Machine Learning Engineer Nanodegree

Capstone Project – AllState Claim Severity

Amit Lathiya
April 17, 2019

I. Definition

Project Overview

When someone meets with any serious car accident or incurs property damage due to bad weather conditions or faces professional or personal liabilities or responsible for any commercial loss in business, filing Insurance Claim and spending time and mental energy pushing through paper work is last thing anyone wants to deal with. Having seamless and quick Claim settlement experience everyone expects in such circumstances.

I have chosen project from Kaggle competition to predict severity of Insurance claim ('AllState Claim Severity'). Allstate, a personal insurer in the United States is continually seeking ways to improve Claim service for the over 16millions households they protect. Allstate is currently developing automated methods of predicting the cost, and hence severity, of claims.

Knowing Claim severity helps insurer to align Claim assignments to appropriate groups and teams who have expertise and clearance authority to process such claims faster. Also Claim severity information is used by actuaries to access predicted future cost of claims which can be used in premium adjustments. This is the key strategy which drives the profitability of Insurance company. To remain highly competitive, Insurers must provide competitive pricing model for premiums to customers and knowing claim severity is critical information towards this strategy.

Problem Statement

Goal of this project is to predict the future cost of claim. Basically, cost of claim is nothing, but the total loss amount incurred to fully settle the claim. Knowing loss value helps insurer to categorize or assign severity to the claim. An example below illustrates how severity can be assigned to claim based on loss value range.

Loss range	**Claim Severity**
Loss >= 50,000	High
50,000 > Loss >= $5,000	Medium
5,000 > Loss > 0	Low

Now loss value of claim can be predicted from several claim features. These features are some form of information captured during initial notice of loss or during claim settlement process. These features can be both categorical or continuous in type.

Potential solution for the given problem statement to predict loss value from claim features can be solved through supervised linear regression training. Loss function to minimize will be difference of value between predicted Loss and actual loss. Once the model is trained, it can be used to predict loss value on test data.

Metrics

Mean Absolute error (MAE) is the metric I have chosen to evaluate performance of the model. MAE is measure of absolute difference between actual loss and predicted loss averaged over all samples. Its formula is given by below equation:

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

Here yi = Actual Loss, xi = Predicted Loss, n = number of samples

This metric is considered for the following reasons:

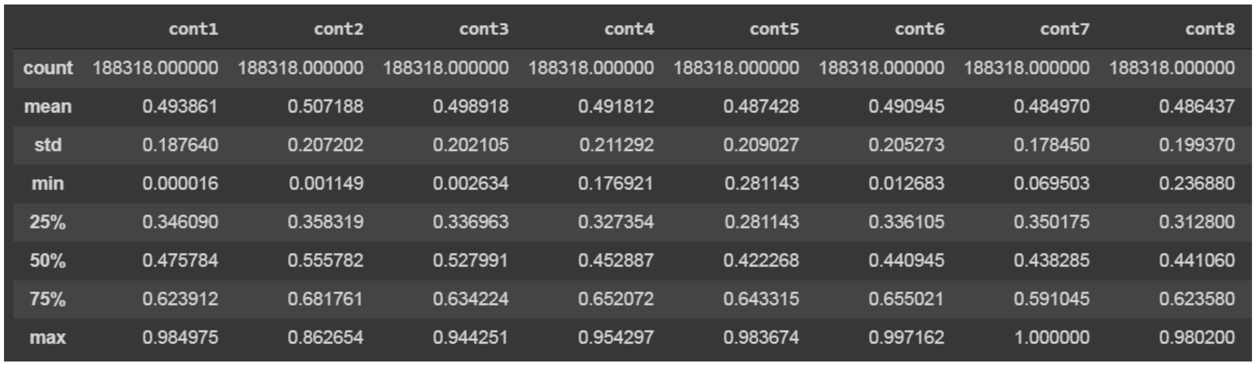
- In this problem statement, prediction of loss value is performed which is numerical quantity. Predicting numerical value from input feature is the case of linear regression. Closer the predicted value to actual, lesser the error term and more accurate regression function can be approximated.
- Another metric for regression problem is Mean Squared Error (MSE). In MSE, square of the difference is taken when evaluating error function. If the error term lies between 0 and 1 then MSE is preferred over MAE as square of number (error term) further reduces. But in given problem statement predicted Loss value can error by significant margin in hundreds or even thousands by positive or negative margin, so considering MAE is more viable as error term and is not further amplified by taking square.

II. Analysis

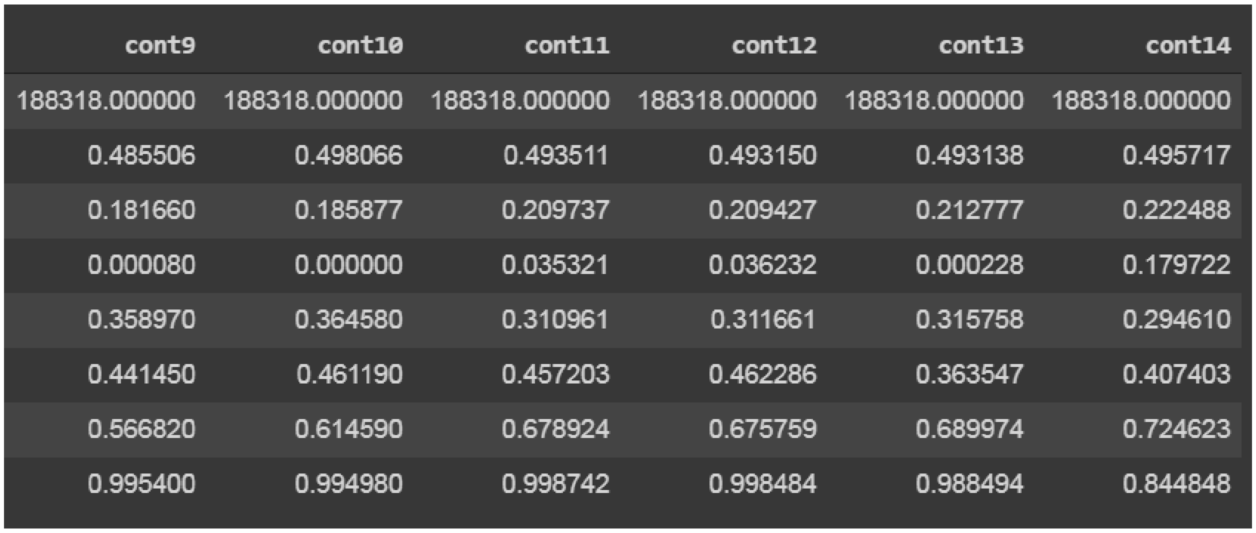
Data Exploration

Provided train.csv dataset from Kaggle has 131 features (excluding loss value column from dataset) and 188318 samples rows. Off total 131 features there are 14 features which are continuous in space (numeric values) and 116 features are categorical (discrete set of values). First column 'Id' represents unique row Id for each sample like index.

