Machine Learning Engineer Nanodegree

Allstate Claim Severity Capstone Project

Jose Garcia
December 31st, 2050

I. Definition

Project Overview

Allstate is on the largest personal insurance companies in the United states and protects millions of people from potentials damages. When car accidents happen or when other unexpected disasters occur it is a huge relief knowing you are cover by some type of insurance. Recently my girlfriend was involved in a car accident that almost totaled her car. Although she escaped with no bodily injuries her car had thousands of dollars in damages. Luckily, she has full coverage insurance and was able to get all repairs paid for and a rental car without paying anything out of pocket. This event peaked my interest in to insurance business which lead me to this project. Although for this particular instance the insurance claim process was easy and straight forward it is not always the case. This is why Allstate is currently developing automated methods of predicting the cost, and hence severity, of a claim. This project will provide insight into better ways to predict claims severity.

Dataset can be found on the Kaggle website at https://www.kaggle.com/c/allstate-claims-severity/data (https://www.kaggle.com/c/allstate-claims-severity/data)

Datasets and Inputs

For this project Allstate provides a test.csv and train.csv.

The train.csv contains the following:

- claim id
- cat 1 to cat 116 - this data is category based and contains either single letters or two letters
- cont1 to cont14 - this data is a continues set of numbers that are not negative
- loss - this is the ammount that the insurance company has to payout and this is also the target variable
- 188,318 rows
- 132 columns

The test.csv file containt the same information as train.csv except for the loss because it needs to be predicted.

- 89,023 rows
- 131 columns

Problem Statement

Allstate is attempting to improve its customer experience by automating methods of predicting the severity of a claim. The severity of a claim is based on a combination of different parameters that are unique to every claim. An insurance adjuster takes a look at each claim and all of its parameters and determines how much will be paid out to the customer. In some cases, the claim is quickly resolved but there is some instances in which it takes a long time to resolve the claim. For this project a training set of data will be used to train a machine learning algorithm to predict the cost of a claim based on the given parameters of the claim. This data will be used to train a linear regression learning algorithm to predict the loss.

Metrics

Kaggle will be evaluating this project using mean absolute error and the lower the mean absolute error the better. Mean absolute error is a measure of the average difference between the actual loss and predicted loss. Other scoring functions could have been used like r2 score or RMSE. The r2 score calculates the variance in the dependent variable that is predictable from he independent variable. This will not work for this particular problem because the data is better understood by calculating magnitude of different of the actual and predicted value. MAE weighs every value the same as opposed to RMSE which penalizes some values more than others. For this problem MAE is better suited because of its simplicity. Also, the fact that this a Kaggle competition, MAE works better for determining a winner out of thousands of entries.

HIOX

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

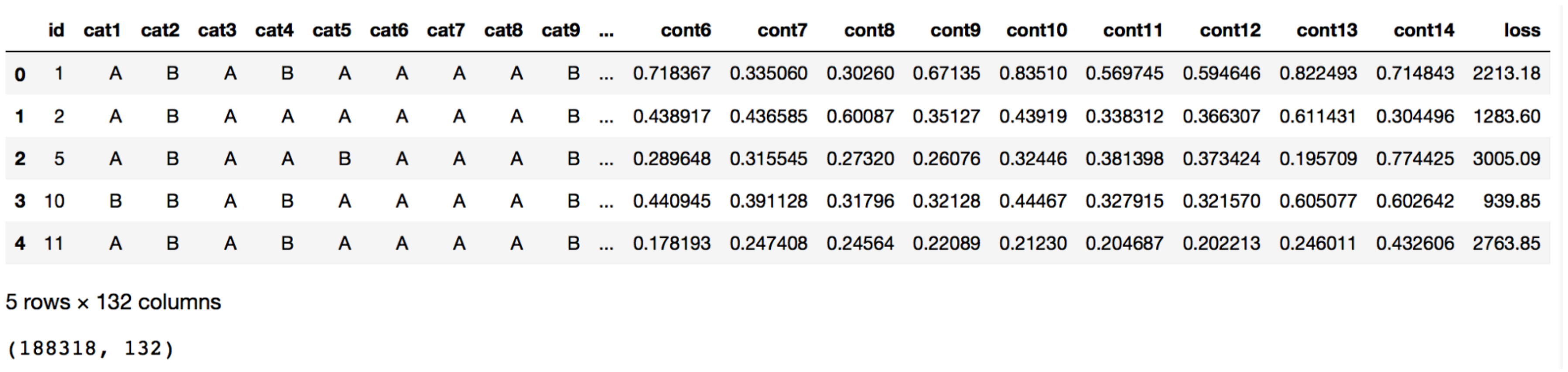
©easycalculation.com

Figure 1: Mean Aboslute Error

II. Analysis

Data Exploration

Below are the first five rows of the "trian.csv" data set. The data set is composed of an id number, category data, continues data, and a column labeled "loss". There is a total of 188,318 rows and 132 columns of data but not all of these parameters are useful for building our predictive model. The column labeled "id" will be dropped because it does not add any value to our algorithm. The "loss" column will also be dropped because this is the value that needs to be predicted and should not be mixed in with our features. Now there are only two types of data, non-numeric, which is composed of letters and numerical data which is composed of continues numbers. There is a total of 14 columns of continues data and 116 columns of non-numeric data. Right below the "train.csv" data set is the "test.csv" data which is exactly the same except that it does not have the "loss" column.

