of the most interesting visualizations was to view the performance of the frequency model against the performance of the two-part model. In Figures 20, 21, and 22 we see the resulting plots for the BI, PD, and COLL datasets.

As was seen in the training of the frequency models, the neural networks and random forests performed best for predicting frequency while the gradient boosted forests lagged behind. Also interesting is the convexity of the trend between the different model types. While the MAE of the two-part model began to slow its rate of increase as the logloss of the frequency model increase for neural networks, it seems to generally speed up in the case of random forests in the same situation.

We can also compare the performance of the severity model parts to the performance of the two-part model. For concision, we have placed these plots in Appendix D.

## 5. Final Models

In the end, we determined the best final model for each dataset based on the MAE computed on the test set of the two-part model.

### *5.1. BI Final Model*

For the BI data, the best two-part model was made up of a neural network frequency model and a neural network severity model. The frequency model had 100 nodes arrayed in a single layer, and a learning rate of 0.01. The severity model had 300 nodes in three layers, with each layer having 100 nodes each, trained on the log of the severity, and also had a learning rate of 0.01. The overall two-part model had an MAE of 476.41 on the test dataset when predicting the expected value of the cost.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 March 2022 doi:10.20944/preprints202203.0206.v1



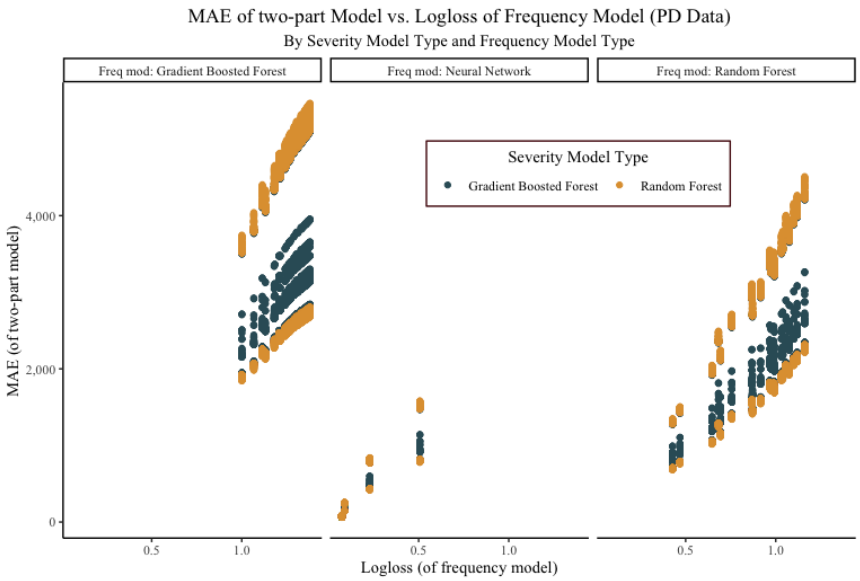


**Figure 21.** Comparison of the logloss of the frequency model part to the MAE of the overall two-part model for the PD data

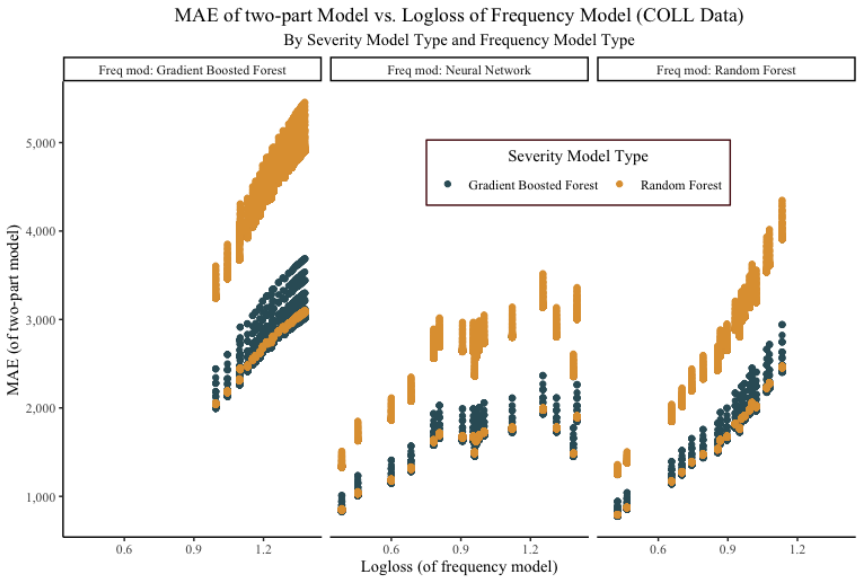


**Figure 22.** Comparison of the logloss of the frequency model part to the MAE of the overall two-part model for the COLL data