Analyzing statistics on 14 continuous features is done to determine what range of values are depicted by each of these features as shown below.



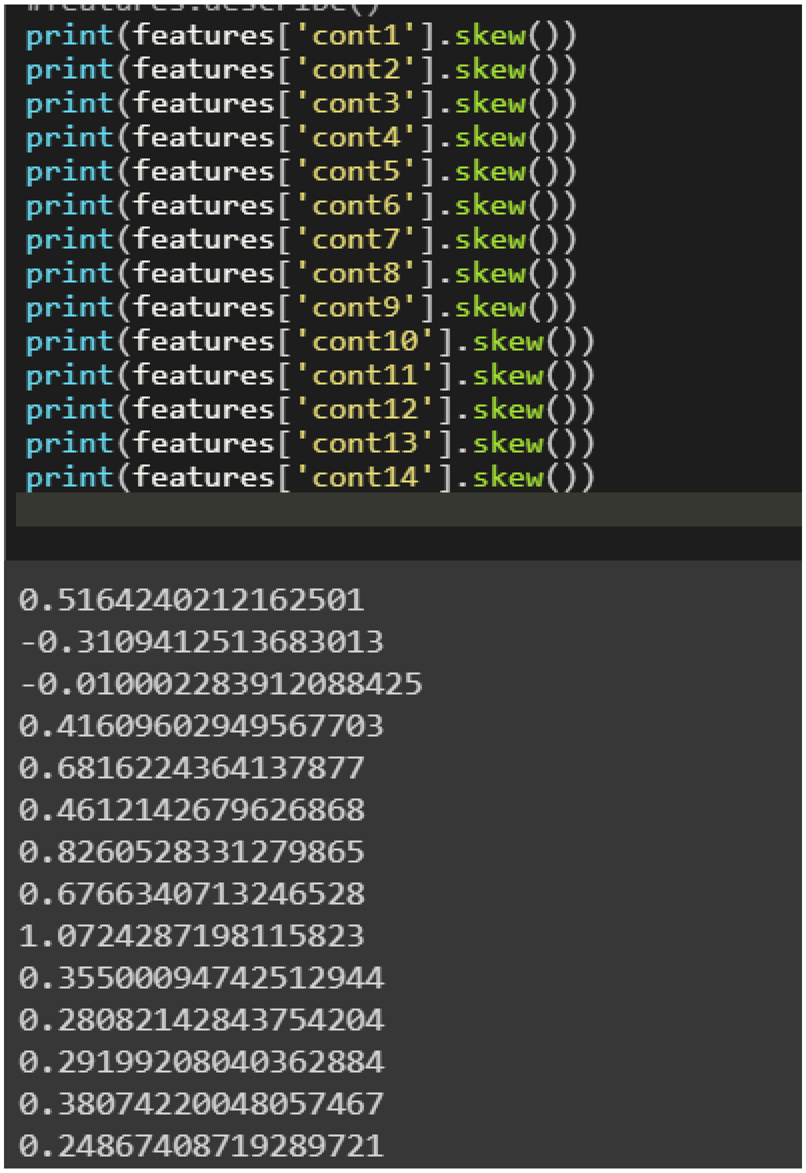
	cont1	cont2	cont3	cont4	cont5	cont6	cont7	cont8
count	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000
mean	0.493861	0.507188	0.498918	0.491812	0.487428	0.490945	0.484970	0.486437
std	0.187640	0.207202	0.202105	0.211292	0.209027	0.205273	0.178450	0.199370
min	0.000016	0.001149	0.002634	0.176921	0.281143	0.012683	0.069503	0.236880
25%	0.346090	0.358319	0.336963	0.327354	0.281143	0.336105	0.350175	0.312800
50%	0.475784	0.555782	0.527991	0.452887	0.422268	0.440945	0.438285	0.441060
75%	0.623912	0.681761	0.634224	0.652072	0.643315	0.655021	0.591045	0.623580
max	0.984975	0.862654	0.944251	0.954297	0.983674	0.997162	1.000000	0.980200



cont9	cont10	cont11	cont12	cont13	cont14
188318.000000	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000
0.485506	0.498066	0.493511	0.493150	0.493138	0.495717
0.181660	0.185877	0.209737	0.209427	0.212777	0.222488
0.000080	0.000000	0.035321	0.036232	0.000228	0.179722
0.358970	0.364580	0.310961	0.311661	0.315758	0.294610
0.441450	0.461190	0.457203	0.462286	0.363547	0.407403
0.566820	0.614590	0.678924	0.675759	0.689974	0.724623
0.995400	0.994980	0.998742	0.998484	0.988494	0.844848

As we can observe from above stats, values for all continuous feature lies between 0 and 1. Mean is closer to 0.5 and standard deviation is not more than 0.22. So, the data for all continuous feature is evenly distributed and normalized.

We can also verify this by checking skewness on each continuous feature using pandas skew() function as depicted in below figure. These values are smaller and none of these features indicate high skewness.



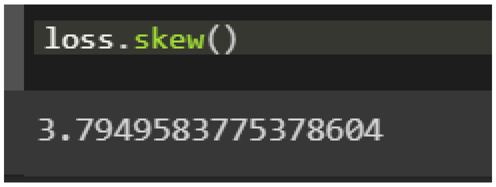
```python
print(features['cont1'].skew())
print(features['cont2'].skew())
print(features['cont3'].skew())
print(features['cont4'].skew())
print(features['cont5'].skew())
print(features['cont6'].skew())
print(features['cont7'].skew())
print(features['cont8'].skew())
print(features['cont9'].skew())
print(features['cont10'].skew())
print(features['cont11'].skew())
print(features['cont12'].skew())
print(features['cont13'].skew())
print(features['cont14'].skew())
```

```
0.5164240212162501
-0.3109412513683013
-0.010002283912088425
0.41609602949567703
0.6816224364137877
0.4612142679626868
0.8260528331279865
0.6766340713246528
1.0724287198115823
0.35500094742512944
0.28082142843754204
0.29199208040362884
0.38074220048057467
0.24867408719289721
```

Likewise, we can review stats for target feature Loss value as shown below.

```
Minimum Loss 0.67
Maximum Loss 121012.25
Mean Loss 3037.3376856699792
Standard Deviation Loss 2904.078475789109
Median Loss 2115.5699999999997
```

Here mean and median differs significantly with large standard deviation. So we can check data skewness on this column by calling skew() method on loss feature as shown below.