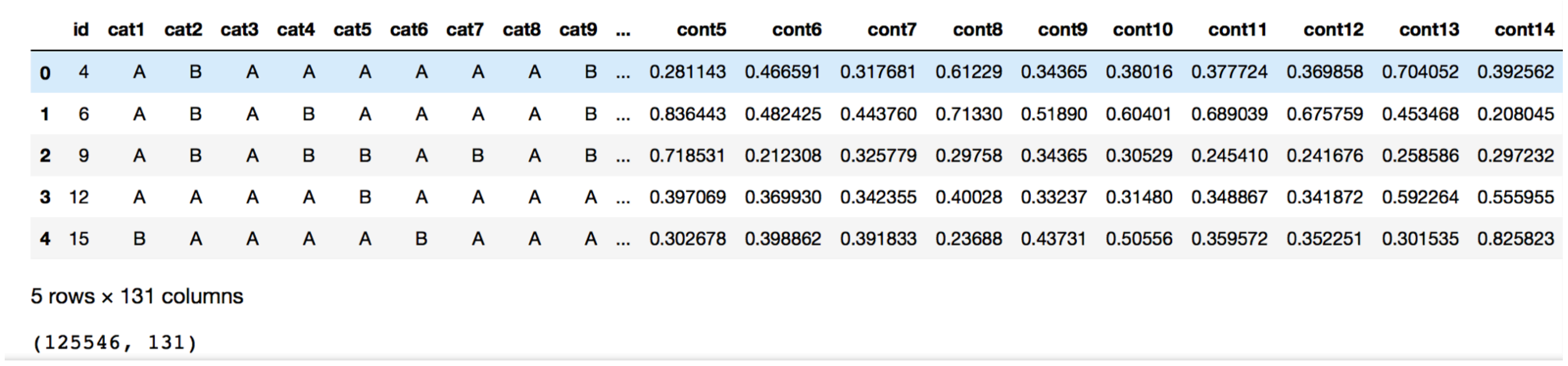


	id	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	...	cont6	cont7	cont8	cont9	cont10	cont11	cont12	cont13	cont14	loss
0	1	A	B	A	B	A	A	A	A	B	...	0.718367	0.335060	0.30260	0.67135	0.83510	0.569745	0.594646	0.822493	0.714843	2213.18
1	2	A	B	A	A	A	A	A	A	B	...	0.438917	0.436585	0.60087	0.35127	0.43919	0.338312	0.366307	0.611431	0.304496	1283.60
2	5	A	B	A	A	B	A	A	A	B	...	0.289648	0.315545	0.27320	0.26076	0.32446	0.381398	0.373424	0.195709	0.774425	3005.09
3	10	B	B	A	B	A	A	A	A	B	...	0.440945	0.391128	0.31796	0.32128	0.44467	0.327915	0.321570	0.605077	0.602642	939.85
4	11	A	B	A	B	A	A	A	A	B	...	0.178193	0.247408	0.24564	0.22089	0.21230	0.204687	0.202213	0.246011	0.432606	2763.85

5 rows × 132 columns

```
(188318, 132)
```

Figure 2: train.csv



	id	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	...	cont5	cont6	cont7	cont8	cont9	cont10	cont11	cont12	cont13	cont14
0	4	A	B	A	A	A	A	A	A	B	...	0.281143	0.466591	0.317681	0.61229	0.34365	0.38016	0.377724	0.369858	0.704052	0.392562
1	6	A	B	A	B	A	A	A	A	B	...	0.836443	0.482425	0.443760	0.71330	0.51890	0.60401	0.689039	0.675759	0.453468	0.208045
2	9	A	B	A	B	B	A	B	A	B	...	0.718531	0.212308	0.325779	0.29758	0.34365	0.30529	0.245410	0.241676	0.258586	0.297232
3	12	A	A	A	A	B	A	A	A	A	...	0.397069	0.369930	0.342355	0.40028	0.33237	0.31480	0.348867	0.341872	0.592264	0.555955
4	15	B	A	A	A	A	B	A	A	A	...	0.302678	0.398862	0.391833	0.23688	0.43731	0.50556	0.359572	0.352251	0.301535	0.825823

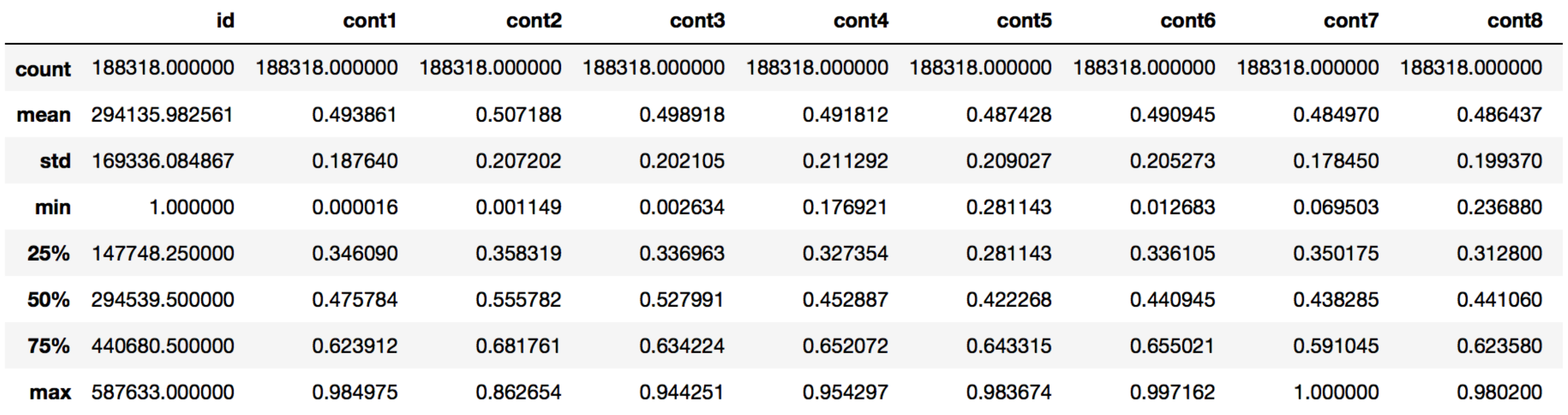
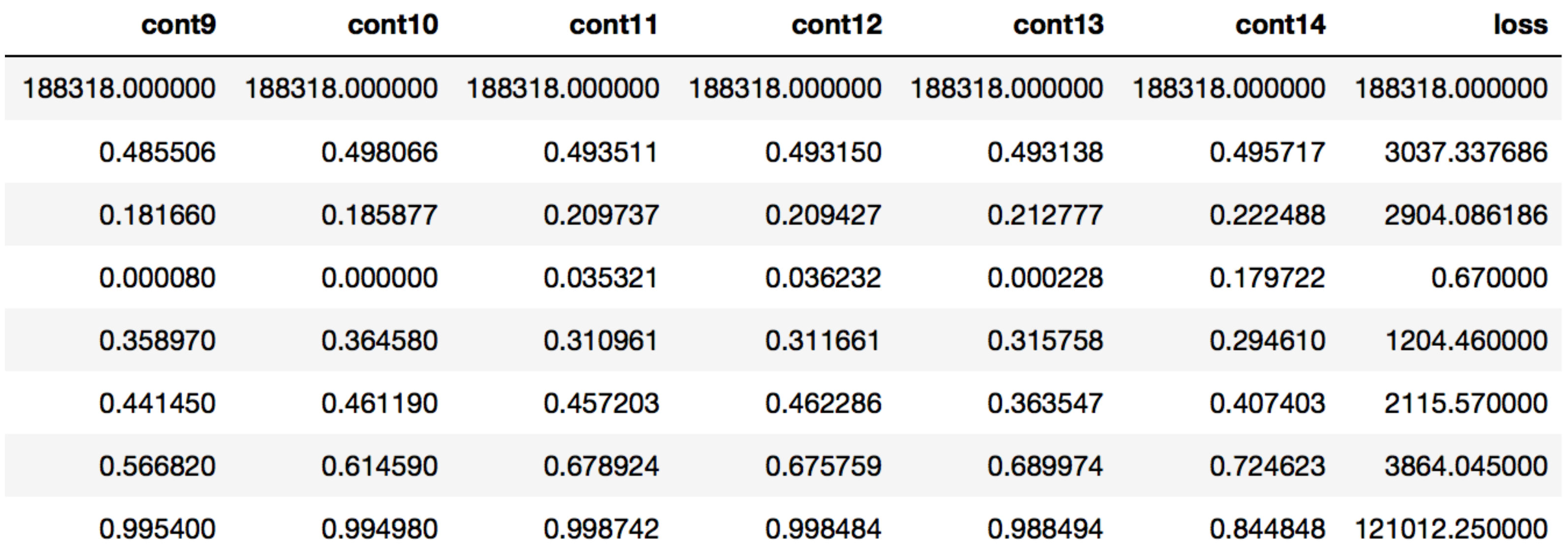
5 rows × 131 columns

```
(125546, 131)
```

Figure 3: test.csv

From this point on the "train.csv" data is the only data set that will be analyzed.

