Predicting Insurance Claim Severity Using Machine Learning

Karim Al Zeer Alhusaini
Nalini Kethineini

April 25, 2020

Abstract

It is important for any insurance company to predict the monetary loss associated with an insurance claim. Insurance companies always strive to implement the latest technology in statistical modeling to automate this process. Throughout this project, we used All State's insurance claim data and implemented multiple machine learning algorithms to predict the total dollar loss on an auto insurance claim. We aimed to reduce the mean absolute error associated with predicting the claim monetary amount. Given the size of the data, we observed the data, checked for any correlations, then implemented a one-hot encoding and label encoding on the categorical variables. Then, we implemented dimensionality reduction methods using Principal Component Analysis. We continued by implementing multiple regression machine learning models including a simple Linear Regression, Principal Component Regression, Lasso Regression, Ridge Regression, Support Vector Machines, Random Forest, and a Multi-Layer Perceptron. We compared the Mean Absolute Error (MAE) associated with each of the models and found that the Multi-Layer Perceptron produced the best prediction results on the testing data set. We concluded our research by proposing additional optimization methods and algorithms that can further help reduce the prediction error. This project relied on existing Python packages including NumPy, Pandas, Keras, and TensorFlow.

Contents

1 Introduction **3**

2 Exploratory Data Analysis **3**

2.1 Feature Distributions & Densities 4

2.2 Correlation Matrix 6

2.3 Feature One-Hot Encoding & Dummy Variables 7

2.4 Data Splitting 7

3 Dimensionality Reduction Using PCA **7**

4 Models & Results **8**

4.1 Multiple Linear Regression 9

4.1.1 Principal Component Regression 10

4.2 Regularization Methods 11

4.2.1 Ridge Regression 11

4.2.2 Lasso Regression 12

4.3 Support Vector Regressor 13

4.4 Random Forest 13

4.5 Multi-Layer Perceptron 14

5 Conclusion **15**

6 Appendix **17**

6.1 Appendix 1: Density Distribution 17

6.2 Appendix 2: PCA Visualization 18

1 Introduction

We pulled our data set from a past competition on Kaggle in which an insurance provider, Allstate US, challenged Data Scientists to use its open-source data to predict the severity (total monetary loss) on an auto insurance claim. The company provided different anonymized in-house and external attributes on **317,465** insurance claims. This is a list of existing insurance claims with the company. The provider, in this case, is looking to create a prediction model that forecasts and assigns a value of loss for each insurance claim instantaneously once a claim is processed with the company.

There was a wide range of data on the claims which totaled 132 features. The features include 1 ID index feature, 116 categorical features, 14 continuous features, and 1 target variable. The target variable is continuous and represents the total dollar monetary loss on the claim. We applied the different regression techniques we learned through this course and assess which method is best at predicting default. Furthermore, we applied two additional techniques outside of the course content that we thought can help improve our models.

Problem statement: The problem we addressed with this project was how to forecast the monetary loss of an insurance claim based on the claim characteristics in the dataset. As our target variable is continuous, this is a regression problem. We built several regression models in order to determine the model with the best predictive capability.

The criterion used to assess the models was the Mean Absolute Error (MAE). The mean absolute error represents the magnitude or absolute value of the error term, which the difference between the actual and the predicted values for each loss value. Our goal is to get the lowest possible MAE value out of the data set. The formula of the MAE is represented below:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| = \frac{1}{n}\sum_{i=1}^{n}|e_i|$$

The reason we used the Mean Absolute Error as our error metrics is that the Kaggle competition required using it. We used it to compare our models against the literature online. The lowest MAE value achieved in the Kaggle competition was 1110, which was calculated on the testing data set. We hoped to achieve a similar or better MAE result. The main tool used in this analysis was Python. We utilized different data science packages, including NumPy, Pandas, Seaborn, Matplotlib, TensorFlow, Keras, and Sci-Kit learn.

2 Exploratory Data Analysis

Before performing any modeling, we explored the data set to observe any correlations and treat any missing values that may exist. The provided data set was split into training and testing sets. The training set contained **188,318** instances while the testing data set contained **129,147** indicating the data was split using a 60:40 ratio. Nonetheless, in this project, we merged both

data sets and later created a different training, validation, and testing data sets.

Furthermore, the data contained 116 categorical features and 14 continuous features. The categorical features have the prefix "cat" while the continuous features have the prefix "cont". The modeling of the data was done on the training and validation data sets. The data required little cleaning and manipulation as it seems the data was already cleaned and presented in an easy to understand and handle format. The data contained no missing of Null values. The continuous variables were normalized on a range of 0 to 1 with a mean of 0.5. Unfortunately, given the proprietary nature of the data, all the features were anonymized and masked to hide the true meaning. This made it difficult to understand the nature of the data. Having named features would have made it easier to apply any domain knowledge to the data that would have helped with dimensionality reduction or model improvement and parameter tuning.

2.1 Feature Distributions & Densities

The first step in our Exploratory Data Analysis is to understand the different distributions of both the categorical and continuous features in the data set. Since there are 166 features in the data sets, it is difficult to visualize all at once. However, we plotted the histogram distribution of the target variable. **Figure 1** represents the histogram of the Loss target variable. As can be seen, the distribution of the loss variable is uneven with most of the loss variables under 5,000.