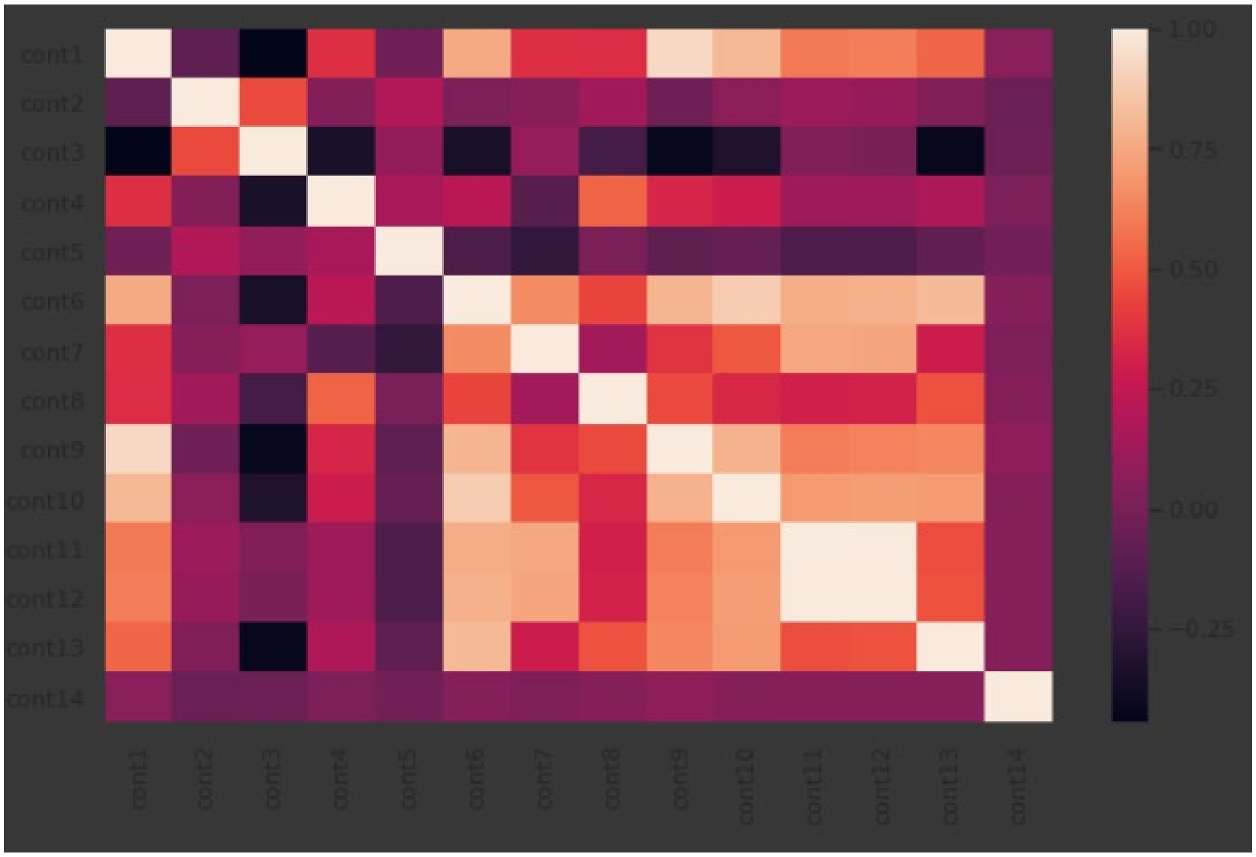
```
loss.skew()

3.7949583775378604
```

As data is highly skewed, we need to transform this data which I have further discussed in data preprocessing section.

Exploratory Visualization

In data visualization process, we need to analyze feature relevance. That is whether all features are independent of each other or there exists some degree of correlation among features. Determining correlation would help us to find out redundancy among features. Then we can eliminate similar features which further simplifies input to model.

To find out correlation among continuous feature I have used pandas .corr() method and pass it to seaborn heatmap to visualize Pearson correlation coefficient as shown in below figure.

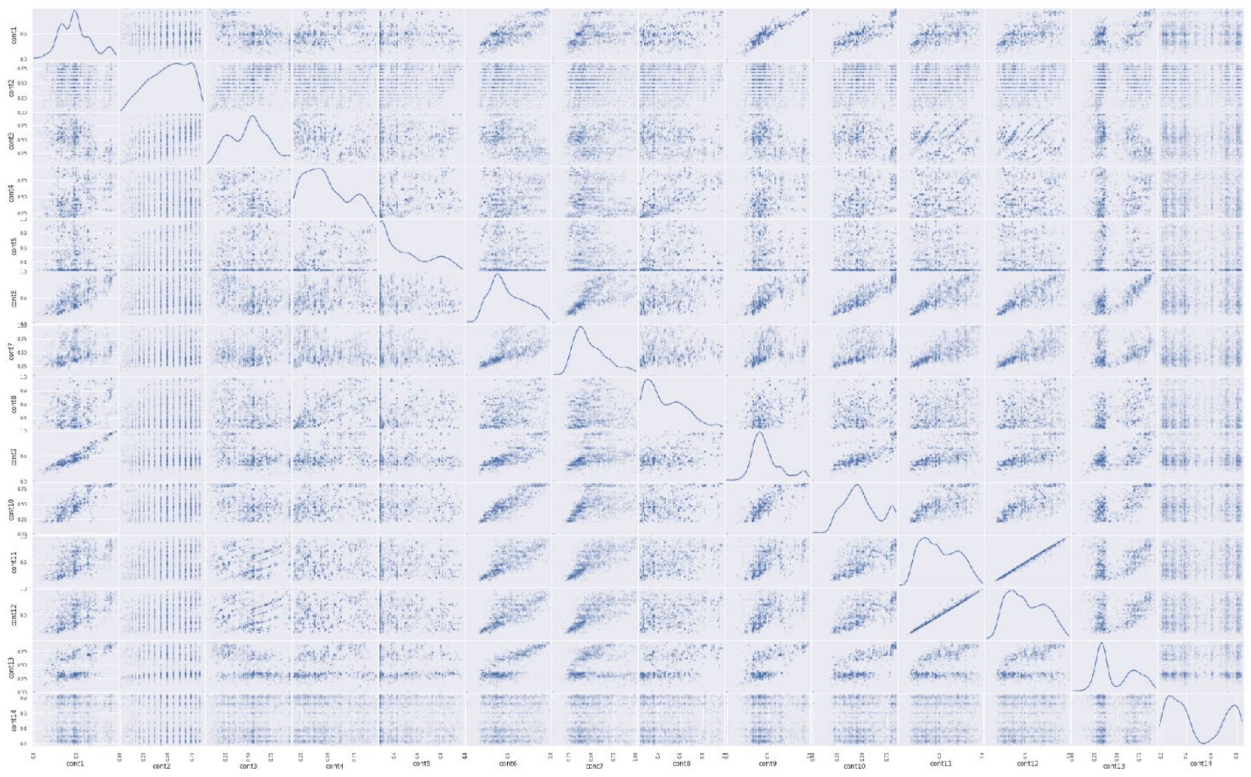


From above figure we can see lighter region in heatmap shows strong correlation and darker region depicts weaker correlation among feature.

Features showing strong correlation are listed below.

- Cont11 and cont12
- Cont1 and cont9
- Cont6 and cont10
- Cont1 and cont10

Another way to visualize correlation is through scatter plot. We can take sample of first 2000 rows and generate scatter plot for each pair of continuous features to visualize correlation.