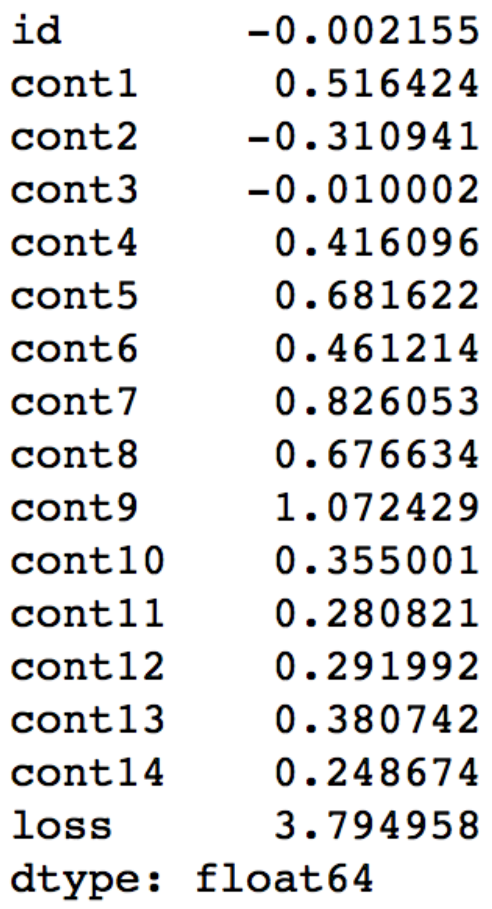
In order to get a better understanding of the data it must be represented in different forms in order to get different types of information. Running basic statistical analysis on the continues data reveals key facts. A few things to take away from the statistical parameters below is that the maximum value for all of the continues parameters is about 1 and the mean average number is about 0.5. Because the range of the continues numbers is small they are prefect to use with a learning algorithm. The "loss" column on the other hand has a wide range of values and this will be problematic when applying it to a learning algorithm.

	id	cont1	cont2	cont3	cont4	cont5	cont6	cont7	cont8	cont9	cont10	cont11	cont12	cont13	cont14	loss
count	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000
mean	294135.982561	0.493861	0.507188	0.498918	0.491812	0.487428	0.490945	0.484970	0.486437	0.485506	0.498066	0.493511	0.493150	0.493138	0.495717	3037.337686
std	169336.084867	0.187640	0.207202	0.202105	0.211292	0.209027	0.205273	0.178450	0.199370	0.181660	0.185877	0.209737	0.209427	0.212777	0.222488	2904.086186
min	1.000000	0.000016	0.001149	0.002634	0.176921	0.281143	0.012683	0.069503	0.236880	0.000080	0.000000	0.035321	0.036232	0.000228	0.179722	0.670000
25%	147748.250000	0.346090	0.358319	0.336963	0.327354	0.281143	0.336105	0.350175	0.312800	0.358970	0.364580	0.310961	0.311661	0.315758	0.294610	1204.460000
50%	294539.500000	0.475784	0.555782	0.527991	0.452887	0.422268	0.440945	0.438285	0.441060	0.441450	0.461190	0.457203	0.462286	0.363547	0.407403	2115.570000
75%	440680.500000	0.623912	0.681761	0.634224	0.652072	0.643315	0.655021	0.591045	0.623580	0.566820	0.614590	0.678924	0.675759	0.689974	0.724623	3864.045000
max	587633.000000	0.984975	0.862654	0.944251	0.954297	0.983674	0.997162	1.000000	0.980200	0.995400	0.994980	0.998742	0.998484	0.988494	0.844848	121012.250000

Figure 4: Data Description

Now the skewness of the parameters are analyzed to get and understand of the overall distribution of the data. The only parameter that significantly stands out is "loss".



```
id         -0.002155
cont1       0.516424
cont2      -0.310941
cont3      -0.010002
cont4       0.416096
cont5       0.681622
cont6       0.461214
cont7       0.826053
cont8       0.676634
cont9       1.072429
cont10      0.355001
cont11      0.280821
cont12      0.291992
cont13      0.380742
cont14      0.248674
loss        3.794958
dtype: float64
```

Figure 5: Skewness

Figure 6 shows the loss data plotted on a histogram chart and we can see that the data is skewed to the right, so it needs to be normalized. Regression algorithms can be sensitive to the distribution of values and can results in the model underperforming if the data is not normally distributed. Figure 6 also shows logarithmic transformation applied to the "loss" data so that it does not negatively affect the performance of the learning algorithm. Appling the transformation will reduce the range of the values.