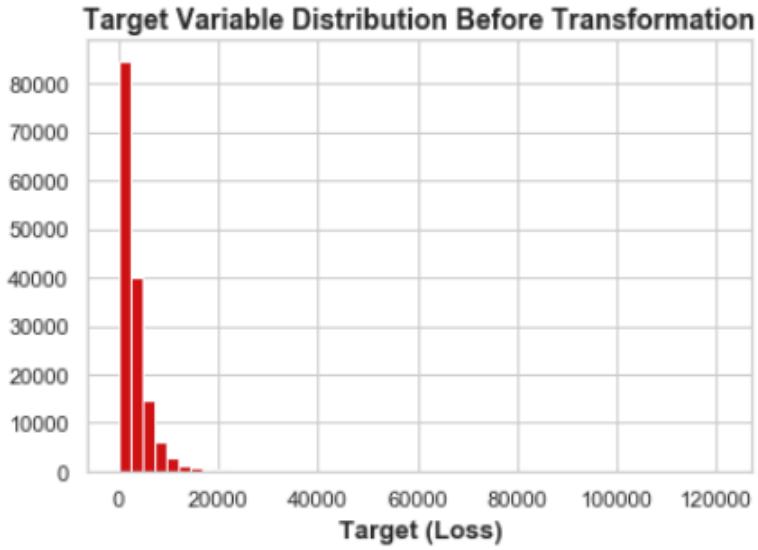

Figure 1: Target Variable Distribution

To mitigate this problem, we applied a log function to the target variable. This creates a more even distribution that is easier to visualize. **Figure 2** displays the distribution of the target variable after applying the log transformation. The distribution certainly looks more symmetric after applying the function.

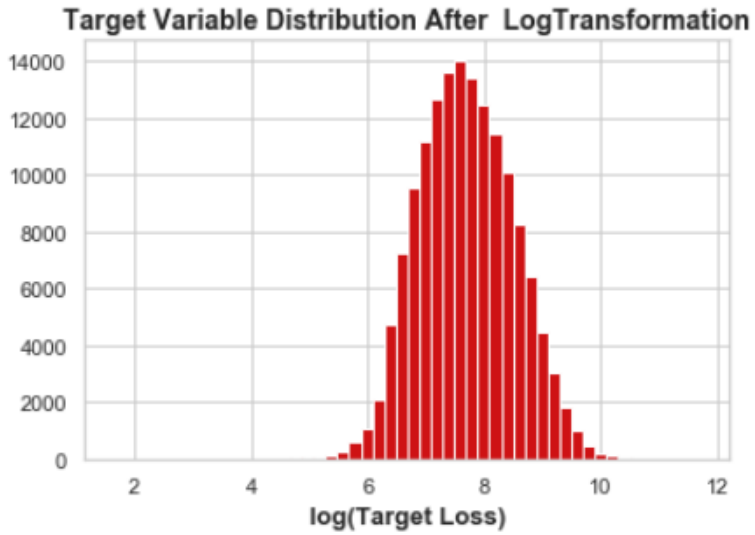

Figure 2: Target Variable Distribution After Log Transformation

In addition to the target variable distributions, we also plotted the distribution of the categorical variables. **Figure 3** shows the box-plot distribution of the first 4 categorical variables against the log transformation of the loss target variable. For the plotted categorical variables, there appear to be small differences between the two classes represented.

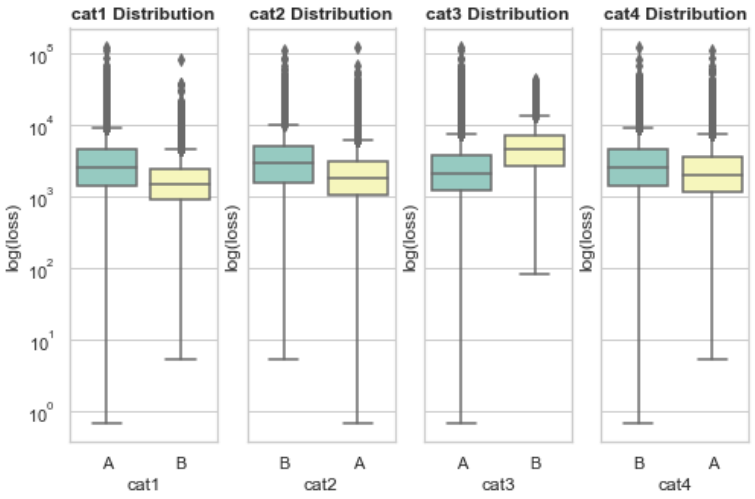

Figure 3: Boxplot of first four categorical variables

The final plot used to explore the data was the density plot of the continuous variables. **Figure 4** shows the density plot of the first four continuous variables. The plot shows the Kernel Density Estimate (KDE) of each of the variables. As can be seen, there seems to be some variation between the different continuous variables. **Appendix 1** shows the Kernel Density Distribution for all the continuous variables.

