# Actuarial Analysis of Worker's Compensation Claims

Brien Pike
922092730
bdpike@ucdavis.edu
STA160 – Practice in Statistical Data Science
*University of California, Davis*

**Abstract**
This project implements the use of Natural Language Processing (NLP) techniques and ensemble learning models to predict workers' compensation claim costs from structured and unstructured data. A dataset of 54,000 claims was preprocessed through cleaning, inflation adjustment, and feature engineering, including transforming free-text injury descriptions by means of TF-IDF vectorization. Ordinary least squares was fit on the data to provide a baseline performance and then several ensemble based models were fit including: random forest, gradient boosting, XGBoost, and LightGBM. All models were trained and evaluated using 5 fold cross-validation. XGBoost achieved the best performance with LightGBM following closely behind. Both models generalized well on Kaggle's test data. Results demonstrate that simple NLP pipelines can effectively extract predictive value from unstructured text but future improvements will likely come from more advanced language models.

## Contents

**1 Introduction 1**
1.1 Literature Review . . . . . . . . 2

**2 Exploratory Data Analysis 2**
2.1 Data Cleaning . . . . . . . . . . 3
2.2 Feature Engineering . . . . . . . 4

**3 Methodology 4**
3.1 NLP . . . . . . . . . . . . . . . . 5
3.2 Regression . . . . . . . . . . . . . 6
3.2.1 Random Forest . . . . . . 7
3.2.2 Gradient Boosting Machine 7

**4 Results 9**
4.1 Kaggle Competition . . . . . . . 10

**5 Discussion and Reflection 11**

**6 Conclusion 12**

**7 References 13**

**8 Appendix: External Resources 14**

## 1 Introduction

Actuarial science plays a crucial role in the insurance industry where it is used to forecast risks, set premiums, and ensure financial solvency. Actuaries implement statistical models to analyze historical data and make informed predictions about future events. In the property and casualty sector, one of the most complex areas actuaries deal with is workers' compensation insurance. This form of coverage is mandated in many jurisdictions to provide wage replacement and medical benefits to employees injured on the job, and it presents unique challenges due to the legal, medical, and occupational components of each claim.

Workers' compensation data is especially heterogeneous and frequently includes semi-structured or unstructured data fields, such as free-text descriptions of incidents. These fields often contain critical context and nuance that are lost when relying solely on structured variables. As a result, using advanced techniques such as Natural Language Processing (NLP) to extract actionable features from these texts is critical to actuarial modeling and insurance analytics. This paper attempts to explore how NLP can be used to transform free text claims data into regressible numeric features for use in ensemble based models to predict dollar amounts for workers compensation claims.

### 1.1 Literature Review

The integration of NLP into insurance analytics has gained increasing importance as insurers seek to extract actionable insights from unstructured textual data. In the context of workers' compensation claims, which often include narrative-based descriptions of accidents, traditional statistical models fall short in capturing the full information value embedded in free-text fields. Recent research has demonstrated the efficacy of NLP in automating the classification and summarization of such texts. A comprehensive survey by Kolambe and Kaur [1] highlights core challenges in this domain, including ambiguity in language, the prevalence of domain-specific terminology, and inconsistencies in how injuries and circumstances are described. They argue that while rule-based systems offer transparency, machine learning approaches—especially those utilizing Named Entity Recognition and text vectorization—yield more scalable and accurate results when applied to claim narratives.

In parallel, efforts to apply topic modeling and hybrid NLP pipelines have shown promise in structuring injury narratives for predictive purposes. For instance, Nanda [3] explored a dual approach combining rule-based logic with supervised machine learning to autocode injury descriptions which resulted in improved classification performance. Similarly, research on topic modeling of occupational injury discourse by Min et al. [2] suggests that unsupervised models like Latent Dirichlet Allocation can reveal latent structures in textual claims data thus supporting better risk segmentation and triage decisions. These studies collectively suggest that NLP can enhance the predictive modeling of claims severity and cost by surfacing features otherwise obscured in raw narrative form.