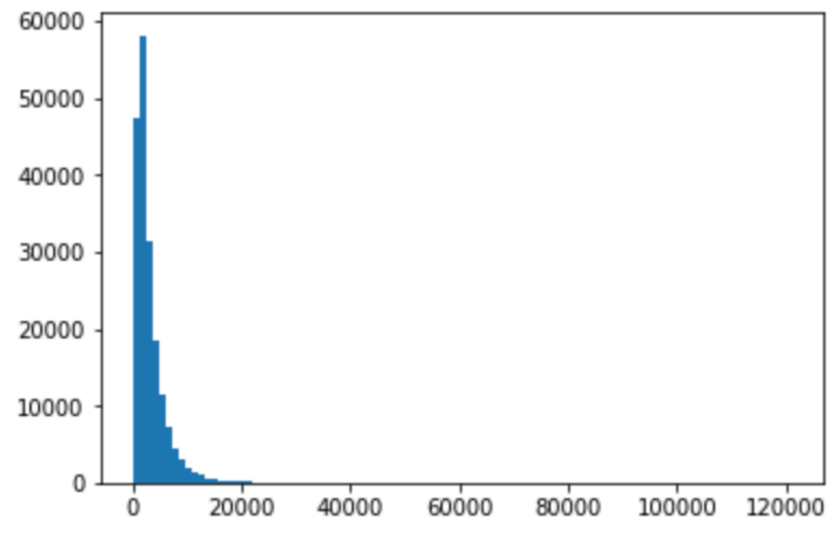
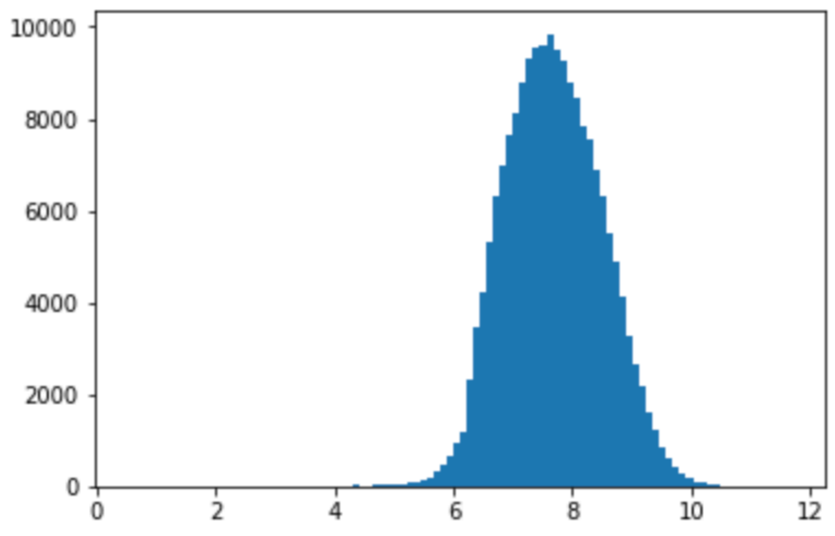
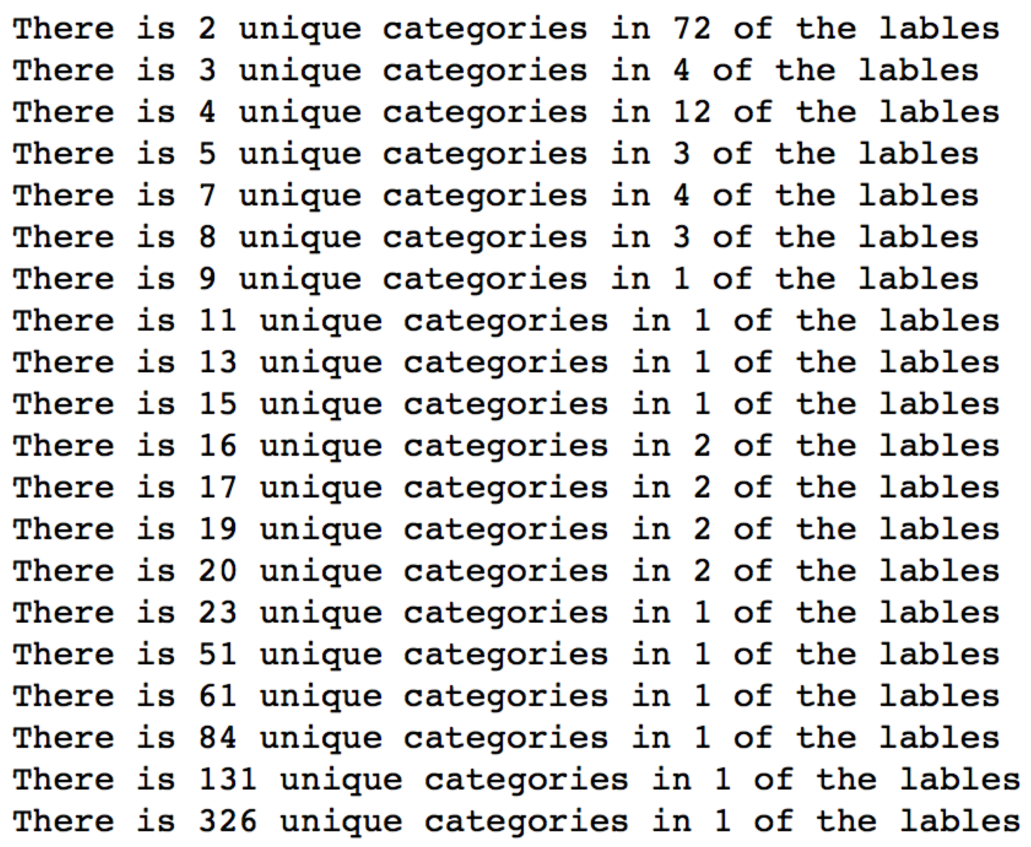
Loss data	Log Transformed Loss Data
	

Figure 6: "Loss" Logorithmic Transformation

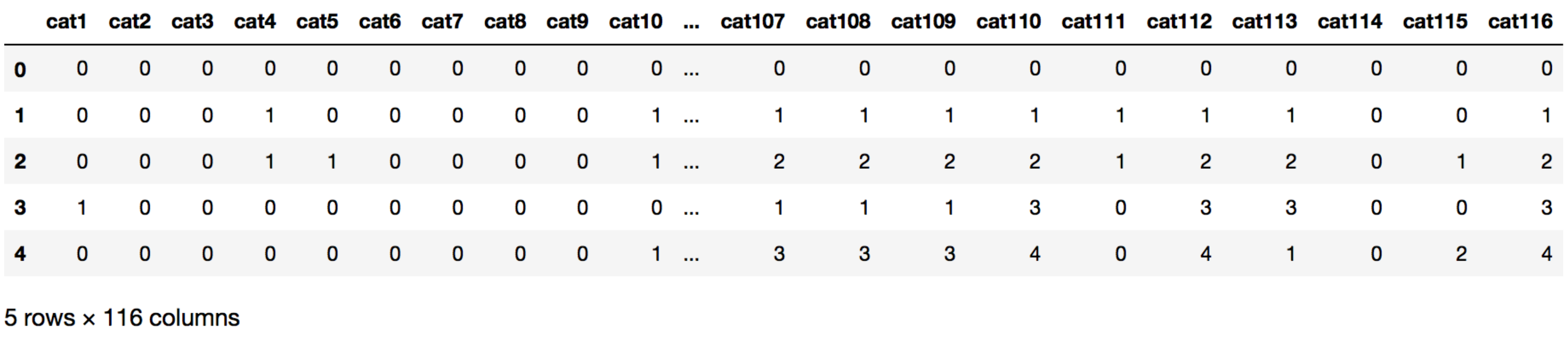
Shown in Figure 7 is the non-numerical data broken down into different components. The table below shows how many unique variables are in each column of non-numeric data.



```
There is 2 unique categories in 72 of the lables
There is 3 unique categories in 4 of the lables
There is 4 unique categories in 12 of the lables
There is 5 unique categories in 3 of the lables
There is 7 unique categories in 4 of the lables
There is 8 unique categories in 3 of the lables
There is 9 unique categories in 1 of the lables
There is 11 unique categories in 1 of the lables
There is 13 unique categories in 1 of the lables
There is 15 unique categories in 1 of the lables
There is 16 unique categories in 2 of the lables
There is 17 unique categories in 2 of the lables
There is 19 unique categories in 2 of the lables
There is 20 unique categories in 2 of the lables
There is 23 unique categories in 1 of the lables
There is 51 unique categories in 1 of the lables
There is 61 unique categories in 1 of the lables
There is 84 unique categories in 1 of the lables
There is 131 unique categories in 1 of the lables
There is 326 unique categories in 1 of the lables
```

Figure 7: Number of Unique Non-Numerical Values

Typically learning algorithms expect input to be numeric so all non-numeric data must be converted into numeric data. There are many ways to do so but for this project we will be using the pandas factorize operation. This method encodes the input values by assigning a unique number to each value. One hotkey encoding code have also been used but it produces allot of parameters that drastically slow down the learning algorithm