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 March 2022 doi:10.20944/preprints202203.0206.v1



#### 5.1.1. Prediction Concerns for the BI Model

Even though we took care to balance the data, we wanted to ensure that the models with the best MAE were not naively predicting zero. To do this, we compared the mean model prediction against the mean model prediction of all other models. When doing this, our suspicions were confirmed.

The frequency part of the model assigned an average probability of 94% to zero claims, when among the other models we would expect zero claims to be assigned around a 43% probability. It also was the model that assigned the smallest probability to the one claim class, on average. Furthermore, the average severity prediction for the severity model part was about 50% lower than the average model, and only one other severity model had a lower average prediction.

In short, what we found was that in spite of our efforts, the double neural network combination had been over-fit to predict small values. To avoid this for the other two datasets, we will limit our search for the best model to be only gradient boosted or random forest models, which tend to be less over-fit and have more useful predictions. And as a plus, these kinds of models can be explained using the mSHAP algorithm, which will allow use to determine why the models made the predictions that they did ( Matthews and Hartman 2021).

### *5.2. PD Final Model*

When we filtered out the neural network models, the best two-part model for the PD data was made up of random forests for both the severity and frequency models.

For a complete analysis of this final model, including parameters and prediction explanations, see Matthews and Hartman (2021).

### *5.3. COLL Data Final Model*

For the COLL data, the best model (sans neural network type models) consisted of a random forest for frequency and a gradient boosted forest for severity. The random forest for predicting the frequency was made up of 100 trees with a max depth of 20 and each split considering 20 different variables. It had a validation logloss of 0.424, which is fairly good for a four-class classification problem. The gradient boosted forest ran over 1000 trees with a max depth of one and a learn rate of 0.001. This model was trained with the untransformed severity and used a Laplace distribution at its core.

Additionally, the model's predictions are less biased towards zero than we saw with the neural networks in the BI data. The frequency model predicts zero about two-thirds of the time and one 27% of the time. Furthermore, the severity model has an average prediction that is about 50% more than the minimum model prediction, meaning it predicts a significantly higher severity than the lowest models. In all, we are satisfied that the model predicts a fair premium and is not excessively biased towards zero.

In the subsequent section, we will dig deeper into the final model for the COLL data. Due to this, additional plots describing the distribution of the predictions of the final model can be found in Appendix D.

### *5.4. Model Explanation Case Study*

In the final section of this paper, we will explore a two-part model that was computed on the collision dataset. The selected two-part model is the best COLL model where both parts of the model are tree-based algorithms (gradient boosted forests or random forests). This will allow us to present explanations of the final model predictions using the mSHAP algorithm, a necessity in the regulated insurance industry (Matthews and Hartman 2021).

Applying the mSHAP algorithm to a subset of the test dataset allows us to explain the amount each variable contributed to the final prediction. The mSHAP algorithm builds off of SHAP values, and its accuracy comes from advances that permit exact computations of SHAP values (Lundberg et al. 2020; Lundberg and Lee 2017).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 March 2022 doi:10.20944/preprints202203.0206.v1