## 2 Exploratory Data Analysis

**Data preparation, preprocessing and visualization**

This analysis is being performed on the Actuarial Loss Prediction dataset from a Kaggle competition. The dataset has 54,000 observations of 14 variables, 13 predictors and 1 numeric target. `DateTimeOfAccident` tells when the accident occured and `DateReported` tells when the accident was actually reported. The data also contains the age, gender, and marital status of the claimant along with whether or not they have children or any other dependents that rely on them for care. There is also employment information that includes weekly wages, hours worked per week, days worked per week, and whether the claimant is classified as full time or part time. Finally there is a free text column `ClaimDescription` that contains information about the accident, type of injury, and where on the body the injury happened, plus columns for initial claim cost estimation by the insurer and the target variable `UltimateIncurredClaimCost`.

### 2.1 Data Cleaning

Close inspection of the data revealed that it was almost ready for regression tasks but there were some areas that needed to be cleaned up. `HoursWorkedPerWeek` had a max of 640, which is an impossible value. Closer analysis showed 463 rows with hours worked per week greater than 84 or hours worked per day greater than 12. In order to get these values more in-line with reality there appeared to be three good options. One, would be to simply take the number of days worked per week and multiply it times 12 hours per day and set that as that rows hours per week. This takes impossible values and smashes them down to the maximum plausible hours. Two, would be to take the average hours worked per day (excluding the outliers) and multiply that number by days worked and that would make these outliers fall in line with the rest of the data. Three, of course, would be to delete the rows with this data. The possible problem with path 3 is that if the hours worked per week or per day is only weakly correlated with the target then we could potentially be throwing out other valuable information. The other two methods have downsides as well of course. A correlation heatmap shows almost zero correlation across the board between hours worked per week and any other variable. I believe then that simply using the average of other values would allow me to keep as much data in the dataset as possible without altering the predictive accuracy.

The only variable that contained any null values was `MaritalStatus`. Marital status had 3 potential values: M for married, S for single, and U for unknown. The 29 null values in this column were simply imputed as U. One potentially large issue with the data is the time frame over which the data was collected. The earliest entry for claim date is 1988 and the latest entry is 2005. This 17 year spread in the reporting could have a large impact on regression since dollars cannot be directly compared over such a large span of time. To address this issue I used data from the The Federal Reserve Bank of St. Louis to create a small csv file with consumer price inflation (CPI) data so that all dollar amounts could be adjusted to a standardized time.

Finally there were potential outliers that needed to be dealt with. The only columns of potential concern here were `WeeklyWages`, `InitialIncurredCalimsCost`[*], and the target `UltimateIncurredClaimCost`. The boxplot for `WeeklyWages` showed a long right tail and a couple of potential outliers. However, these maximum values were approximately 18 times the mean value for this column. Because wages tend to increase significantly for upper management in large corporations I chose not to remove these values. I suspect that employee wages will not significantly impact the final claim value and I did not want to remove rows simply because it was the CEO that got injured and not a blue collar worker. The initial and final claims amounts had such extended right tails that the boxplots would need a log scale to visualize them properly. This is mostly expected as medical costs can be extremely high for some injuries

[*]Note that claims here is misspelled in the dataset, and this is not an error on my part.

or procedures, however there were two values that were so far outside of the range of the rest of the data I felt they had to be dealt with. Both of these values were two orders of magnitude higher than the mean and I felt they would negatively impact modeling accuracy. Thus only those 2 rows of data were removed, which still left a long right tail but the boxplots looked a lot better.