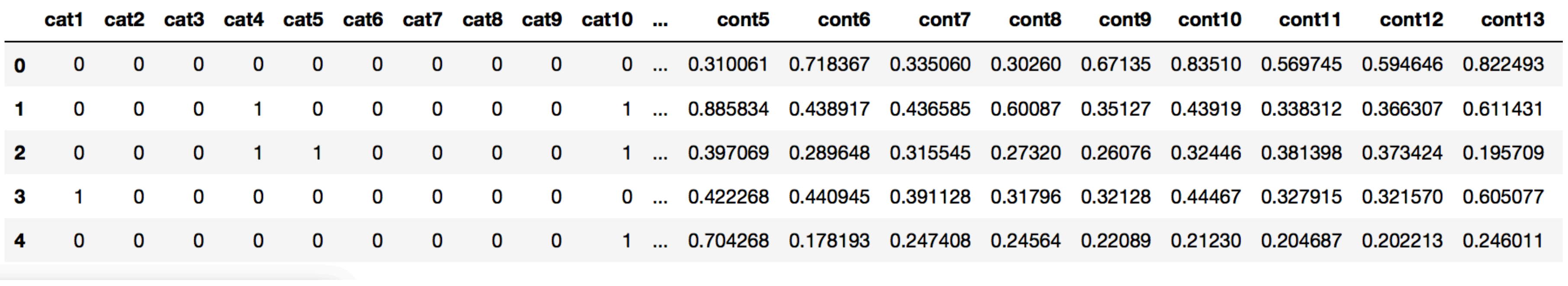


	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	cat10	...	cat107	cat108	cat109	cat110	cat111	cat112	cat113	cat114	cat115	cat116
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	1	...	1	1	1	1	1	1	1	0	0	1
2	0	0	0	1	1	0	0	0	0	1	...	2	2	2	2	1	2	2	0	1	2
3	1	0	0	0	0	0	0	0	0	0	...	1	1	1	3	0	3	3	0	0	3
4	0	0	0	0	0	0	0	0	0	1	...	3	3	3	4	0	4	1	0	2	4

5 rows × 116 columns

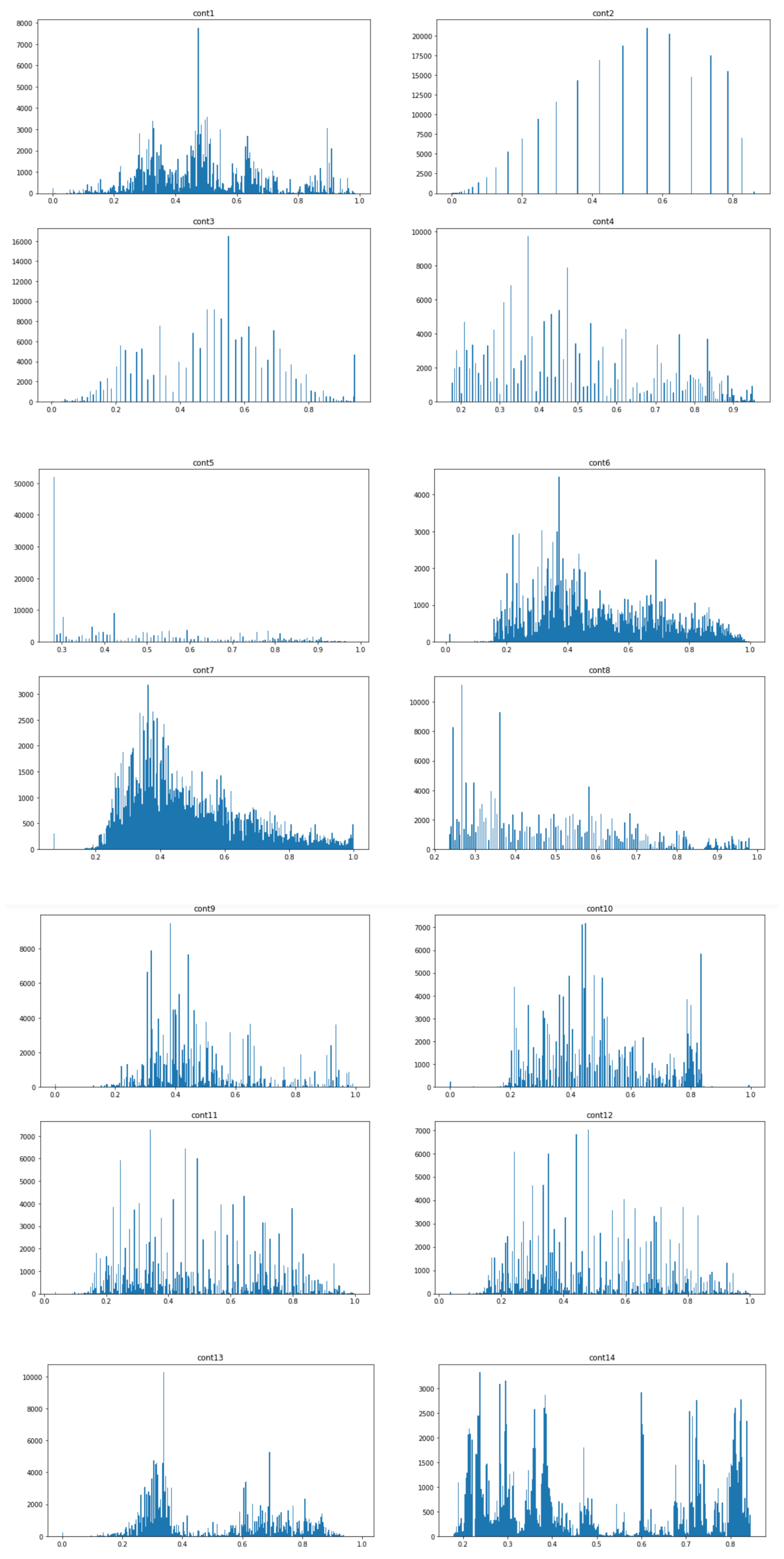
Figure 8: Non-nemeric Data Coverted to Numeric Data

The data has been clearly defined and pre-processed. In Figure 9 is the end result that we need in order to start training out learning algorithm.



	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	cat10	...	cont5	cont6	cont7	cont8	cont9	cont10	cont11	cont12	cont13
0	0	0	0	0	0	0	0	0	0	0	...	0.310061	0.718367	0.335060	0.30260	0.67135	0.83510	0.569745	0.594646	0.822493
1	0	0	0	1	0	0	0	0	0	1	...	0.885834	0.438917	0.436585	0.60087	0.35127	0.43919	0.338312	0.366307	0.611431
2	0	0	0	1	1	0	0	0	0	1	...	0.397069	0.289648	0.315545	0.27320	0.26076	0.32446	0.381398	0.373424	0.195709
3	1	0	0	0	0	0	0	0	0	0	...	0.422268	0.440945	0.391128	0.31796	0.32128	0.44467	0.327915	0.321570	0.605077
4	0	0	0	0	0	0	0	0	0	1	...	0.704268	0.178193	0.247408	0.24564	0.22089	0.21230	0.204687	0.202213	0.246011

Figure 9: Training Data After Cleaning



Data Preprocessing

Before the data can be feed into the any learning algorithm it must be separated into a training and testing sets. Splitting the data and randomizing it removes any hidden bias or variances. This is because the model needs to be tested on unseen data. The mean absolute error of the testing model is then compared to the training model to check for over fitting or under fitting.

The data is slit into a training and testing data sets.