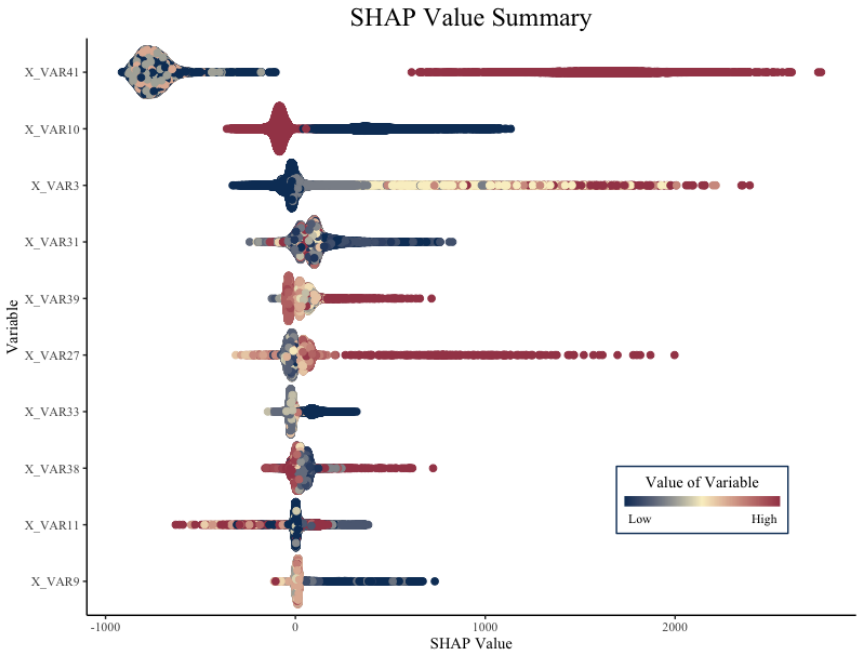


**Figure 23.** A summary of mSHAP values from 50,000 observations in the COLL test dataset

In order to compute the SHAP values (which are necessary for applying mSHAP) of our final model, we trained the model in python using the scikit-learn module ( Pedregosa et al. 2011). Using this, we could then obtain and save the SHAP values, which were saved to a file and then imported into R in order to use the mSHAP R package to get the final contributions (Matthews and Hartman 2021). Since the main purpose of computing the mSHAP values was for visualization, we only computed SHAP values on 50,000 observations in our COLL test dataset.

Figure 23 shows a summary plot of mSHAP values for the 10 most important variables in our final two-part model. We see that X_VAR41, which is known to be year, is the most important variable, and that the state variable (X_VAR27) is also one of the most important variables. This plot also shows trends in the relationship between the variable value and the SHAP value. Generally, the extremes are correlated with extreme variable values, which is interesting.

In addition to a summary plot, the mSHAP values of the final two-part model allow us to create a plot showing how the model arrived at individual predictions. Three example plots are shown in Figure 24, showing three different kinds of predictions.

These plots allow us to explain exactly how the model arrived at its prediction for the expected cost of the given policy. They are created in R using the mSHAP R package, which is available on the CRAN. With plots like these, actuaries can explain to regulators and consumers why the insurance rates are set the way they are, and what factors contribute the most to the insurance cost. Although we have explored many machine learning methods for predicting the loss cost of a policy, this exploration is less valuable if the methods cannot be applied in the field. The mSHAP algorithm provides the potential for two-part models to be implemented in a regulated industry such as insurance.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 March 2022 doi:10.20944/preprints202203.0206.v1



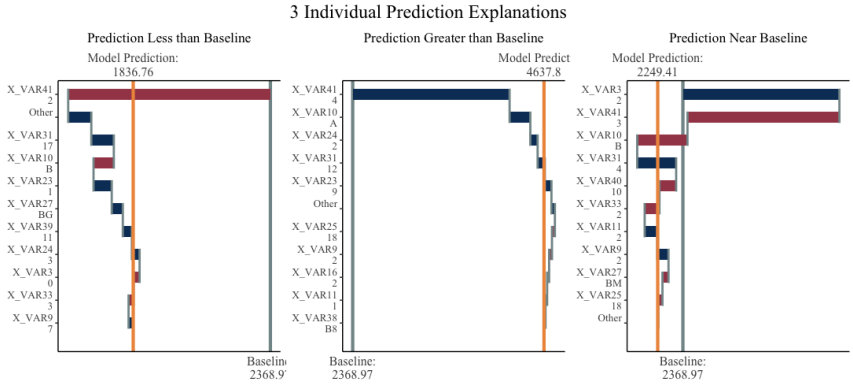


**Figure 24.** An explanation of three different individual predictions on the COLL test dataset using the mSHAP values

## 6. Conclusion

In conclusion, there is great potential for exploration in the area of machine learning applied to two-part ratemaking models. The dataset obtained from the Casualty Actuary Society is a great opportunity for exploring these methods, and we have seen significant improvements over classic generalized models in our results. Additionally, using machine learning in ratemaking is potentially feasible due to the mSHAP algorithm, which explains the predictions of two-part models and allows for transparency in insurance pricing.

