# Machine Learning Engineer Nanodegree
## Capstone Project

Allstate Claims Severity - Kaggle Contest

Siddartha Tondapu December 10, 2017

# I. Definition

### Project Overview

Automobile accidents are involuntary and are often unexpected. Nearly 1.3 million people in the world and 37,000 people in the United States alone lose their life in these tragic accidents. While these are major accidents, there are many minor accidents and fender benders that cause devastating psychological damages. During these times, the victim wants to focus more time on his family, friends, and loved ones than to deal with insurance claims. Unfortunately, I was involved in a small fender bender recently, and I felt the whole process of claiming for insurance to be long and tedious. With so many accidents happening every day, I realized there can be better ways to make the whole process quick and seamless. I am using the following dataset from Allstate Kaggle challenge: https://www.kaggle.com/c/allstate-claims-severity/data.

### Problem Statement

Allstate, an insurance company, saw this as a bottleneck and decided to automate the whole process by predicting the cost and severity of the claim. Our goal in this project is to use machine learning algorithms to model the cost and severity of a claim. We have been given some input variables (both continuous and discrete/categorical). These data points are anonymous i.e. label names were pseudo-randomly generated. Using these input variables, we will be predicting the output cost of the claim by using a regression model.

### Metrics

Using the feature labels provided, we will use a regression model to predict the outcome, which in this case is the cost associated to a claim. Our model will be used to predict the cost of testing data. The predicted results with be compared with the actual results using mean absolute error (MAE). We chose MAE, because it doesn’t penalize harshly for large errors. Other error calculations like R^2, RMSE, and MSE usually square the error, thereby penalizing harshly for larger errors. Another reason we chose MAE, is because the Kaggle project uses this as baseline metric for measuring its accuracy, and

I wanted to stay consistent. MAE is calculated by subtracting predicted error from actual error, find the absolute value, and eventually the mean/average of these numbers. The average MAE of Kaggle submissions after removing few outliers was 1222.24.

```
# Import required modules
import numpy as np
import pandas as pd
from IPython.display import display # Allows the use of display() for DataFrames

# Pretty display for notebooks
%matplotlib inline
# Find the Baseline Score that we compare with our model results
base_model = pd.read_csv('Data/allstate-claims-severity-publicleaderboard.csv')

# Drop Scores greater than 5000
base_model = base_model.drop(base_model[base_model.Score > 5000].index)

# Mean of scores
print (base_model['Score'].mean())
1222.23689139
```

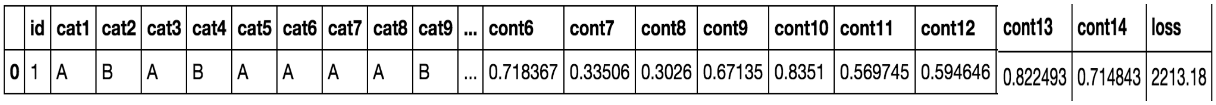
# II. Analysis

## Data Exploration

The data was provided by Kaggle as part of their problem in train.csv.  Lets analyze this data a little more.

```
# Load the Boston housing dataset
data = pd.read_csv('Data/train.csv')
loss = data['loss']
features = data.drop(['id','loss'], axis = 1)

print "Allstate dataset has {} data points with {} variables each.".format(*data.shape)

# Success - Display the first record
display(data.head(n=1))
Allstate dataset has 188318 data points with 132 variables each.
```

1 rows × 132 columns



| | 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 | 0.33506 | 0.3026 | 0.67135 | 0.8351 | 0.569745 | 0.594646 | 0.822493 | 0.714843 | 2213.18 |

Allstate dataset has 188318 data points with 132 variables or columns. Column[1] is id, column[2-117] are categorical (116), column[118-131] are continuous (16), and column[ 132] is also continuous and corresponds to the loss or cost of a claim. As we can see, id and loss are the only columns with meaningful title. Since Id is not required, we will remove it from our dataset. Similarly, we are using features to predict the 'loss' variable or the target. Hence, we are removing this from the dataset and storing it in different variable called loss.

Let's analyze the loss column a little more, as our model will be predicting it.

```
Minimum price: $0.67
Maximum price: $121,012.25
Mean price: $3,037.34
Median price $2,115.57
Standard deviation of prices: $2,904.08
```

Based on the statistics above, we can see that loss variable is very skewed. In addition, the min and max values are very far apart from the mean, suggesting many outliers and high standard deviation. In the later section, we will remove these outliers. In addition, the lets find the log of loss to see if it normalizes the data.

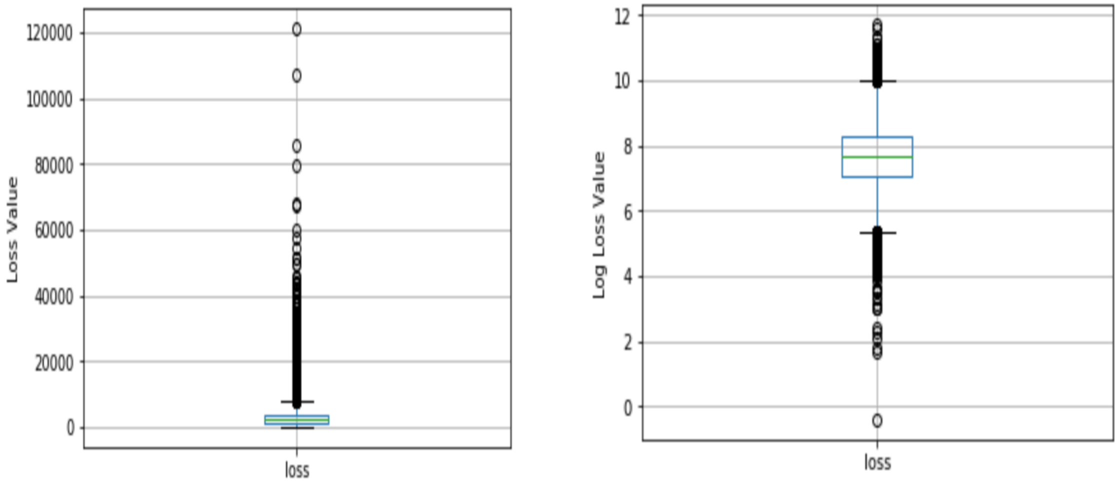


Figure on the left is a boxplot of loss variable and we can see that the data is positively skewed. Figure on the right is a boxplot of log loss and we can observe that the data is normally distributed now.