- Training set has 141238 samples.
- Testing set has 47080 samples.

Algorithms and Techniques

Three different repressors will be taken into consideration linear regression, random forest regression, and extreme gradient regression. The mean absolute error of a un-tuned Linear regression model will be used as our bench mark that we need to beat.

Linear Regression

- Linear Regression is used to model the relationship between two variables
- It does this by plotting a line and minimizing the distance between that line and the data as show in Figure 10
- It great starting point when analyzing data because of its simplicity
- The error increases when trying to apply this to non linear data

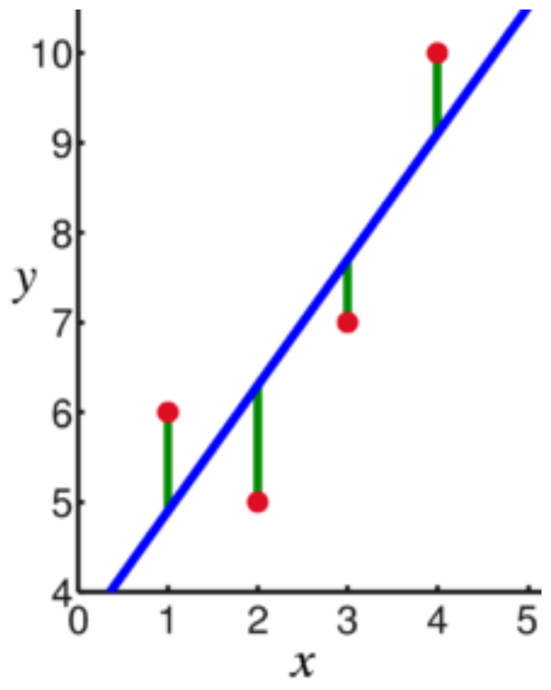
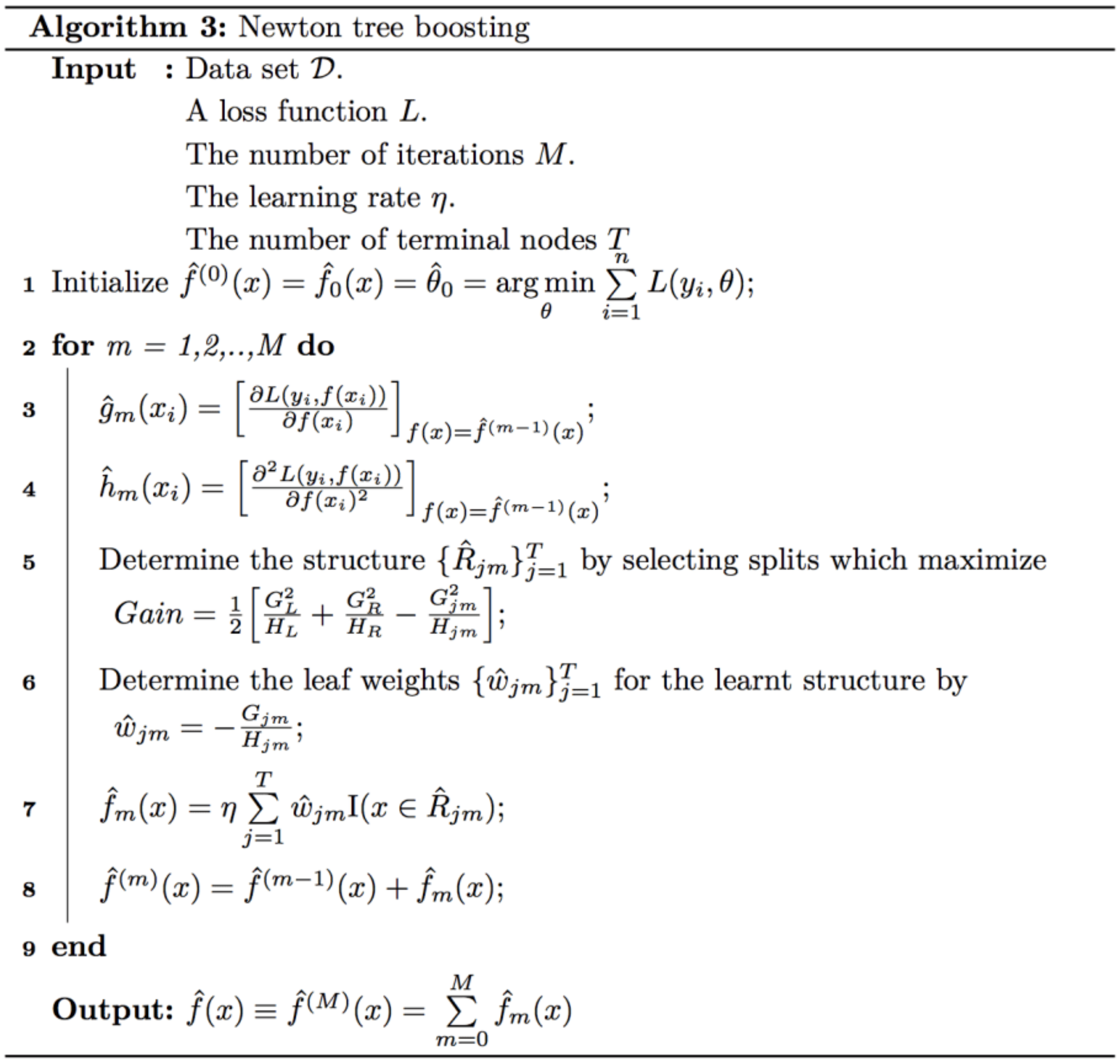


Figure 10: Linear Regression Example

XGBoost (Extreame Gradient Decent)

- XGboosting stands for extreme gradient boosting algorithm and is part of a group of learning algorithms called ensemble methods.
  - There are three types of ensemble methods and XGBoosting is a part of the boosting class of ensemble methods.
  - XGBoosting runs multiple decision tree algorithms and each of which learns to fix the prediction errors of a prior model in the chain.
  - This can be compared to how a basketball player shoots a basketball. If the basketball player misses the shot, then the next shot the basketball player takes is corrected based on how far off his first shot was from the goal. The basketball player will continue to adjust this technique until he makes perfect shots.
  - Below in **Figure 11** is the Newton Tree Boosting algorithm used by XGBoosting
  - The main advantage of XGBoosting is the executiong speed and accuracy
  - A disadvantage is that it is prone to over fitting