**Author Contributions:** Data curation, S.M.; formal analysis, B.H.; funding acquisition, B.H.; investigation, S.M.; methodology, S.M.; project administration, B.H.; writing—original draft, S.M.; writing—review and editing, B.H. All authors have read and agreed to the published version of the manuscript.

**Funding:** This paper was funded by an individual grant from the Casualty Actuarial Society.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Data was provided by a private insurance carrier to the Casualty Actuarial Society (CAS) after anonymizing the data set. This data is available to actuarial researchers for well-defined research projects that have universal benefit to the insurance industry and the public. In order to obtain the data, contact CAS through Brian Fannin with a project proposal.

**Acknowledgments:** Brigham Young University Department of Statistics Computing Cluster; Brian Fanin and the Casualty Actuarial Society for providing the data; and Isabelle Matthews for proofreading.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## Appendix A. Data

The data used in this paper was obtained from the Casualty Actuary Society (CAS). Readers interested in obtaining the data for their own research projects should email Brian Fannin of the CAS a short proposal of the project they hope to do.

## Appendix B. CAS Repo

All the code used in the cleaning of the data, preparation of the data, modeling of the data, and analyzing of the data can be found in the github repo at https://github.com/srmatth/CA Additionally, this repo includes many of the resulting data sets from our analysis in the output/ directory. Scripts ran on a server (with a large amount of RAM) are found in CODE_Server/, and code used to analyze the results and create plots can be found in

 doi:10.20944/preprints202203.0206.v1



CODE_Analysis/. There is also a CODE_Reference/ directory which contains a couple of files pointing out useful tips for working within the framework that we did.

For further information, see the .README file in the github repo.

## Appendix C. Computing Considerations

The train data for each of the three data sets had 12 million rows and 42 predictors. In order to train these models, we used the BYU statistics department servers. These machines had enough RAM to deal with the data in its raw format, and effectively model it. In our experience, in order to model this data in the way that we did, H2O needs to have at least 50GB of memory allocated to the cluster.

After the initial hurdle of figuring out where and how we could model the data, we began the process of running the model tuning RScripts on the servers. The severity models ran fine (they use less data, see Section 3.2.2) and without too much difficulty, but tuning the frequency models was more challenging. Aside from running slower, the Rscript would often quit in the middle of the process for seemingly no reason. Many long hours were spent babysitting these models and ensuring that they ran properly. Ultimately, we discovered that part of the problem was the H2O package being updated in R without the core H2O cluster being updated on the servers. Once we realized this and fixed it, the frequency models ran much smoother, although there were still some script failures. Overall, H2O is very useful for modeling large data, but can be a little bit unstable when running for long periods of time.

Aside from the lessons learned about H2O, we discovered many ways to save memory and time when modeling. In each loop we wanted to save the predictions of the model on the test set so that ultimately we could compute tuning metrics for the overall two-part model. However, writing this data every loop was expensive, and we instead landed on a solution that allows the user to input the save frequency, or how often the tuning results and the predictions on the test data set are saved.

Another lesson learned was from the model combination where we computed the metrics on the test set for the two-part model. Due to the large number of combinations of models, we needed to ensure we were using as many cores as possible on the servers. To do this, we used the futures package from the tidyverse in R that allows for multiple computations to be run simultaneously (Vaughan and Dancho 2021). In this framework, however, the data is copied to each core individually, meaning that we had to be careful about how many cores we set the process to work on, as a single instance can use up to 30GB of RAM. This process could be made more efficient by storing the predictions for each model in a separate file, instead of one large file like we did. That would allow the user to simply pass the two names of the files to the parallel mapping, which requires almost no duplicated memory allotment.

## Appendix D. Additional Plots

Due to the exploratory nature of the paper, many plots were created that are not necessary to include in the body of the paper. This section contains those plots, along with descriptions of each one.