Scatter plot visualization depicts similar correlations between features compared to what we discovered in heatmap visuals.

Now it's difficult to find the correlation among categorical variables as number of categorical variables are too many and dataset is large enough to effectively calculate and visualize these correlations. In this case we will use PCA for dimensionality reduction. More details will be discussed later in data preprocessing section.

Algorithms and Techniques

For the given dataset, number of categorical features are more than continuous features, it will be good case for tree-based algorithms such as Decision Tree regressor and XGBRegressor (provides both bagging and boosting capabilities). For setting optimal parameter we need to perform model complexity analysis. This will help us to set range for optimal parameter configuration otherwise providing broad range of parameters while training will significantly slow down performance.

Deep Neural network will also be good solution towards solving this regression problem as deep NN perform great when input dimension of data (features) are significantly higher. Using backpropagation technique, gradient decent step can be applied to loss function to minimize error for each batch of training samples.

I have design deep NN using keras with tensorflow as backend. Following hyperparameters have been tuned to optimize learning.

- Epochs – Number of iterations through training dataset
- Lr - Learning rate given to Adam optimizer
- Number of layers in network
- Batch_size – Number of training samples processed in single forward and backward pass
- Dropout – To avoid overfitting, I have set dropout probability after each layer

Training of deep NN model is done on GPU. Batches are randomly shuffled and loaded into GPU memory for processing. 10% of training data is used for inferencing (Validation). Model checkpoint saved for minimum validation loss. Later model is loaded from checkpoint to predict on test data.

Benchmark

Benchmark has been established on LinearRegressor model from sklearn. Training data is feed through LinearRegressor (with feature input and target) without much data preprocessing. Some of the basic preprocessing steps done on data are listed below.

- Removed feature 'Id' from train and test data as it doesn't provide any value for Loss prediction. It's like index attribute to uniquely identify each claim row.
- Categorical variables are one hot encoded so that numerical values are given as input to model.

After fitting through LinearRegression(considering all default parameters), prediction is done on test data to compute MAE. Some of the very high predictions are ignored as possible outlier.

MAE scored from this model training = 1275.98

III. Methodology

Data Preprocessing

Data preprocessing can be broken down into three phases as illustrated below in detail.

Feature transformation for Continuous variables

From the data visualization step we observed feature correlation for continuous variables through heatmap and scatter matrix plots. We noticed correlation among below set of features.

- Cont11 and cont12
- Cont1 and cont9

- Cont6 and cont10
- Cont1 and cont10

We can remove these redundant features from training dataset through PCA transformation. From the 14 continuous features, 6 features show strong correlation. So, we can select 3 features out of these 6 features which would uniquely represent these correlations. In this process we won't manually select which feature to remove instead use PCA to reduce dimensionality.

Below steps are followed to reduced dimensionality in continuous features.

- First step in this process is to split the training data into two datasets by feature type. Separating continuous features from categorical.
- On continuous feature dataset run PCA with number of components = 14 through fit and transform methods.
- Print the explained variance ratio for all 14 components.
- Here we notice that first 11 dimensions represents 99% of variance in data. This also supports our findings from heatmap and scatter matrix visualizations.
- Finally running PCA with number of components = 11.
- Resultant dataset represents transformed reduced continuous features dataset.

Feature transformation for Categorical variable

Its difficult to comprehend correlation among categorical variable when large number of categorical variables are present in dataset. PCA makes this process automated when optimal number of components are initialized in fit process. Below steps illustrates process for PCA done on Categorical variable dataset.

- Before running PCA on categorical variable we first need to one hot encode all variable because PCA can only be done on numeric features. I have used pandas dummies function to encode all categorical variable.
- After encoding we review dimension of dataset to find out resultant number of features. For categorical dataset we notice 1139 feature. (that is 116 cat variables are encoded into 1139 numeric representations)
- Now we run PCA on categorical train dataset with number of components = 1139.
- After fit and transform we print explained variance ratio for all 1139 feature.
- From explained variance ratio we can conclude that first 345 captures 99% variance in data. So, we can ignore rest of dimensions as their presence won't make much difference as far as variance is concerns.
- In final step we run PCA with number of components = 345. Resultant dataset represents transformed reduced categorical features dataset.

Normalizing skewed data

From earlier observation we discussed skewness of Loss target variable. Skewness in data can be reduce by performing log transformation. This step ensures data gets normalized without getting too much skewed towards certain range. Also effect of outliers gets reduced to much extent after transforming data to log space.

```
data skewness before log transform 3.7949583775378604
data skewness after log transform 0.09297454555136214
```

Implementation

After transforming continuous and categorical features from prior data preprocessing step, we can combine both datasets using numpy hstack method. This will be our final dataset ready to be feed into model training. Total feature in this dataset can be derived by adding number of components from continuous and categorical datasets (11 + 345 = 356).

Implementation is done on three models as illustrated below in details.

Decision Tree Regressor

DecisionTreeRegressor library is imported from sklearn.tree. An instance of this class is initialized with parameters passed. I have used max_depth and min_samples_split parameter since tree gets complicated with increased depths if input features are too many. Now the optimal value of these parameters is derived from model complexity graph (discussed in detail in Model Evaluation section). Below steps are followed to train regressor.

- Split the training dataset into train and test data so that some part of data is used for test prediction using sklearn train_test_split function. I have reserved 10% of data for test as metric is scored on this data.
- Instantiate Regressor with optimal max_depth and min_samples_split parameters value passed as an argument.
- Fit regressor with training features and target loss value.
- Run the prediction on test data reserved for model testing.
- Pass the predicted loss value and actual loss value to performance metric function which computes and returns MAE.

Deep Neural Network using Keras

I have used Keras sequential model with multi layer deep NN architecture to train regressor. Below steps are followed to train deep neural network regressor.

- Split the training dataset into train and test data so that some part of data is used for test prediction using sklearn train_test_split function. Split is done in ratio of 9:1.
- We have three FC layer (Fully connected) in this architecture with first layer comprising of same number of units as input feature dimension.
- Last layer outputs single value (predicted loss value) so we have used Linear activation function.
- Dropout layer is added after each layer except final layer to avoid overfitting. Dropout ensures that all trainable weights are given equal weightage avoiding model getting influenced by few weights.
- Adam Optimizer is used for this model with optimal learning rate.
- Model is complied with Loss function of ‘Mean_Absolute_Error’ and optimizer.
- Model Checkpoint is saved after every epoch to keep track of optimal model parameters (weight matrix) during training process to file.

- Fit function trains the model for set batch size for all epochs. 10% of data is used for inferencing. Every time validation loss is lesser than prior value then weights are saved to checkpoint file.
- After training, model is loaded from last saved checkpoint with best optimal parameters.
- Prediction is done on test data to compute and return MAE.

XGBRegressor

XGBRegressor is parallelizable onto GPU's and across networks of computers making it feasible to train on very large datasets as in our case. It is an implementation of gradient boosted decision trees designed for speed and performance.