### 2.2 Feature Engineering

From the initial inspection I felt that there was an opportunity to engineer a couple of new features from the data that may impact model performance. The dataset has two date columns `DateTimeOfAccident` and `DateReported` that should not have any impact on claims cost [†]. While it is unreasonable to expect that accidents that happen in May would be more costly than accidents that happen in July, I wondered if the delta between the date that the incident occured and when it was reported may have some value to the model. For instance if an injury is very severe it will almost certainly be reported immediately, however some more minor accidents may not bother the employee enough initially for them to bother making an immediate claim. I thus created `ReportingDelta` as simply `DateReported - DateTimeOfAccident`. Additionally, the data has hours per week, and days per week. For workers compensation claims it may be interesting to look also at hours worked per day when predicting the target. People get tired over long periods of time and those working greater than a certain threshold could be more susceptible to bad injuries. So I've engineered another feature as `HoursWorkedPerWeek\DaysWorkedPerWeek`. The only potential correlation issues between variables were between `DaysWorkedPerWeek` and `HoursWorkedPerWeek`, and between `HoursWorkedPerWeek` and `HoursWorkedPerDay` (Fig. 1). This correlation was to be expected and is not high enough to be concerning. The highest correlated feature with the target was the initial claims cost.

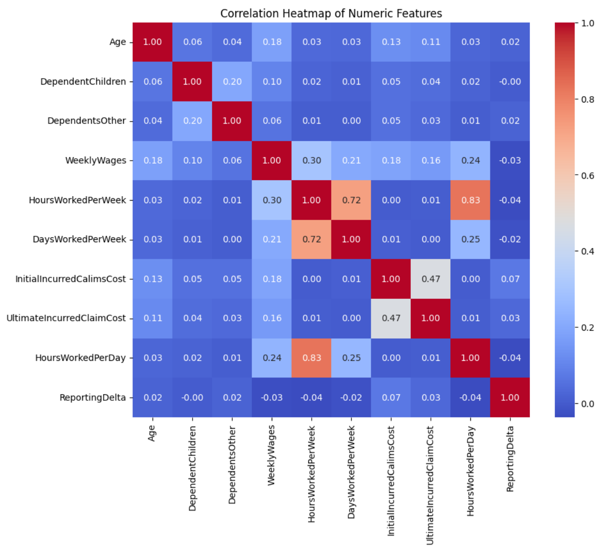

Figure 1: Correlation Heatmap

## 3 Methodology

**NLP feature extraction, parameter tuning and regression**

[†]After accounting for inflation.

The numeric and categorical variables have all been cleaned and some new features extracted, however the data is not ready for modeling just yet. There is one very important part of the process that still needs attention. The feature engineering of claims information is the most challenging part of this analysis. `ClaimDescription` contains free text descriptions of the incident with text such as the following: 'STEPPED AROUND CRATES AND TRUCK TRAY FRACTURE LEFT FOREARM', 'REACHING ABOVE SHOULDER LEVEL ACUTE MUSCLE STRAIN LEFT SIDE OF STOMACH', 'CUTTING DOWN A SMALL STEP SPRAINED TWISTED RIGHT FOOT', and 'STRIKING AGAINST FORKLIFT PRONGS BRUISED RIGHT RING FINGER'.

These descriptions follow the rough format of method of injury, location of injury, and injury type. One issue however is that not all 3 elements of this information are included in every description and the descriptions are not laid out in a format that makes extraction of this information easy. One potential method to engineer new columns from this information would be to manually read through the claims information and try to create categories such as lifting, stepping, or struck. But there are 54,000 observations making this task extremely inefficient and unlike to generalize well to unseen data. A second method might be to manually tag 500 to 1000 rows and then use machine learning methods based on that tagging to fit a model on the claims description and engineer the features that way. This method will still require significant amounts of time and is prone to the same error as the prior method. Instead I used a term frequency - inverse document frequency vectorizer (TF-IDF).

## 3.1 NLP

Before any vectorization of the text data can be performed it must first be standardized. All text is converted to lowercase and any punctuation and extra spaces are removed. Next all words are passed through a spelling check using the TextBlob package in python, before applying lemmatization using SpaCy. Lemmatization reduces words to their root form using NLP processing that uses context to determine the correct form. Some manual changes had to be implemented however as it would sometimes change 'left' to 'leave' and would not always change 'lower' to 'low'.