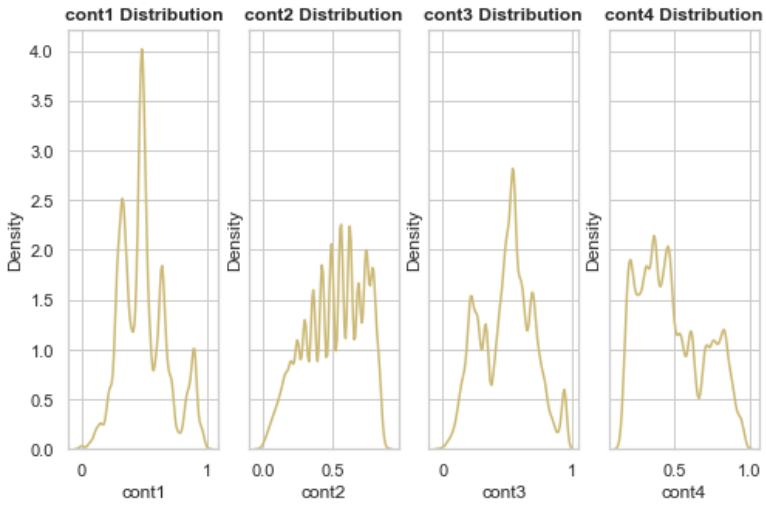

Figure 4: Density Distribution of first 4 continuous variables

2.2 Correlation Matrix

The second step in the Exploratory Data Analysis (EDA) process is to calculate and plot the correlation between the different features. The correlation metric used is the Pearson Correlation Coefficient which measures the linear correlation between two variables. Using the Python function to do so, we were able to plot the correlation matrix for the continuous features in the dataset. **Figure 5** displays the resulting correlation matrix.

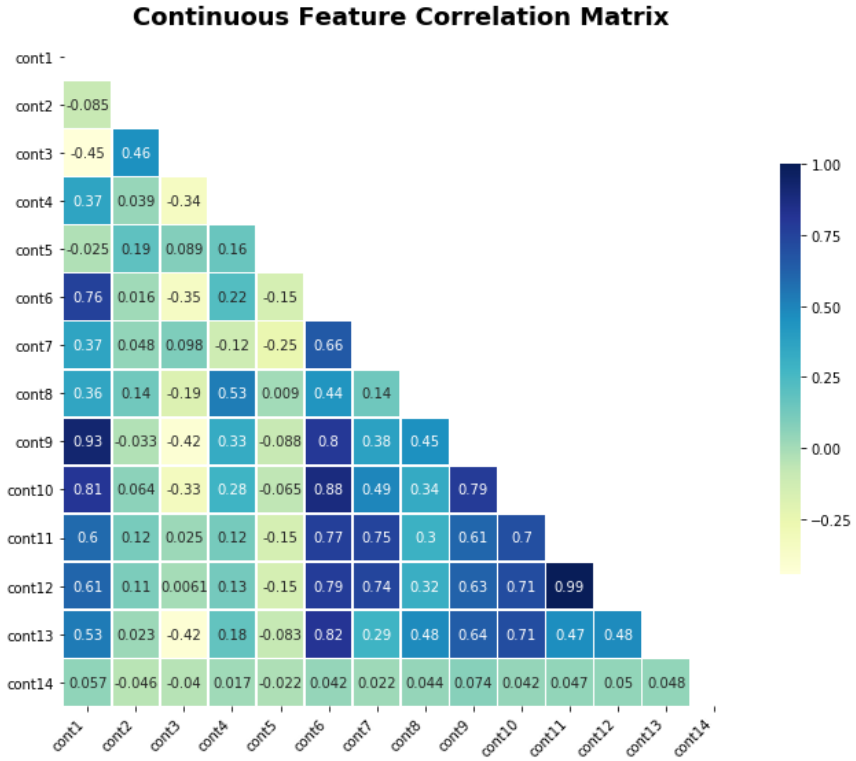

Figure 5: Correlation Matrix of Continuous Variables

The matrix shows a strong positive correlation between features *cont11* and *cont12*. As a result, the feature *cont12* was dropped from the data set. This is helpful since it reduces

the dimensionality and removes any unnecessary correlation in the regression model. There also seems to be a strong correlation between *cont9* and *cont1*, however, we decided against removing this feature since dropping it will not improve the regression models.

2.3 Feature One-Hot Encoding & Dummy Variables

Since this data set mostly consisted of categorical features, it was imperative to create dummy variables corresponding to each of the classes represented. In Python, this is done through Label Encoding and One-Hot Encoding (OHE). By implementing OHE, a binary vectorized variable is created for the features. While this method adds more dimensions, it makes it easier to create regression models with strong predictions. This method works particularly well with advanced methods, such as Support Vector Machines, and Multi-Layer Perceptrons (Neural Networks). One Hot Encoding is explained below.

$$var : [X, Y, Y, Z] == var\_X : [1, 0, 0, 0], var\_Y : [0, 1, 1, 0], var\_Z : [0, 0, 0, 1]$$

Nevertheless, One Hot Encoding does make the regression model more cumbersome as Dummy variables need to be handled delicately while building regression models. The final dataset with the dummy variables ended up containing 1190 features. Therefore, an alternate method is the label encoding method which essentially converts the text categories into numeric values. This method is explained below.

$$var : [X, Y, Y, Z] == var : [1, 2, 2, 3]$$

Both of the aforementioned methods were applied and used to build the different models of this project.