The two reasons to use XGBRegressor

- Bagging and Boosting features
- Model Performance

I have used XGBRegressor API from XGBoost class. It has lots of trainable parameters. The ones used in training model are listed below:

- n_estimator = It's an int value which represents number of trees to fit
- booster = Specify which booster to use: gbtree, gblinear or dart
- subsample = Subsample ratio of the training instance

Boosting and Bagging process can be computationally very extensive and time consuming. With large dataset it becomes worse unless appropriate params are tuned to optimal values.

After instantiating XGBRegressor with optimal parameters, model is fitted with train data features and Loss target variable. Prediction is done on trained model with test data and finally MAE is computed and returned based on ground truth and predicted values.

Refinement

For all the three regression techniques, refinements have been done to improve model by reducing the MAE score from initial to final value as illustrated below.

Deep NN using Keras

Below table summarize various hyperparameter changes to tune model to improve MAE score. Please note lower the MAE better the performance score. Red highlights in table are changes in each of refinement process.

- With initial model configuration as shown in first row in table, MAE score of 1198.2 was recorded.
- Changed number of units in hidden layer from 100 to 200. With this step score become bit worse but this could be due to model overfitting and failing to generalize well on validation data. So, in next step I have added dropout to address this problem.
- Added dropout after each internal layer to avoid model overfitting. This resulted in improved MAE score.

- Increase batch size from 100 to 250. This resulted in two improvements
    1. Improved speed from initial 5s to 2-3s for each epoch
    2. Improved MAE score
- Reduced the learning rate (lr) to improve model performance.
- Increased the no of epochs to further improve model MAE score

epochs	**Batch_size**	**Lr**	**Dropout**	**Layers**	**MAE**
20	100	0.01	Not added	356X100X1	1198.42
20	100	0.01	Not added	356X200X1	1214.84
20	100	0.01	[0.3,0.2]	356X200X1	1213.51
20	250	0.01	[0.3,0.2]	356X200X1	1210.65
20	250	0.0001	[0.3,0.2]	356X200X1	1183.24
50	250	0.0001	[0.3,0.2]	356X200X1	1166.49

XGBRegressor

Below table summarizes parameters settings to improve score of this model. Generally, boosting algorithms are configured with weak learners, decision trees with few layers, sometimes as simple as just a root node, also called a decision stump rather than a decision tree. Weak learners give rise to strong learners as model training progresses. We need to ensure good number of weak learners if dataset is large enough. Here, we are setting n_estimators=1000 where simpler trees (such as decision stumps) require many more trees to achieve results. Shallow fewer trees would tend to overfit the data.

First, I tried setting booster to 'gblinear' thinking from linear regression perspective, but results were not good or even worse than benchmark. So, the next step was to set subsample size which brought great improvements. It denotes the fraction of observations to be randomly samples for each tree. Lower values make the algorithm more conservative and prevents overfitting, but too small values might lead to under-fitting.

With default value of subsample (subsample=1) we further noticed slight improvement which refers to fact that much of the rows represents similar correlation meaning random sample of data to much extend can fully represent whole dataset to derive predictions.

n_estimators	**subsample**	**booster**	**MAE**
1000	Default	gblinear	1499.38
1000	0.5	Default	1202.45
1000	Default	Default	1201.28

DecisionTreeRegressor

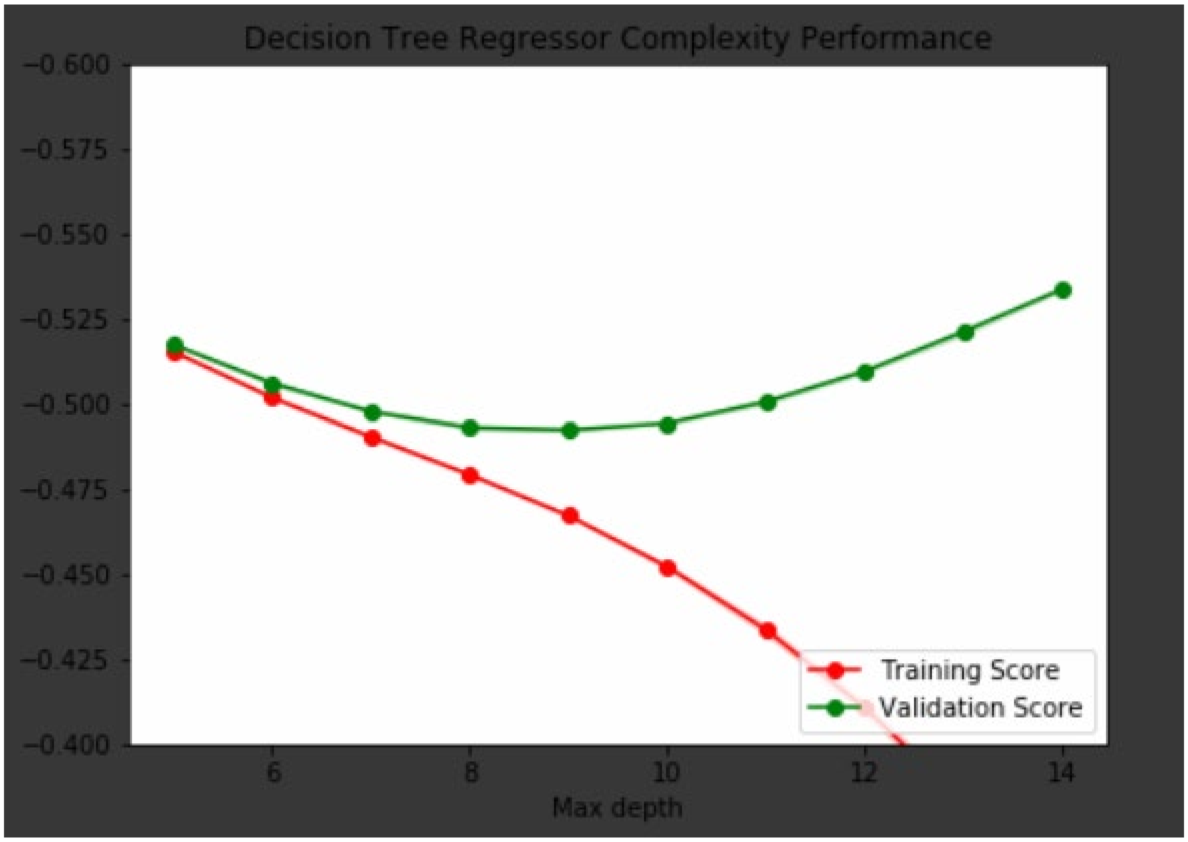
Initially DecisionTreeRegressor model with default parameter setting is run to baseline MAE score as reported in table below. MAE from this execution is very high and significantly higher than model benchmark we evaluated earlier. So, to make refinement model complexity analysis is performed as illustrated below in detail.

Model Complexity Analysis:

In this process we have to find the most optimal parameter setting which gives best MAE score. I have chosen max_depth parameter with range options from 5 to 15.

max_depth = [5,6,7,8,9,10,11,12,13,14,15].