After the free text information was standardized for case, punctuation, and spelling TF-IDF was used to create a sparse feature matrix from the text descriptions so that regression models could be fit on the data. The TF-IDF score is calucluated by multiplying TF $\times$ IDF , where the term frequency is calculated as

$$\frac{\text{number of time the term appears}}{\text{total terms in the document}}$$

and the inverse document frequency is calculated as

$$\log\left(\frac{\text{documents in the corpus}}{\text{documents in the corpus with term}}\right)$$

and with only 1 document in this corpus it reduces down to just term frequency. This is primarily being used to generate a numeric representation of the various terms found in the data so that a regression task could be performed. Rather than attempting to classify the free text into columns that make sense to a person they are instead weighted by how often unigrams (single terms) and bigrams (two words grouped together) appear in the data. This has the advantage of being extremely fast and easy to regress on, with the disadvantage that words that appear an equal number of times are treated as the same feature.

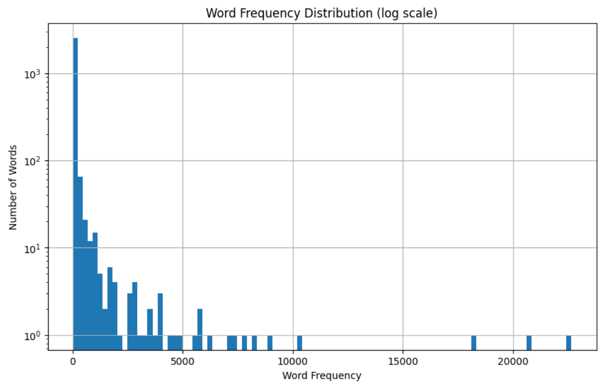

Figure 2: Log Word Frequency

Even though the TF-IDF vectorizer creates a sparse matrix of data, it still has a large impact on modeling performance. Visualizing the distribution of word frequency helped give a starting point on the number of terms to include in the matrix (Fig. 2). It can be seen from the figure that the majority of the terms appear fewer than 5000 times. However, fitting models with a 53998 × 5000 TF-IDF matrix was far too slow on my hardware. After some trial and error an arbitrary value of 587 was chosen as it provided reasonable modeling performance and captured all terms with a frequency greater than 25. To visualize the effect of TF-IDF vectorization on feature engineering, a weighted word cloud was created (Fig. 3).

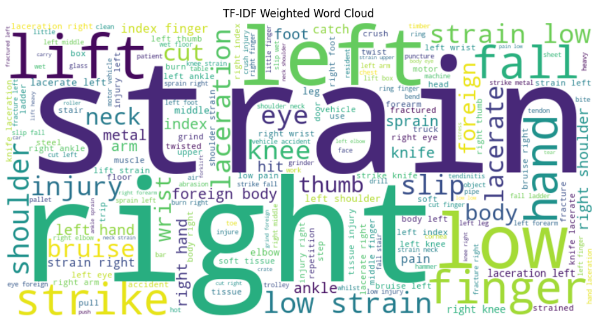

Figure 3: Word Cloud of Vectorized Data

## 3.2 Regression

To fit models on the data for regression, every model followed the same basic process. All data was cleaned, preprocessed, and new features engineered according to the details in the EDA and NLP sections of this report. Then all numeric variables were standardized and all categorical features were 1 hot encoded. All models were fit using 5 fold cross-validation to reduce overfitting and enhance generalizability, and then root mean square error (RMSE), $R^2$, and mean absolute error (MAE) were computed to compare performance across models.

Several model types were fit starting with ordinary least squares (OLS) as a baseline to compare performance (Table 1).

| **OLS** | **RMSE** | **$R^2$** | **MAE** |
|---|---|---|---|
| Train | 46229 | 0.258 | 15847 |
| Validation | 46798 | 0.239 | 16122 |

Table 1: OLS regression performance metrics.

It was not expected that OLS would perform well at all on this dataset, on the contrary it was expected to be the worst performing model but it should serve as a check on the performance of other models. In addition to OLS, several tree based models were fit on the data.

### 3.2.1 Random Forest

The first tree based model fit on the data was random forest.