2.4 Data Splitting

As mentioned previously, the data provided by the insurance company was already split into training and testing sets. However, these two data sets were merged to create one large data set. This data set was later split into an 80% training set and a 20% testing set. The training set was further split into an 80% actual training set, and a 20% validation set. This means the training set comprised 64% of the entire set, the validation set comprised 16% of the entire set. The splitting was done randomly using Python.

3 Dimensionality Reduction Using PCA

It was mentioned in the previous section that the One Hot Encoding method creates many dummy variables that increase the dimensionality of the data set. In fact, the result of the OHE dataset contained 1190 features. One effective way to cure this problem is by implementing unsupervised learning techniques such as Principal Component Analysis (PCA). Principal Component Analysis uses the single value decomposition matrix of the features to create new

features that are linearly uncorrelated, known as Principal Components.

We implemented this method on the One-Hot Encoded dataset. Each one of the Principal Components (PC) explains a certain portion of the variance that exists in the original dataset. **Figure 6** shows the cumulative sum of the variance that is explained by adding more PCs.

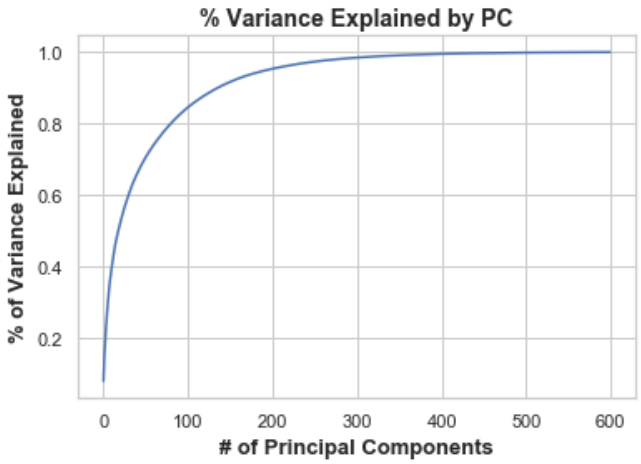

Figure 6: Variance Explained by Adding Principal Components

The Principal component analysis created 600 Principal Components out of the 1190 features that exist in the OHE data set. As can be seen, these 600 Principal Components explain up to 100% of the variance in the original dataset. In fact, using up to 400 principal components will explain 100% of the variance, which produces good dimensionality reduction results. These PCs will later be used to create a Principal Component Regression model.

Principal components can be difficult to visualize, however, by plotting two PCs at a time, we can get a better idea of what the Principal Components represent. **Appendix 2** shows the distribution of the log-transformed loss on the first two principal components.

In addition to PCA, there are other advanced embedding methods that can help reduced high dimensionality, such as *t-SNE* (t-distributed Stochastic Neighbor Embedding) and *UMAP* (Uniform Manifold Approximation and Projection). However, given the size of the data set, we ran into memory issues while executing the code on both these methods, and decided to exclude them from this project.

4 Models & Results

Once the data has been explored, and PCA was performed on the data set, we can explore the different machine learning algorithms that will be used in this project. Each of the models below was tuned using different hyper-parameter tuning methods. The models used in this project were:

1. Multiple Linear Regression

2. Ridge Regression
3. Lasso Regression
4. Support Vector Machines
5. Random Forest
6. Multi-Layer Perceptron

4.1 Multiple Linear Regression

As this is a regression problem, it is prudent to start with a Multiple Linear Regression model and assess its accuracy. Linear regression provides a simple supervised learning approach that predicts a quantitative response. The objective a of regression problem is to minimize the Mean Squared Error or the square distance between the actual and predicted values.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}_{(x_i)})^2$$

Running the model on the One-Hot Encoded data produced reasonable loss results on the training data set, but was ineffective on the validation data set, and produced abnormally high numbers. The reason could be due to the high number of dummy variables. Therefore, the regression model was run using the Multi-Label encoded data set. This produced reasonable loss results that were used to make predictions on the testing data set. **Figures 7 and 8** show the actual vs predicted loss value on the training and testing sets respectively.

Figure 7: Actual vs Predicted on Training Set

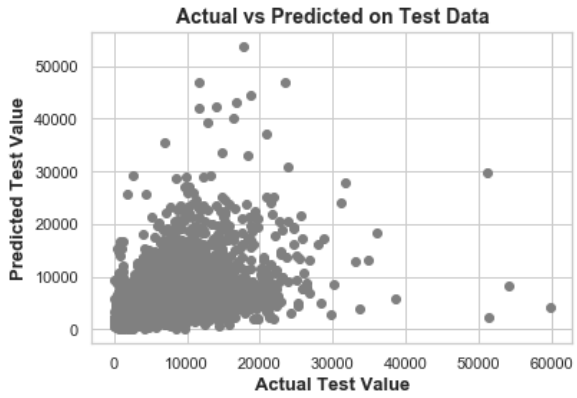

Figure 8: Actual vs Predicted on Testing Set

The final regression model using the label encoded data set produced a *1299* MAE on the training set, and *1312* on the Validation set.