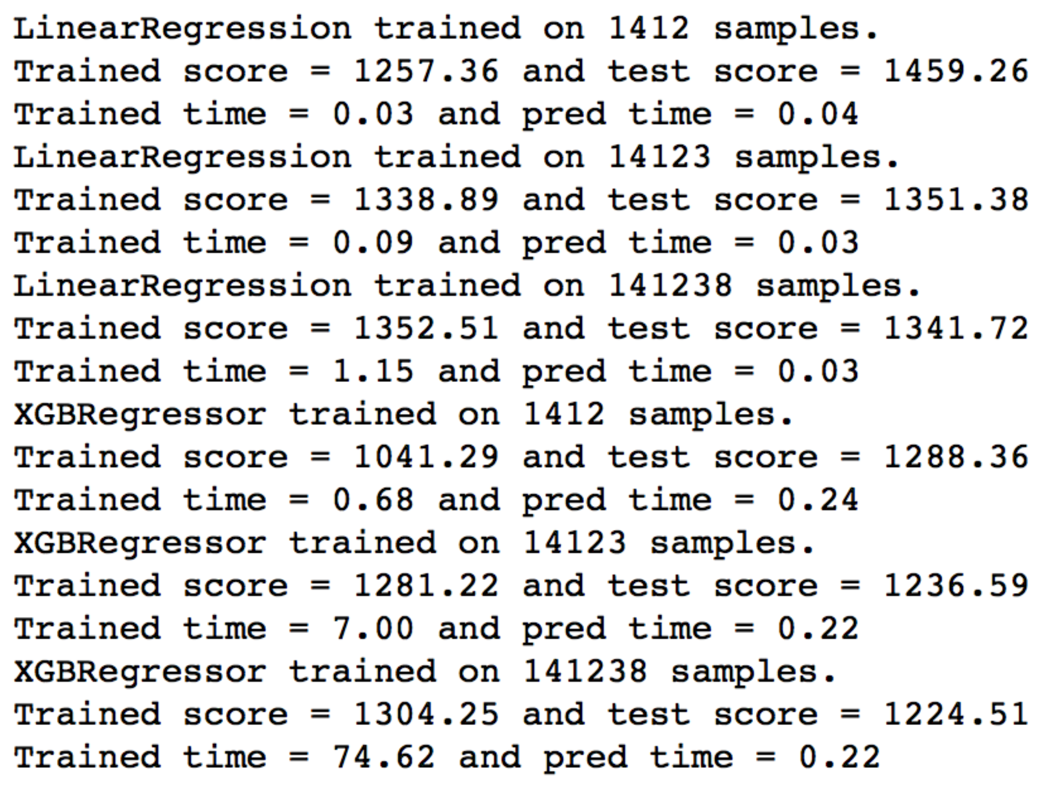


```
Algorithm 3: Newton tree boosting
   Input  : Data set $\mathcal{D}$.
            A loss function $L$.
            The number of iterations $M$.
            The learning rate $\eta$.
            The number of terminal nodes $T$
1  Initialize $\hat{f}^{(0)}(x) = \hat{f}_0(x) = \hat{\theta}_0 = \arg\min_{\theta} \sum_{i=1}^{n} L(y_i, \theta)$;
2  for $m$ = 1,2,..,$M$ do
3      $\hat{g}_m(x_i) = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=\hat{f}^{(m-1)}(x)}$;
4      $\hat{h}_m(x_i) = \left[\frac{\partial^2 L(y_i, f(x_i))}{\partial f(x_i)^2}\right]_{f(x)=\hat{f}^{(m-1)}(x)}$;
5      Determine the structure $\{\hat{R}_{jm}\}_{j=1}^{T}$ by selecting splits which maximize
         $Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L} + \frac{G_R^2}{H_R} - \frac{G_{jm}^2}{H_{jm}}\right]$;
6      Determine the leaf weights $\{\hat{w}_{jm}\}_{j=1}^{T}$ for the learnt structure by
         $\hat{w}_{jm} = -\frac{G_{jm}}{H_{jm}}$;
7      $\hat{f}_m(x) = \eta \sum_{j=1}^{T} \hat{w}_{jm} \mathrm{I}(x \in \hat{R}_{jm})$;
8      $\hat{f}^{(m)}(x) = \hat{f}^{(m-1)}(x) + \hat{f}_m(x)$;
9  end
   Output: $\hat{f}(x) \equiv \hat{f}^{(M)}(x) = \sum_{m=0}^{M} \hat{f}_m(x)$
```

Figure 11: Newton Tree Boosting

Benchmark

The data in training on an un-tuned Linear Regression and XGBoost Model to set our baseline. The benchmark for this project is set by the un-tuned Linear Regression model as a basis for determining how well our XGBoost Model performs. Mean Absolute Error is the metric that will be used for determining how good our models perform. The lower the mean absolute error is the better our model is performing. The train.csv file has a total of 188,318 data points and this is split up in to 25% training and 74% testing. In **Figure 11** below an un-tuned linear regression and XGB regression algorithm area applied to 1%, 10%, an 100% of the data to see how the error changes as you add more data. The lowest mean absolute error for using Linear Regression is 1341.72 and that will be the benchmark for the XGBoost model.



```
LinearRegression trained on 1412 samples.
Trained score = 1257.36 and test score = 1459.26
Trained time = 0.03 and pred time = 0.04
LinearRegression trained on 14123 samples.
Trained score = 1338.89 and test score = 1351.38
Trained time = 0.09 and pred time = 0.03
LinearRegression trained on 141238 samples.
Trained score = 1352.51 and test score = 1341.72
Trained time = 1.15 and pred time = 0.03
XGBRegressor trained on 1412 samples.
Trained score = 1041.29 and test score = 1288.36
Trained time = 0.68 and pred time = 0.24
XGBRegressor trained on 14123 samples.
Trained score = 1281.22 and test score = 1236.59
Trained time = 7.00 and pred time = 0.22
XGBRegressor trained on 141238 samples.
Trained score = 1304.25 and test score = 1224.51
Trained time = 74.62 and pred time = 0.22
```

Figure 12: Linear Regression and XGBoost Results

III. Methodology

Implementation and Refinement

Initially the XGBoost algorithm was implement that was in the form of an numpy array. Early on training and predicting the models was very time consuming taking up to and hour to train what I thought was simple model. This made me realize that I was more than likely doing something wrong. I went back through the XGBoost documentation to make sure that I understood exactly how the algorithm worked. Through this I found out that XGBoost does not work efficiently with numpy arrays and instead prefers to use a Dmatrix. The Dmatrix is a data structure that is specifically used by XGBoost in order to optimize memory and training speed.

Running the XGBoost algorithm using a linear objective function and with 50 boosting rounds results in a mean absolute error of 1162. This will used as the base model because all parameters were set to their default values. In order to optimize the algorithm the following parameters will be changed one at a time until the optimal combination of parameters is found.

- eta - learning rate of the model
- max_depth - This is the maximum depth of the tree. A low number will probably not capture nonlinear features and a high number makes the model more complex.
- min_child_weight - This parameter prevents under fitting

The data exploration section shows and explains how the data was preprocessed. Now the next step to separate the data in to training and testing sets. After which we can begin to train the algorithm but before anything can be done the correct libraries must be imported.