A random forest is an ensemble learning technique that builds multiple decision trees during training and outputs the average prediction of the individual trees. Each tree is trained on a random subset of the data with bagging and at every split in a tree, a random subset of features is considered. This randomness ensures that the trees are de-correlated which reduces the overall model variance and guards against overfitting. This is important for this type of dataset with a spare TF-IDF feature matrix as it helps to ensure the model is not fitting the noise and will generalize well to unseen data. Prior to modeling a grid search was implemented to find optimized hyper-parameters (Table 2).

| **Hyperparameter** | **Value** |
|---|---|
| Maximum Depth | 10 |
| Minimum Samples per Leaf | 2 |
| Minimum Samples per Split | 5 |
| Number of Trees | 100 |

Table 2: Optimized hyperparameters for the Random Forest model.

Here maximum depth limits how deep each tree is allowed to grow, preventing overfitting by controlling complexity. Minimum samples per leaf and minimum samples per split ensure that each node in the trees has enough data to justify a split or to form a leaf which reduces noise and overfitting. The last hyperparameter, the number of trees, limits the overall ensemble size. This helps improve prediction stability by averaging multiple decision trees. The model was then fit on these hyperparameters and the results tabulated (Table 3).

| **RF** | **RMSE** | **$R^2$** | **MAE** |
|---|---|---|---|
| Train | 40153 | 0.440 | 13498 |
| Validation | 46157 | 0.260 | 14425 |

Table 3: Random Forest regression performance metrics.

### 3.2.2 Gradient Boosting Machine

After random forest was fit the next step was implementing a gradient boosting machine (GBM). Unlike random forest, which constructs trees in parallel and averages their predictions, GBM builds trees sequentially with each new tree attempting to correct the errors of the previous ensemble. This boosting approach allows for more nuanced refinements in prediction accuracy. This makes it particularly effective when working with complex data and sparse feature matrices like those produced by TF-IDF vectorization. A grid search was similarly performed to identify the optimal hyperparameters for GBM (Table 4).

| **Hyperparameter** | **Value** |
|---|---|
| Learning Rate | 0.05 |
| Maximum Depth | 3 |
| Minimum Samples per Leaf | 2 |
| Minimum Samples per Split | 5 |
| Number of Trees | 100 |

Table 4: Optimized hyperparameters for the Gradient Boosting Machine.

For GBM the learning rate controls the contribution of each new tree added sequentially. This ensures that the model adjusts gradually and avoids overfitting. The maximum depth limits the complexity of each boosting tree, while the minimum samples per leaf and per split ensure that each node in the tree is statistically robust before performing a split which helps to reduce noise. Finally, the number of trees specifies the total boosting rounds and allows the model to progressively correct errors from earlier iterations. The model was then fitted with these optimized hyperparameters, and the results tabulated (Table 5).

| **GBM** | **RMSE** | $\mathbf{R}^2$ | **MAE** |
|---|---|---|---|
| Train | 43495 | 0.343 | 14001 |
| Validation | 45617 | 0.277 | 14331 |

Table 5: Gradient Boosting Machine regression performance metrics.

Of the models fitted to this point, GBM showed the most promising results. I wanted to next attempt to improve performance by applying regularization. Regularization is a method that discourages overly complex models by adding a penalty to the loss function. This penalty forces the model to favor smaller weights and simpler structures and leads to a reduction in overfitting. Applying regularization meant finding optimum values for $\alpha$ and $\lambda$ but this was challenging as grid search was extremely slow and this would be impractical on the scikit learn implementation of GBM. So Instead I next fit models with both XGBoost, and LightGBM. Both of these are advanced implementations of gradient boosting that offer significant improvements in speed and efficiency compared to standard GBM models.

XGBoost employs second-order gradients that are optimized memory usage, along with parallel tree construction, to accelerate training and reduce overfitting. LightGBM leverages a leafwise tree growth strategy along with other techniques, such as gradient-based one-sided sampling and exclusive feature bundling to decrease computation time and memory consumption.