Validation curve of training and testing scores are plotted with max_depth on x-axis and scores on y-axis. Scoring function 'neg_mean_absolute_square' is used for evaluation.



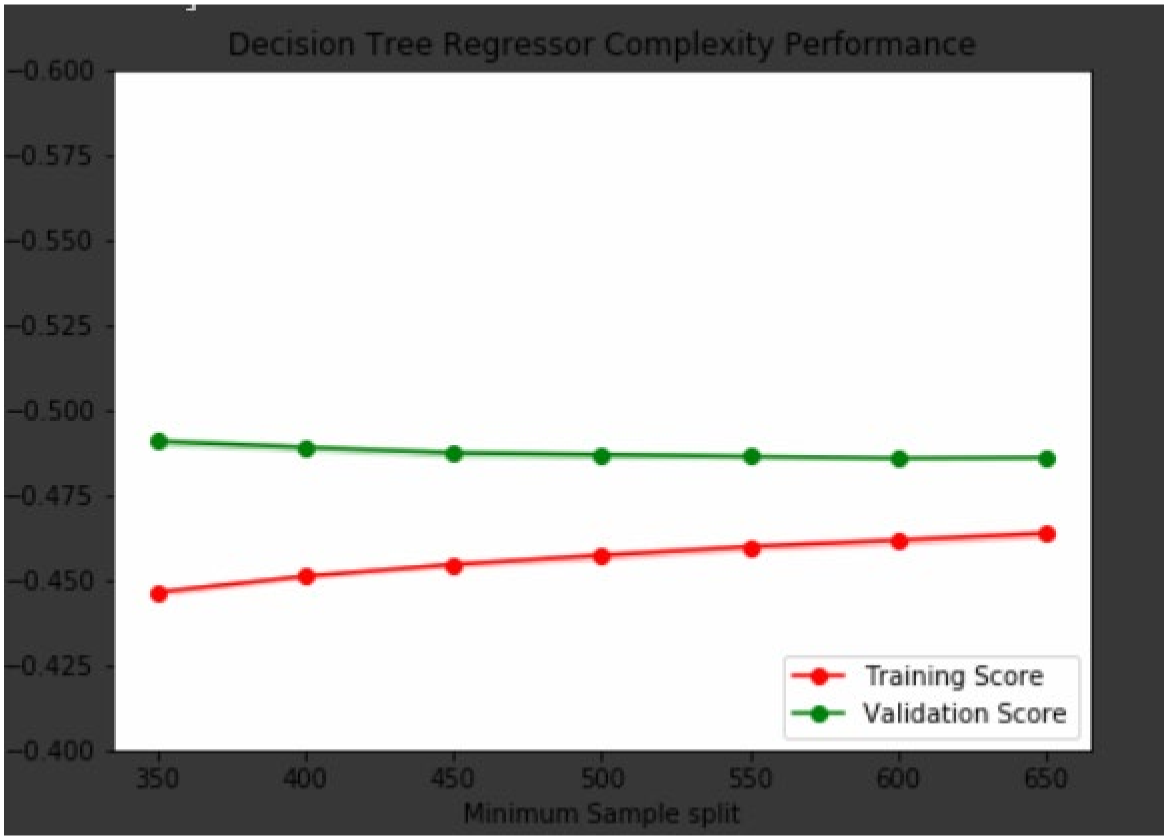
Observation:

As we can observe from model complexity plot above,

- For max_depth=[5,6,7], both training and testing scores are low showing high bias and model is underfitting.
- For max_depth=[10,11,12,13,14,15], training score keeps increasing as max_depth is increased from 10 to 15 but on other hand, testing score keeps decreasing as max_depth is increased from 10 to 15. This shows sign of high variance as model is overfitting and failing to generalize well.
- For max_depth=[8,9] we observe both train and test score are highest and there is not much divergence. Though testing score looks slightly higher for depth 9. We will run regression for both values of depth and evaluate the resulted MAE score.

MAE we evaluated from max_depth parameter analysis is still higher than benchmark we established from LinearRegression so we look for more ways to optimize and check if the further improvement can done. Below figure depicts model complexity curve for parameter min_samples_split. It represents minimum number of samples required to split node in Decision Tree. Train and testing curve are plotted for each value of min_samples_split param options.

min_sample_split = [350,400,450,500,550,600,650]



Observations:

- For values less than 500 we observe low train and testing score depicting high bias and underfit.
- For values 500 onwards we observe train and testing curves starts converging. Testing score almost stabilizes without further increase and train score goes down as min_samples_split increases. So, we get best optimization for value 500.

In below table model MAE score are evaluated with parameters settings done based on model complexity analysis performed.

max_depth	**min_samples_split**	**MAE**
Not specified	Not specified	1857.98
8	Not specified	1358.67
9	Not specified	1357.56
9	500	1350.60

IV. Results

Model Evaluation and Validation

Comparing the results from three regression model, we select the one as final model which gives best prediction and lowest MAE. Here, I have taken best MAE for each of the model evaluated earlier.

Model	**MAE**
DecisionTreeRegressor	1350.60
XGBRegressor	1201.28
Deep NN	1166.49

Deep NN outperforms the other models in evaluation process. Testing the robustness of the Deep NN requires how model performs when it sees variation in input and test data.

We used sklearn train_test_split() splitter function to randomly slice 10% of data for model testing on which prediction is done to evaluate MAE. To test robustness of this model we will run multiple iterations of model training changing the train and test data each time model evaluates MAE.

I have chosen to iterate this process three times. In each loop train_test_split function segregates train and test data with 9:1 split ratio. Random seed has been initialized and data shuffle is true by default.

In each training process, keras model can separate a portion of training data into a validation dataset and evaluate the performance of your model on that validation dataset for each epoch.This can be done by setting the validation_split argument on the fit() function to a percentage of the size of training dataset. Here we have specified that as 0.1.

After completion of each model training iteration, its MAE score has been added to list. Once all iteration is over, mean value of MAE score is calculated with standard deviation as shown below.

Iteration 1	1168.28
Iteration 2	1169.56
Iteration 3	1167.54
Mean MAE	1168.46
Standard Deviation	0.835

This process validates robustness of model as Standard deviation is very low and MAE score is consistent within narrow range.

Please note that K-Fold cross validation and StratifiedShuffleSplit cannot be performed for this regression problem as target data (loss value) is continuous in nature. It only works in case of categorical target data. But there is another way through which continuous data can be converted into categorical data through digitization(https://medium.com/@danilzherebtsov/continuous-data-stratification-c121fc91964b).

Justification

Both the benchmark and final model predictions are done on 10% test data with the MAE score as reported below.

	Model	MAE
Benchmark model	LinearRegression	1275.98
Final model	Deep NN using Keras	1168.46

Robustness of final model has been tested with random split of data with multiple iterations of training as discussed in model validation section. Model doesn't change prediction much with variation in input data with recorded standard deviation of 0.835.