4.1.1 Principal Component Regression

Given that the simple regression model did not perform well on the One-Hot Encoded data, we decided to use the Principal Components extracted from the Principal Component Analysis. This data set works well with the One-Hot Encoded dataset. Even though 400 Principal Components or more explain 100% of the variance, we decided to start with a small number of Principal Components and gradually increase the number to analyze how this impacts the training and validation regression models.

Figure 9 shows the training and validation error by using a different number of principal components. As can be seen, the 600 principal components seem like an ideal number that produces the lowest training and validation errors.

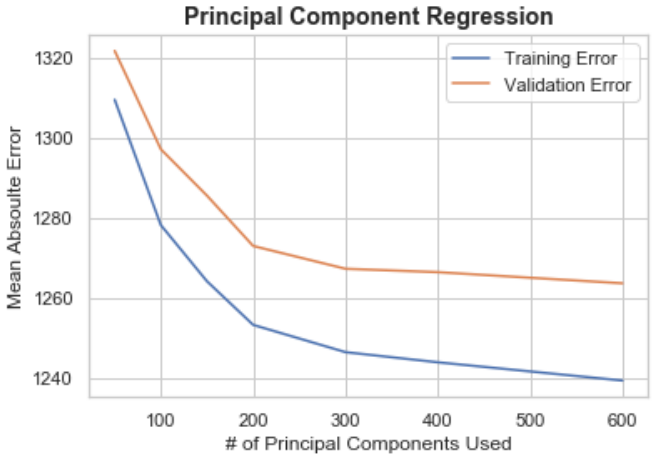

Figure 9: Training and Validation MAE by number of PCs

Using 600 as the number of Principal components to create a regression model produced better MAE results than the simple regression model. The final PC regression model using the One-Hot Encoded data set produced a *1240* MAE on the training set, and *1265* on the Validation set.

4.2 Regularization Methods

This dataset presents a problem of high dimensionality. When the number of dimensions increases relative to the number of instances in the data set, it eventually leads to a larger variance due to overfitting of the data. The Principal Component Analysis explored above is one method to treat high dimensionality by reducing the number of dimensions.

Regularization is another method that is aimed at tackling the problem of high dimensionality using variable selection. Some of the coefficients of the regression models are reduced to 0, mean some variables are constrained when used in the model. This reduces the variance and improves the accuracy of the model.

4.2.1 Ridge Regression

The first method of regularization that we explored is Ridge Regression also known as L1 Regularization. This method estimates the same aims to minimize the *Residual Sum of Squares* or *RSS* in addition to a new value. This is represented in the equation below.

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2 = RSS + \lambda\sum_{j=1}^{p}\beta_j^2$$

Where λ is a tuning parameter, p is the number of features. The regularization penalty is multiplied to the square of the parameters. When building a regularization model, the goal generally is to test different values for λ that produce the best loss results. Using the GridSearch and testing different values for λ. Using the above formula, a simple linear regression is achieved when $\lambda = 0$.

Using the GridSearch method, the best λ value was achieved for the Ridge Regression. **Figure 10** below displays the training and validation loss for different values of λ. The training and validation errors plotted have inverse positions, however, after an extensive search, an optimal λ as achieved. The resulting MAE was *1286.56* for the training data and *1302.32* for the validation data.

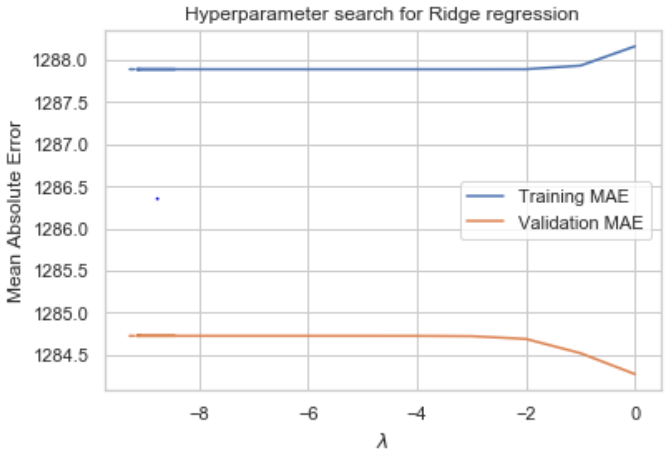

Figure 10: Training and Validation for different lambdas (Ridge)

4.2.2 Lasso Regression

Like Ridge Regression, Lasso Regression is another regularization method that penalizes models for high dimensionality. The difference is Lasso regression reduces the magnitude of the parameters rather than exclude them. The equation below represents the Lasso Regression.

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| = RSS + \lambda\sum_{j=1}^{p}|\beta_j|$$

We followed a similar step as with the Ridge Regression. We used the GridSearch method in Python to search for the optimal value for λ. **Figure 11** represents the training and validation losses achieved for different values for λ.