They are:

- from sklearn.cross_validation import train_test_split - this will allow the data to be split into tranining and testing sets
- from sklearn.metrics import mean_absolute_error - this will calculate mean absolute error
- from sklearn.metrics import make_scorer - allows means absolute error to be used with XGboost
- from time import time - this will import a clock to keep track of time
- import xgboost as xgb - this will import extreme gradient boosting and all functions associated with it

Training the Base Model

The follow step were taken to train the base model:

1. The data is separated into a training and data sets with the code below.
    - X_train, X_test, y_train, y_test = train_test_split(cat_cont_data, loss_log_transformed, test_size = 0.25)
2. The data has to then be converted into a DMatrix because it will allow XGBoost to run more efficiently
    - dtrain = xgb.DMatrix(data=X_train, label=y_train)
    - dtest = xgb.DMatrix(data=X_test, label=y_test)
3. The parameters dictionary has to be set will all relevant parameters. For this model the only parameter that will be use is linear regression.
    - xgb_params = {'objective': 'reg:linear',}
4. The algorithem is now trained.
    - bst_cv1 = xgb.train(params=xgb_params, dtrain=dtrain, num_boost_round=50)
5. The testing data can now be predicted
    - predictions_test = bst_cv1.predict(dtest)
6. The actual "loss" values for the testing data are know so they are pulled to compare to the predicted values
    - y_te = dtest.get_label()
7. Because the "loss" data was transformed using the log function they have to be converted back using the expoenetial function
    - np.exp(y_te)
    - np.exp(predictions_test)
8. The mean aboslute error can now be calculated
    - MAE = mean_absolute_error(np.exp(y_te), np.exp(predictions_test))
    - MAE = 1162

The following steps were taken to tune the algorithm.

For each of the follow sets of parameters a for loop was created to train each combination of parameters. After each set an optimal parameter was found and added to the next set of parameters.

1. 'eta': [0.2, 0.3, 0.4, 0.5, 0.6] was trained with num_boost_round = 50

   - optimal param = 'eta': 0.3
   - MAE = 1162
2. 'eta': 0.3, 'max_depth': [6,8,10,12,14] , 'child_weight': [4,6,8,10,12], 'num_boost_round': 50

   - optimal param = eta': 0.3, 'max_depth': 6 , 'child_weight': 8, 'num_boost_round': 50
   - MAE = 1160
3. 'eta': [0.05, 0.1, 0.2, 0.3], 'max_depth': 6 , 'child_weight': 8, 'num_boost_round': 100

   - optimal param = eta': 0.2, 'max_depth': 6 , 'child_weight': 8, 'num_boost_round': 100
   - MAE = 1152
4. 'eta': [0.05, 0.1, 0.2, 0.3], 'max_depth': 6 , 'child_weight': 8, 'num_boost_round': 200

   - optimal param = eta': 0.1, 'max_depth': 6 , 'child_weight': 8, 'num_boost_round': 200
   - MAE = 1147

Below is a basic example of how the learning algorithm was trained.

In [ ]:

```
# This dictionary contains all parameters that will be used
xgb_params = {'objective': 'reg:linear'}

# The scoring function is assigned
scorer = make_scorer(mean_absolute_error)


results = {}
start = time()

# The algorithm is trained with the given parameters
bst_cv1 = xgb.train(params=xgb_params, dtrain=dtrain, num_boost_round=50,feval=s
corer)
end = time()
results['train_time'] = end - start

start = time()
# predictions are made on the training and testing data
predictions_train = bst_cv1.predict(dtrain)
predictions_test = bst_cv1.predict(dtest)
end = time()
results['pred_time'] = end - start

# The predictions are pulled from the Dmatrix
y_tr = dtrain.get_label()
y_te = dtest.get_label()


# The results are first normalized because they are in exponential form.
# Then the mean absolute error can be calculated
results['acc_train'] = mean_absolute_error(np.exp(y_tr), np.exp(predictions_trai
n))
results['acc_test'] = mean_absolute_error(np.exp(y_te), np.exp(predictions_test)
)

# The code below prints on the mean absolute error and time it took to train and
prediction the model
print "params {}".format(xgb_params)
print "Trained score = {:.2f} and test score = {:.2f}".format(results['acc_train
'],results['acc_test'])
print "Trained time = {:.2f} and pred time = {:.2f}".format(results['train_time'
],results['pred_time'])
```

IV. Results

Model Evaluation and Validation

After fine tuning the model using various parameters the best answer was achieved using the following parameters.

---

xgb_params = {'eta': 0.1, 'objective': 'reg:linear', 'max_depth': 6, 'min_child_weight': 8,}

Using these parameters, a Mean Absolute Error of 1147.20 was found.

A quick sensitivity analysis was performed on the algorithm using these optimal parameters to see how the model reacts to a different set of data. The data set was split using a random state of 5 and 10.

- Original split: mae = 1147.20
- Random state 5: mae = 1146.85
- Random state 10: mae = 1151.03

There is a difference of about 3.83 between the highest and lowest MAE score. This is due to the fact that all three of these models were trained and tested on different combinations of the same data set. Some slight variations are expected but for the purpose of this project the variation in the sets are acceptable.

Justification

Since this is a Kaggle competition it is difficult to say whether the problem has been solved or not. The only thing we can go by is by the benchmark that was set and by comparing with other Kaggle scores. The final model outperformed the benchmark score of 1334. Currently the top Kaggle score is 1096 which is 189 points lower than what was calculated here. The model still needs work but for the purpose of this project it has met all expectation.

V. Conclusion

Visualization

Figure 13 shows the feature importance graph produced by the extream gredient boostin model. This chart shows the F_score for each feature in the data set. The F_core is calculated by taking the average number of times the feature is used to split a tree that improves the perforance measure. The features with the highest F_score have the most impact on the model. It is interesting to see that out of all 130 feature there is only a handfull of features that have a significant impact on the model.

Figure 13: Feature Importance

Reflection

- In this project a data set was taken and processed in order to make it usable for the learning algorithm. Two different types of learning algorithms were used in this project, linear regression and XGBoost. An un-tuned linear regression model was used as a base model to compare our XGBoost model to. An un-tuned XGBoost model had a mean absolute error of 1178 but by tuning various parameters the mean absolute error was brought down to 1153.
- This model was very complex in the sense that it had many parameters and allot of data points. This posed many challenges and limitations on obtaining a better solution. The fact that the data set was so large made running models very time consuming.
- This was a very interesting and eye-opening project and motivates me to learn more about data science and machine learning. I have a couple of friends that work for large corporations like Walmart and AT&T in the automation and data science departments. Their job is find area of work were automation can be implemented in order reduce cost. In some cases, it can turn into a tense situation because people are losing their jobs, but this is the way forward.

Improvement

There is always more work to be done but running these models is very time consuming. Another option might be to use cloud computing to speed up the process but that is out of the scope of this project. One option that I did not consider until now is finding the features that are most important and focusing more on those. This might help eliminate noise cause by features with little relevance. I used XGBoost because it was simple to understand but maybe in the future I will try to solve this problem using a Convolution Neural Network.

References

1. Allstate Claims Severity, Kaggle

    - https://www.kaggle.com/c/allstate-claims-severity (https://www.kaggle.com/c/allstate-claims-severity)
    - https://www.kaggle.com/c/allstate-claims-severity/data (https://www.kaggle.com/c/allstate-claims-severity/data)

2. Regression

    - https://machinelearningmastery.com/ensemble-machine-learning-algorithms-python-scikit-learn/ (https://machinelearningmastery.com/ensemble-machine-learning-algorithms-python-scikit-learn/)
    - https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/ (https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/)

3. Allstate Insurance

    - https://en.wikipedia.org/wiki/Allstate (https://en.wikipedia.org/wiki/Allstate)