Final model MAE is 107.53 less than benchmark established which is significant error considering huge volume of data. If we add up this error difference cumulatively for all rows, then it amounts to 20,249,834.54 (107.53 * 188318) which is substantial Loss value and can make difference to decision making.

Also, for the benchmark model we ignored high predictions or outliers, but in final model evaluation that has been taken into consideration.

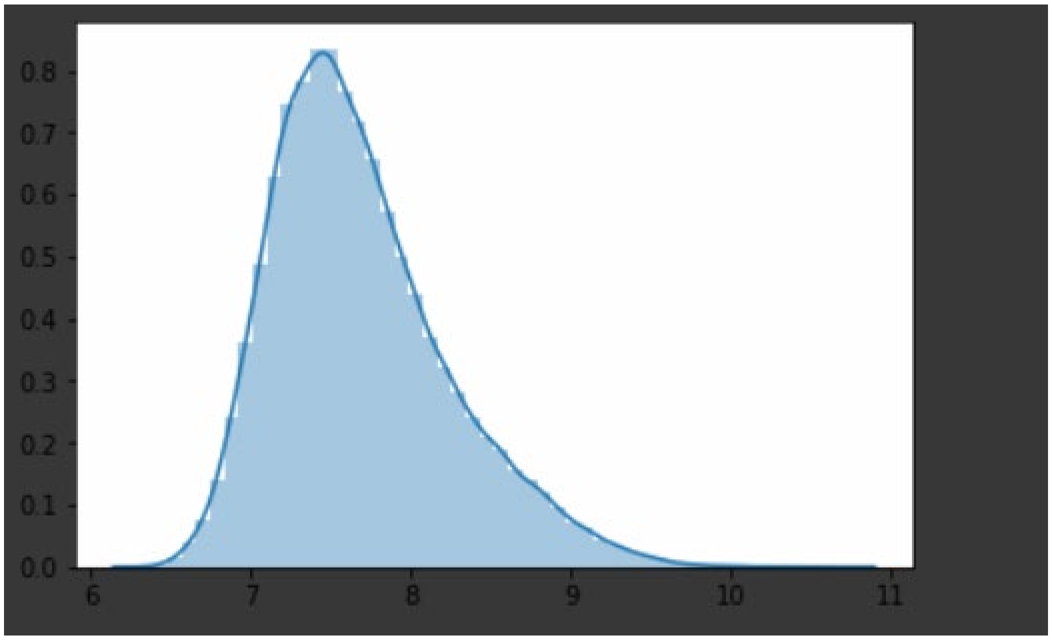
V. Conclusion

Free-Form Visualization

In this section, I have done visualization on test dataset provided in Kaggle competition. Though we don't have true predictions on test data but can get some visuals on predicted values obtained from running test data on trained model.

Test dataset has undergone similar data transformation (PCA) as what we did for train dataset. Model is loaded from the saved checkpoint (created during training process) and final prediction is run on all the test dataset rows 125546.

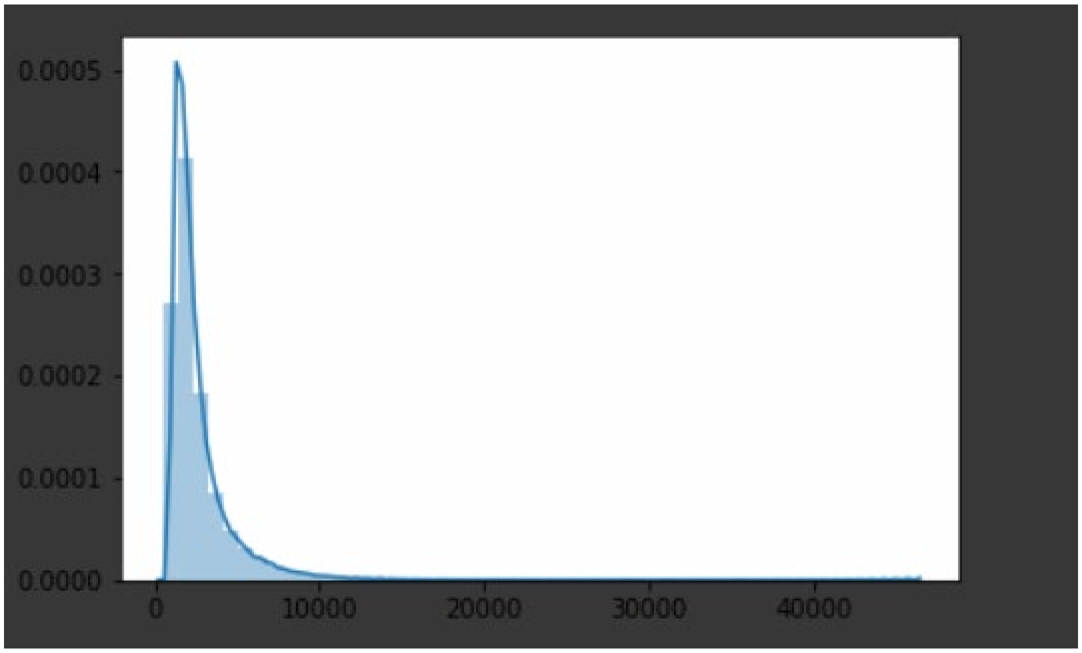
Below figures depicts predicted log output.



Log Scale

Here distribution depicts good normalization pattern for Loss log result.

Below figure shows actual predicted distribution when converted from log scale to actual data.



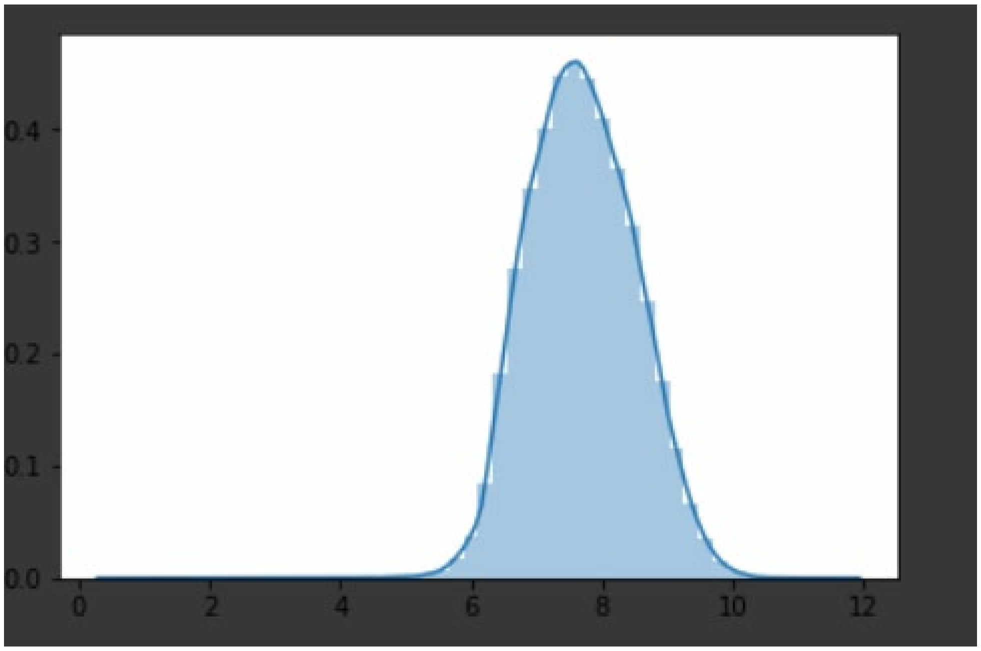
Actual Data

As we can observe much of the predicted Loss lies between 500 to 10000. There are few very high predictions as can be seen from long tail stretching more beyond 40000.