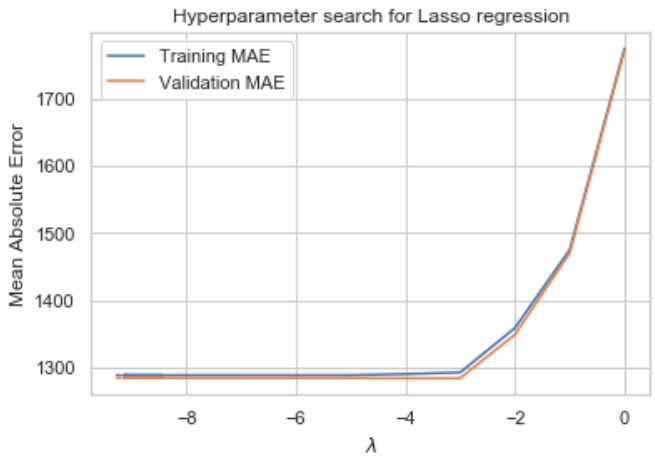

Figure 11: Training and Validation for different lambdas (Lasso)

Unlike Ridge Regression, lower values for λ appear to produce better results for Lasso Regression. The results for training and validation sets are very close. The optimal value of λ produced MAE value of *1286.35* for the training data and *1302.35* for the validation data. The best MAE values achieved in both Ridge and Lasso regressions were almost identical.

4.3 Support Vector Regressor

Support Vector Machines provide flexible yet powerful solutions to both classification and regression problems. In essence, Support Vector Machines try to find the maximum distance between two linearly separable classes. Support Vector Machines can also be applied to regression problems. The objective with Support Vector Regression is to try and find the hyperplane that minimizes the distance to the farthest data points or enclose all the points within the supporting hyperplanes.

Sci-Kit learn library in Python provides a Support Vector Regressor than can handle regression problems. Support Vector Machines can be very computationally expensive and memory intensive. Therefore, the training and validation sets were reduced in order to achieve the best and most efficient results. We used the One-Hot Encoded data set as it worked well with SVMs despite the high dimensionality.

For Support Vector Machines, there are multiple hyper-parameters that must be tuned to achieve the best results. One of these parameters is C which is the regularization parameter, which represents the L2 regularization penalty. **Figure 12** shows the training and validation MAE for different values of C regularization. Another parameter than can be tuned is the Kernel function used. By default, the Python library uses the RBF (Radial Basis Function). Other kernels include the Polynomial, Linear, and Sigmoid kernels. The kernel that produced the best results was the Linear kernel.

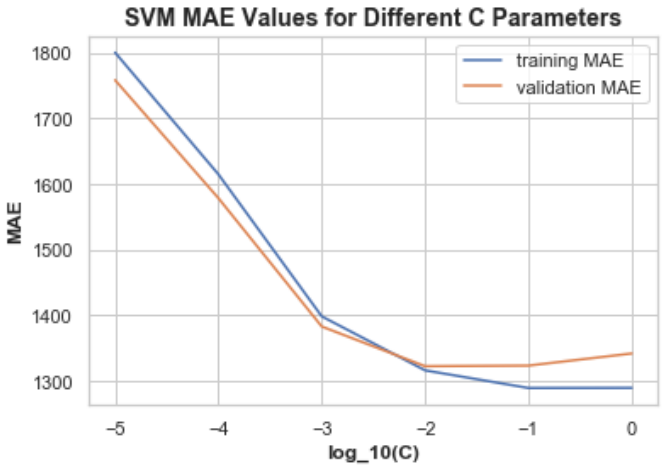

Figure 12: Training and Validation for different C-Regularization

The optimal hyper parameters that produced the best results were a Linear Kernel and C-Regularization value of 0.01. The optimal value of C produced MAE value of *1296.36* for the training data and *1308.77* for the validation data.

4.4 Random Forest

Once we ran the basic classification models, we decided to experiment with additional models that could be more suitable for such data set. The first of these models is the Random Forest.

A Random Forest creates an ensemble of decision trees on the training set, then produces the mode of the classes of all the decision trees. One benefit of using a Random Forest over a single decision tree is to avoid falling into the overfitting trap.

To find the best parameters for the Random Forest, there were a few parameters that needed to be tuned and optimized such as the depth of the forest (max_depth), the number of features(max_features), and the number of trees. Using Sci-Kit Learn GridSearch in Python, we were able to find the best parameter. Based on the GridSearch results, the best features resulted in a depth of 17, and features of 99.

After running the tree model with the above parameters, the result was a training MAE of *466.53* and the validation MAE was *1250.68*. One advantage of the Random Forest model is it has a high explainability and it produces a list of each of the features' importance. **Figure 13** shows the importance of each of the features. It appears some of the continuous features have high importance in the model.

4.5 Multi-Layer Perceptron

We decided to conclude by using a Multi-Layer Perceptron (MLP) or Neural Network model. Although this topic is outside the scope of this class, we have acquired a good knowledge of MLPs through IE7860. Multi-layer Perceptrons provide flexible yet powerful solutions to both classification and regression problems. However, getting the best results requires experimentation with different parameters and techniques. Therefore, we decided to build a standard model with built-in optimizer libraries to provide the best results possible.