First I applied grid search with regularization to find optimized hyperparameters for XGBoost (Table 6).

| **Hyperparameter** | **Value** |
|---|---|
| Column Sample per Tree | 0.8 |
| Learning Rate | 0.05 |
| Maximum Depth | 3 |
| Minimum Child Weight | 5 |
| Number of Trees | 200 |
| L1 Regularization ($\alpha$) | 1 |
| L2 Regularization ($\lambda$) | 10 |
| Subsample Ratio | 1.0 |

Table 6: Optimized hyperparameters for XGBoost with regularization.

For XGBoost the learning rate controls how much each new tree contributes to the overall model. This ensures the model learns in small, steady steps. The maximum depth constrains the complexity of each tree. Minimum child weight demands enough training instances before another split is made. The sampling parameters, column sample per tree and subsample ratio, introduce randomization to help reduce overfitting. L1 and L2 regularization, represented by $\alpha$ and $\lambda$ respectively, restrict large weight values. Lastly, the number of trees sets the total boosting rounds and permits the model to iteratively refine its predictions. The model was then trained using these optimized hyperparameters and the performance metrics tabulated (Table 7).

| **XGBoost** | **RMSE** | $\mathbf{R}^2$ | **MAE** |
|---|---|---|---|
| Train | 44132 | 0.323 | 13949 |
| Validation | 45376 | 0.285 | 14198 |

Table 7: XGBoost regression performance metrics.

For LightGBM, a Bayesian optimization approach was implemented using the Optuna package. This method begins with an initial exploration of the hyperparameter space and then uses the results of early trials to inform subsequent evaluations. It efficiently refines the search process and quickly converges on optimal settings while reducing computational cost compared to traditional grid search techniques. This wound up being the most efficient hyperparameter search process used in this analysis. The optimized hyperparameters with regularization were collected and tabulated (Table 8).

| **Hyperparameter** | **Value** |
|---|---|
| Number of Trees | 200 |
| Learning Rate | 0.035 |
| Maximum Depth | 3 |
| Number of Leaves | 36 |
| Minimum Child Samples | 10 |
| L1 Regularization ($\alpha$) | 0.518 |
| L2 Regularization ($\lambda$) | $9.5 \times 10^{-5}$ |
| Minimum Split Gain | 0.021 |

Table 8: Optimized hyperparameters for the LightGBM model.

For LightGBM the learning rate governs the contribution of each boosting round and ensures that the model updates its predictions in small, stable increments. Maximum depth limits how deep each tree can grow and the number of leaves increases the model's capacity to capture complex interactions while guarding against overfitting. The minimum child samples parameter makes sure that nodes are only split when there is sufficient data available. L1 and L2 regularization ($\alpha$ and $\lambda$) restrict large weight values and encourage simpler and more generalizable models. Minimum split gain prevents splits unless they bring a meaningful improvement in error reduction and the number of trees sets the total boosting rounds. These hyperparameters were optimized using a Bayesian approach with the Optuna package and then integrated into the final LightGBM model. The model was then trained using these optimized hyperparameters and the performance metrics tabulated (Table 9).

| **LightGBM** | **RMSE** | **R$^2$** | **MAE** |
|---|---|---|---|
| Train | 44273 | 0.319 | 13993 |
| Validation | 45415 | 0.283 | 14215 |

Table 9: LightGBM regression performance metrics.

## 4 Results

### Summary of model fit data

After fitting was complete all of the model performance metrics for every model fit on this data were combined and tabulated for comparison (Table 10). The results of the model fitting process mostly fell in line with what was expected from this dataset. The OLS model had the worst performance of all models but surprisingly showed minimal signs of overfitting. The RMSE showed a delta of only \$569 between the training and the validation sets and the MAE and $R^2$ also had surprisingly small deltas.

Random forest was the worst of the ensemble learning techniques used on this data, and surprisingly exhibited the worst overfitting of all models (even OLS). The RMSE had a delta of \$6004 between the training and validation sets which was approximately triple that of the next worst model. Despite overfitting the data, the RF model had validation RMSE, $R^2$, and MAE all better than OLS. While RMSE was fairly close, MAE was not. This suggest that the RF model would generalize much better to unseen data than the OLS model and this was completely in-line with expectations on data that has complex nonlinear relationships.