### Appendix D.0.1. COLL Final Model Predictions

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 March 2022 doi:10.20944/preprints202203.0206.v1



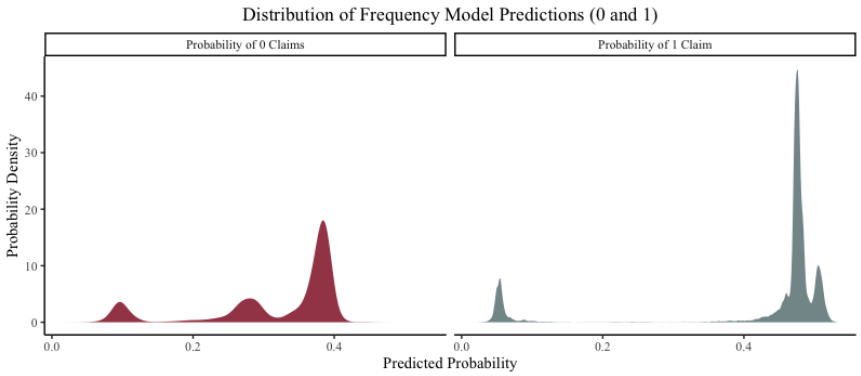


**Figure A1.** The frequency model's predicted probability of the final COLL model (only for the zero and one claim classes)

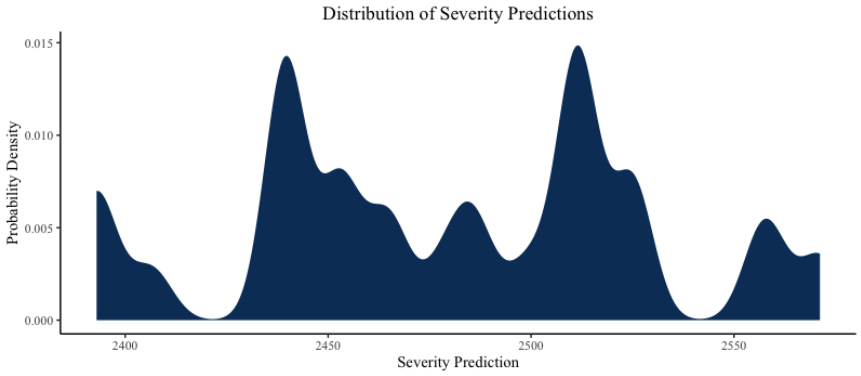


**Figure A2.** The severity predictions of the final COLL model

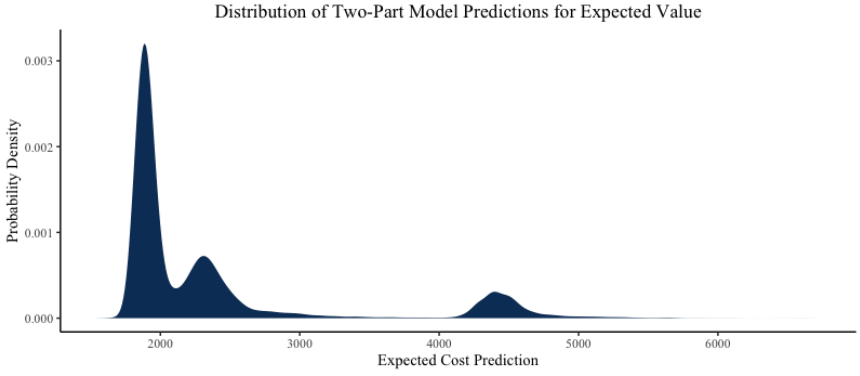


**Figure A3.** The expected value predictions of the final COLL two-part model

 doi:10.20944/preprints202203.0206.v1



## References

Akinyemi, Kemi and Ben Leiser. 2020. The use of advanced predictive analytics for rate making in insurance.

H2O.ai. 2021, January. *h2o R Package*. Version 3.34.0.1.

Kuhn, Max. 2008. Building predictive models in r using the caret package. *Journal of Statistical Software, Articles 28*(5), 1–26. doi:10.18637/jss.v028.i05.

Lundberg, Scott M, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. 2020. From local explanations to global understanding with explainable ai for trees. *Nature machine intelligence 2*(1), 2522–5839.

Lundberg, Scott M and Su-In Lee. 2017. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems 30*, pp. 4765–4774. Curran Associates, Inc.

Matthews, Spencer and Brian Hartman. 2021. mshap: Shap values for two-part models. *Risks 10*(1), 3.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12*, 2825–2830.

Vaughan, Davis and Matt Dancho. 2021. *furrr: Apply Mapping Functions in Parallel using Futures*. R package version 0.2.2.

Wickham, Hadley, Romain François, Lionel Henry, and Kirill Müller. 2018. *dplyr: A Grammar of Data Manipulation*. R package version 0.7.6.