For our model, we used a single hidden layer with 500 neurons. The activation function we used was *ReLU*. The model was built to train on 100 epochs, with a batch size of 128 for optimal performance. Additionally, we used the optimized *Adam* which provides a powerful gradient optimization. The model was asked to optimize the MAE. **Figure 13** shows the results of the model training on the training and validation sets for 100 epochs.

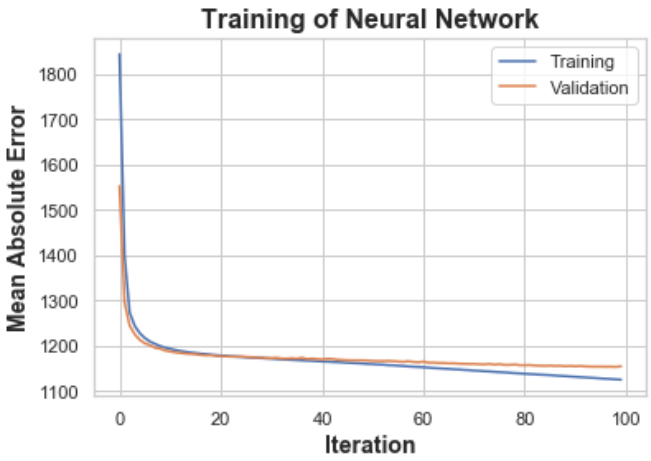

Figure 13: Training of Neural Network Model

The results of the Neural Network model were superior to any of the models attempted above. The MAE value of the training set was *1125* for the training data and *1154* for the validation data. The MAE on the original testing set was *1180*.

5 Conclusion

After running all the models, it is clear that the Neural Network model is the strongest at predicting loss value associated with an insurance claim as it had the best MAE results on the training, validation, and testing sets. In fact, the winning submission on Kaggle had an MAE of *1110* while our best model produced and MAE of *1180* on the testing set. This shows the strength that a simple neural network model can have in such problems.

Furthermore, the Random Forest model also produced positive results. With proper tuning and better computing power, this model can be fine-tuned to achieve better results. The Principal Component Regression was also a good model for this regression problem. It required creating the principal components for data set then using the optimal number of components to create a regression model. The Multiple, Ridge, and Lasso regression models produced similar results. It appears Regularization methods do not add much value to creating a model. **Table 1** summarizes the MAE results gained from each of the studied models.

Model	**Training MAE**	**Testing MAE**
Linear Regression	1299	1312
PC Regression	1240	1265
Ridge Regression	1287	1302
Lasso Regression	1286	1302
SVM	1296	1308
RF	467	1251
MLP	1125	1154

Table 1: Finals Results of Models

For further analysis, we hope to optimize some of the models above. First, explore advanced hyper-parameter tuning methods that can tune the models above. Second, it seems prudent to run the Random Forest model prior to running any logistic regression model as the Random Forest can help determine which features to select for building the regression models. Furthermore, we will explore different machine learning algorithms such as XGBoost, Adaboost, or other Boosting methods to improve the accuracy of the models built in this analysis. The winning competition on Kaggle utilized the XGBoost model. Lastly, we hope to overcome the computing power problem and utilize all the tables in the provided data set.

References

[1] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, New York, New York, 2008.

[2] Robert Tibshirani, Trevor Hastie, Gareth James, and Daniela Witten. *An Introduction to Statistical Learning*. Springer, New York, New York, 2013.

[3] Simon Haykin. *Neural Networks and Learning Machines*. Pearson, Upper Saddle River, New Jersey, 2009.

[4] Michael Galarnyk. *PCA using Python (scikit-learn)*.
`https://bit.ly/2XsRFdz`

[5] Kaggle. *Allstate Claims Severity*.
`https://www.kaggle.com/c/Allstate-claims-severity/data`

[6] Nagesh Singh Chauhan. *A beginner's guide to Linear Regression in Python with Scikit-Learn*.
`https://bit.ly/3a8y7y0`

[7] Will Koehrsen. *How to Visualize a Decision Tree from a Random Forest in Python using Scikit-Learn*.
`https://bit.ly/2ww5oFl`

[8] Kyoosik Kim. *Ridge Regression for Better Usage*.
`https://bit.ly/2yRqbnD`

[9] Saptashwa Bhattacharyya. *Ridge and Lasso Regression: L1 and L2 Regularization*.
`https://bit.ly/3eeGAD2`