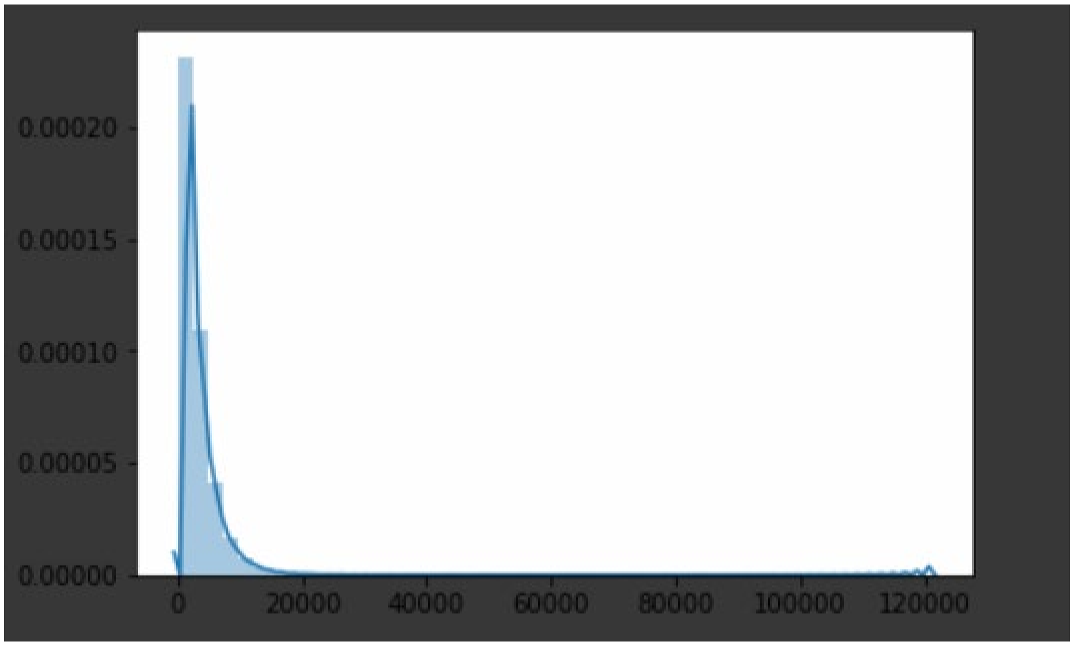
Reviewing stats from predicted loss we conclude much of the Loss value is around 1988.92 with some difference from mean of 2595.51 as there are few high predictions which are stretching mean higher than median.

- Mean – 2595.51
- Standard Deviation – 1931.66
- Minimum – 545.28
- Maximum – 46140.69
- Median – 1988.92

Now performing visualization on training data, we see similar normalization pattern in Loss value in log scale and actual data.



Log scale



Actual Data

- Mean – 3037.34
- Standard Deviation – 2904.08
- Minimum – 0.67
- Maximum – 121012.25
- Median – 2115.57

Like in test data, here also we observe few of the high loss values stretches mean higher than median.

It's interesting to observe both Train and Test data follows similar data distribution pattern and it is also evident from the fact that these are auto insurance claims reported by Allstate.

Reflection

This project has been interesting journey. My initial work was more towards understanding the problem domain and statement, why AllState is trying to solve this problem, what benefits it can provide to customer directly or indirectly. Being from Insurance IT Claims background makes this project more interesting for me as I can correlate what differences it can make to life of customers. With fast and efficient claim processing to better price optimization for premium, this predictive modelling benefits both Insurer and its customer significantly.

We started with data analysis, understanding statistics on data distribution, type and number of features involved (continuous vs categorical). After this we did data preprocessing by studying correlation between features. We looked at the ways on how best we can visualize and interpret correlation (heatmaps vs scattermatrix), understanding data skewness and finally we did data transformation and dimensionality reduction through PCA technique.

After doing much of data interpretation and feature engineering we evaluated different implementation techniques for Regression. Starting with model benchmarking using LinearRegression we evaluated three model techniques to train model.

- DecisionTreeRegressor
- XGBRegressor
- Deep NN using Keras

We evaluated each of these implementations, this phase of project was challenging as I had to re-run and perform optimization (hyperparameter tuning) for each model. Determining which hyperparameter to tune and how it can bring optimization required lot of research. This led me to do model complexity analysis and choosing right parameter value for optimum results.

It was interesting to train deep neural network as with each optimization step I could see improvement in MAE score. Adding dropout layer to model, changing layer architecture, changing learning rate, playing around with batch size etc was good learning on how to make model more efficient.

From all three models, Deep NN performed best with MAE lowest. We also tested robustness of this model by running multiple iterations with variation in train and test data.

Lastly, I analyzed prediction on Kaggle test data and reviewed the normal distribution and statistics of Loss amount. We observed how this distribution matched with train Loss data concluding that we expect to see similar trend in both dataset as these are auto insurance claims.

Improvement

Some of the potential areas of improvements we can make to this project are listed below.

- In Data preprocessing, we analyze correlation between continuous vs continuous features, categorical vs categorical features. Here we could also derive correlation between categorical vs continuous features through LinearRegression technique. Predicting each of the continuous feature from 116 categorical variable and comparing against truth can show degree of correlation involved. This would certainly require some good time and effort as it needs to be done for each continuous feature but it’s worth the effort.
- In data validation, we discuss the limitation of target variable being continuous as we cannot perform StratifiedKfold and we adopted another technique to bring variation in data but if we digitize target continuous variable into categorical (that is binning data to make it categorical) then Stratifiedkfold can be applied to test robustness of model.
- We adopted different model optimization approaches and there is scope for further optimization. Like for DecisionTreeRegressor we performed model complexity analysis on two params (max_depth and min_samples_split) and we can consider more params for optimization which can bring potential improvement in MAE score. Likewise, more research can be done on XGBRegressor to improve its performance runtime and perform further parameter optimization. XGBRegressor has great potential and its comparable to deep NN when it comes to scoring.
- Further improvement can be made to deep NN to improve score. Working more on hyperparameter tuning, experimenting with network architecture and trying different activation functions can potentially bring more improvements to minimize MAE.

Some of the other ML techniques that can be applied to this problem domain listed below can be evaluated as scope for improvement:

- Polynomial Regression
- Support Vector Regression
- Elastic Net Regression