The gradient boosting machine represented a significant increase in performance from the prior two models. It showed much less overfitting when comparing the training and validation data than RF exhibited and it had the best RMSE, $R^2$, AND MAE of all models tested to this point. The ability for a GBM to build trees sequentially and learn from the residual errors of prior fits appears to be particularly well-suited for this data.

The biggest problem with GBM was that a grid search of the hyperparameter space was extremely computationally expensive and took too much time to fully optimize and additionally I want to implement regularization to further improve model performance. It was to this end that XGBoost and LightGBM were used to fit GBMs on the data. The addition on regularization and the utilization of more efficient modeling algorithms produced the two best models. Both XGBoost and LightGBM exhibited extremely small deltas between test and validation sets for both RMSE and MAE. These two models also produced the highest $R^2$ values and the lowest errors of all models. The XGBoost model was the winner here as it edged out LightGBM in every category, albeit only slightly.

| **Model** | **Train RMSE** | **Val RMSE** | **Train R$^2$** | **Val R$^2$** | **Train MAE** | **Val MAE** |
|---|---|---|---|---|---|---|
| OLS | 46,229 | 46,798 | 0.258 | 0.239 | 15,847 | 16,122 |
| RF | 40,153 | 46,157 | 0.440 | 0.260 | 13,498 | 14,425 |
| GBM | 43,495 | 45,617 | 0.343 | 0.277 | 14,001 | 14,331 |
| XGBoost | 44,132 | 45,376 | 0.323 | 0.285 | 13,949 | 14,198 |
| LightGBM | 44,273 | 45,415 | 0.319 | 0.283 | 13,993 | 14,215 |

Table 10: Comparison of performance metrics across models. All values are rounded for clarity.

## 4.1 Kaggle Competition

In order to test how well the top two models generalize on unseen data, the models were fit on the Kaggle test dataset and uploaded as a late submission to the competition. The test data was processed according to the same preprocessing and feature engineering pipeline as the training data was including adjusting for inflation. The data was then fit using the saved vectorizer from the TF-IDF process, the saved scaler and one hot encoder from the processing pipeline and the saved model from the fitting process to prevent data leakage. Finally the inflation compensation was undone and the results were uploaded to Kaggle for scoring (Table 11).

| **Model** | **Pub RMSE** | **Priv RMSE** |
|---|---|---|
| XGBoost | 23595.98 | 30076.35 |
| LightGBM | 23627.65 | 30093.07 |

Table 11: Performance of XGBoost and LightGBM on Public and Private scores.

The scores between the two models was extremly close with XGBoost just barely edging out the performance of the LightGBM model[‡]. This mimicked the behavior on the validation data. The performance of the XGBoost model

[‡]The RMSE scores here are lower than for the previously reported models as the inflation adjustment was undone.

was within 1.03% of the private leader, and 1.18% of the public leader. I cannot upload the URL for my submissions as the competition is closed and nothing will show on the leaderboard. However both csv files are saved in the github repository linked in the document appendix so that these results can be easily verified.

## 5 Discussion and Reflection

**Takeaways from project analysis**

The final result of this project, in terms of model accuracy was fantastic, but there were a ton of challenges and a lot of room for improvement. This was my first attempt at implementing any kind of natural language processing and I was unable to try all of the methods that interested me. While this report presents the NLP portion of the methodology and straighforward the reality was that it was anything but straightforward. I struggled for quite a while in how to create regressible features from the claim description column and my final results represent a compromise between what I wanted to achieve and what became practical. I initially tried to create a rules based system that was created by manually tagging information in the dataset so that a machine learning model could be run on that to create a system in which the entire data could be autocoded. However this was impractical and the threshold for getting it to work properly was likely several thousand rows of data. While that represents a small fraction of the data it was a large amount of work for one person.