6 Appendix

6.1 Appendix 1: Density Distribution

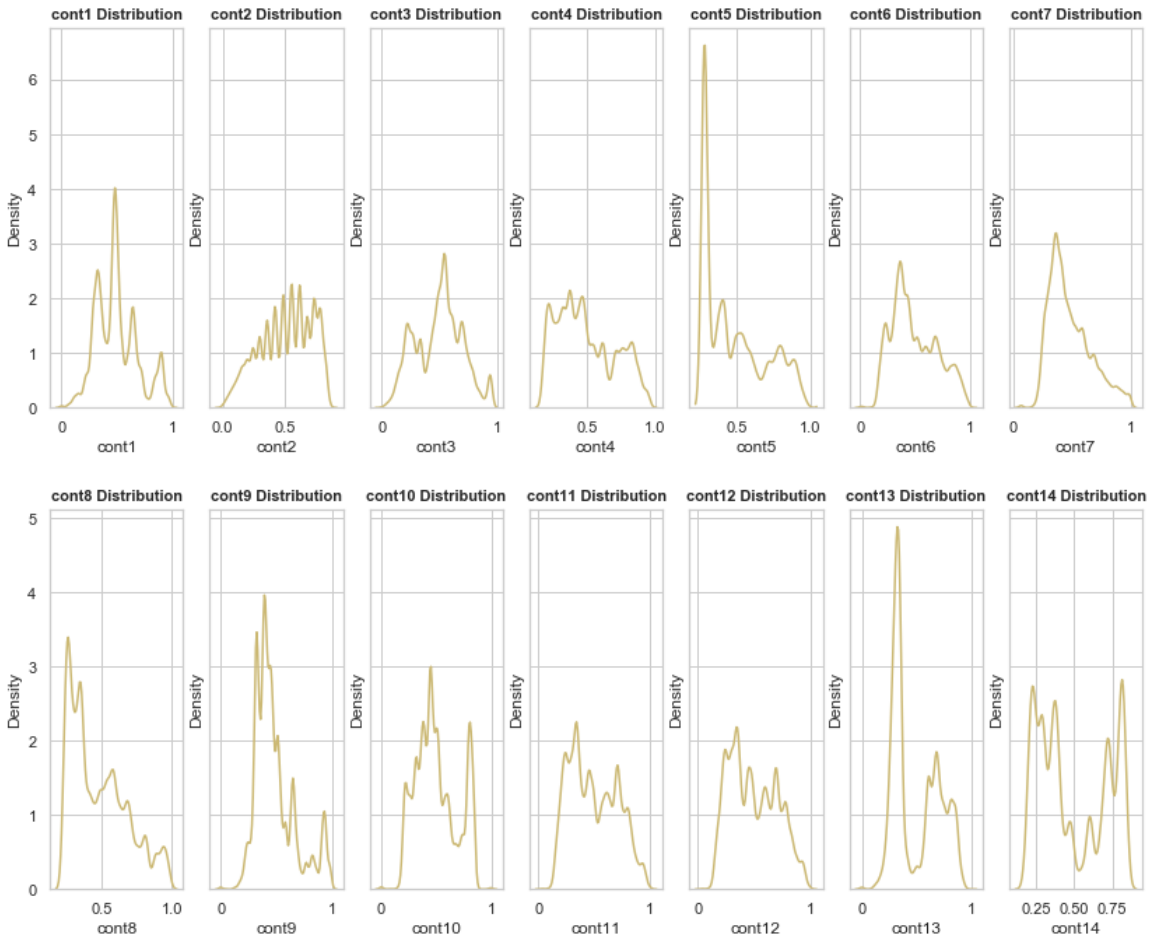

Figure 14: Density Distribution of first all 12 continuous variables

6.2 Appendix 2: PCA Visualization

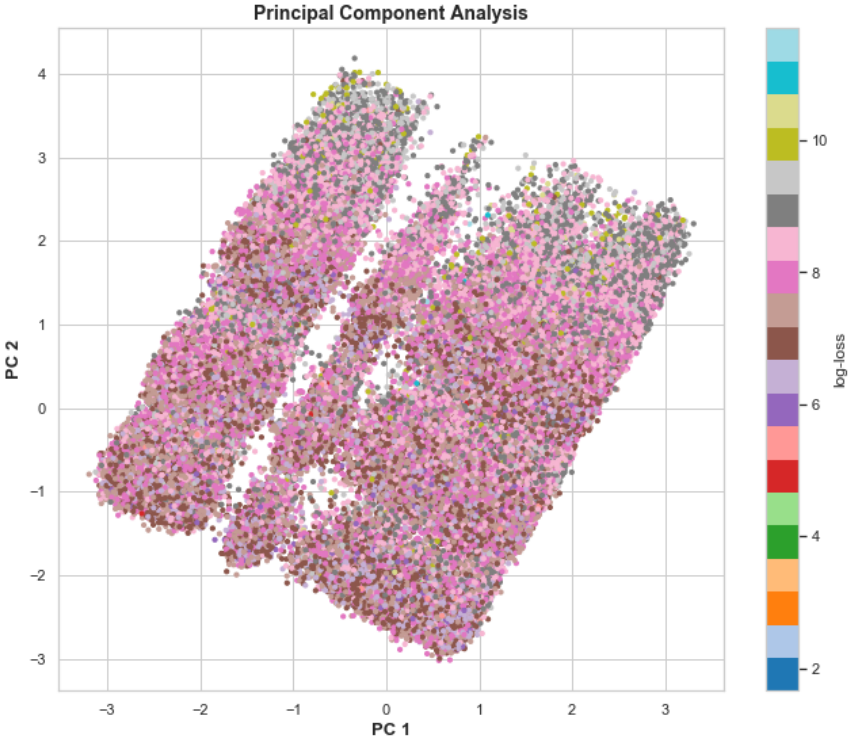

Figure 15: Principal Component 1 vs Principal Component 2