There was also the issue of a creating a system of rules in the first place by which to manually categorize and classify everything. I frequently found myself struggling on how to encode several of the text features. In the end by compromising on the TF-IDF vectorization I was able to regress on this text data. NLP processing gave me text that was standardized for spelling and lemmatization by context with only a few manual rules to take care of common issues. While this approach also has its own limitations, such as treating words that appear with identical frequency as the same feature this wound up not impeding model accuracy. One reason was that NLP was also used to create bigrams from the data that grouped up common two word combinations. This helped to ensure that terms like 'low back' and 'back strain' got accounted for in the vectorization and subsequent regression on the data. The processing of `ClaimDescription` into regressible features with quality modeling performance was by far the bulk of the work of this project for me. Once I could actually begin fitting models the project started to move along much more quickly. The biggest issue with modeling this data was simply due to the size of the dataset, especially when combined with the sparse feature matrix created by the TF-IDF vectorizer. Hundreds of predictors combined with tens of thousands of observations made finding ideal hyperparameters and the implementation of regularization quite challenging. Access to more compute power would help but fortunately highly efficient libraries like LightGBM exist to address this issue.

I thought that the biggest issue I would face with the TF-IDF vectorizer was generalizability, however as the Kaggle competition scoring showed, this was a non-issue. I still believe however that this approach has limitations that could be exposed by another larger test set. One last thing to note is the assumption of normality with the numeric features. Many of the features were nowhere close to normal. The majority of employees in this data had no dependents but several had as many as 9 dependents. This created a very long right tail. This same skewness was present in many other metrics as well, such as wages, initial claim cost, and ultimate claim cost. Unfortunately this is sort of like a real estate dataset where these long tails are expected. Fortunately standardization takes care of these issues and allows accurate regression on the data.

## 6 Conclusion

The goals of this project were to process free text data into regressible features that could be used to make accurate predictions that generalize well to unseen data. To that end the project was very succesful. I showed that TF-IDF vectorization of text features after basic NLP processing was a fast and efficient approach to tackling this sort of problem. The results on unseen test data were excellent as shown when comparing to Kaggle competition results. One downside to this approach however is interpretability. The vectorization process loses information about the text in the way it is implemented in this process as words with identical frequency are treated the same. This means that unpacking the features at the end isn't actually possible with a high degree of accuracy.

Another downside is that model performance increased very little from baseline with hyperparameter tuning and regularization, meaning there is not much room for performance increases without changing the way that text features are engineered. Future considerations for improving predictive accuracy and making insurance providers more profitable would be to use different models in an attempt to engineer these features. Implementation of bidirectional encoder representations from transformers (BERT) may increase model performance by providing a more context based implementation of text vectorization than TF-IDF with bigrams provided. Another possibility may be training of a large language model (LLM) to do the rules based encoding of text information that proved too challenging for me.

## 7 References

[1] Sapana Kolambe and Parminder Kaur. Survey on insurance claim analysis using natural language processing and machine learning. *International Journal of Recent Innovations in Computer Science and Communication*, 2023. URL https://www.researchgate.net/publication/382801528_Survey_on_Insurance_Claim_analysis_using_Natural_Language_Processing_and_Machine_Learning.

[2] Kyoung-Bok Min, Seung-Hyun Song, and Jin-Young Min. Topic modeling of social networking service data on occupational accidents in korea: latent dirichlet allocation analysis. *Journal of Medical Internet Research*, 22(8):e19222, 2020. doi: 10.2196/19222. URL https://www.jmir.org/2020/8/e19222/.

[3] Gaurav Nanda. *Improving the Autocoding of Injury Narratives Using a Combination of Machine Learning Methods and Natural Language Processing Techniques*. PhD thesis, ProQuest Dissertations Publishing, 2017. URL https://search.proquest.com/openview/335bc3c8d3e1f6b00fdac19a94d3e254/1?pq-origsite=gscholar&cbl=18750.

## 8 Appendix: External Resources

1. **GitHub Code Repository:** `https://github.com/brienpike/STA160_Final_Project`
2. **Kaggle Competition Dataset:** `https://www.kaggle.com/competitions/actuarial-loss-estimation`
3. **Federal Reserve CPI Data:** `https://fred.stlouisfed.org/series/CPIAUCSL`