### Exploratory Visualization

Lets see if there are any relationship between continuous features in our dataset.

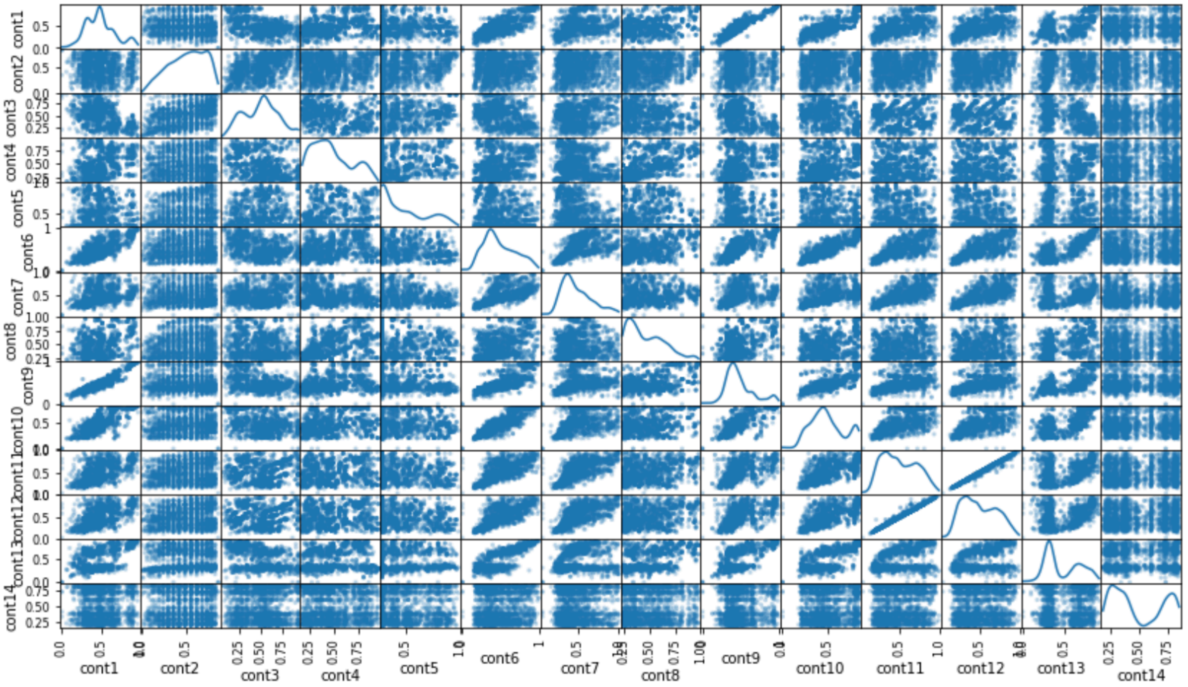


Figure 1

Figure 1, Scatter plot of the continuous features above shows a strong linear correlation between these features: (cont11 and cont12), (cont1 and cont9), (cont6 and cont10). In other words, we can predict the value of cont12 from cont11 and vice versa. Similarly, let's look at (cont1 and cont14), the values are randomly distributed and there is no correlation. This means one cannot predict the value of cont1 from cont14 and vice versa. If two features have strong correlation, at least one of them can be removed, thereby decreasing the computation time to train. However, in this project, I have decided not to remove any points, because my focus is on accuracy, and not performance.

## Algorithms and Techniques

We are using features or independent variables to determine the dependent variable or the target. This is a perfect example of Supervised regression problem. Firstly, we will split the data into two sections: Testing and Training. We will train our data with three different models listed below, and use the most optimum model to predict the target variable cost on the testing data. This will help us generalize our model on un-seen data.

*Linear Regression*

- Application: Predicting the cost of house based on its features.
- Strengths: Extremely fast and has low variance. We donot have to worry about correlation between independent features. Handlers outliers well and doesn't overfit easily.
- Weaknesses: Not good for non-linear relationships and results in high bias for complex models.
- Reason for choosing: Data seems to be linearly separable. Efficient to compute. Doesn't over fit easily.
- Summary: Linear Regression uses the formulate y = mx + b, where y is the outcome we are predicting. m is the slope or weights we use to reduce the error, x are the inputs or independent variables as vector and b is a constant. The formula draws a line for (two variables), plane (for three variables), or a hyperplane for (three or more variables).

*Decision Trees*

- Application: Decision Trees are used in Data Mining.
- Strengths: Easy to visualize data for smaller levels. Data doesn't have to be linearly separable and deals well with outliers.
- Weaknesses: Can over fit very easily on complex data and also generalizes very poorly on testing data.
- Reason for choosing: Since most of the features are categorical and encoded as binary numbers (0 or 1) using one-hot, Decision Tree algorithm should shine in this case. It can also be used to visualize the data.
- Summary: Lets use figure 2 below to explain Decision Trees. In the below example, our decision tree model predicts if a person is fit. The output of our model is a Boolean Yes for fit and No for unfit. The inputs of our models are features like ‘eats lot of pizzas’ or ‘exercise in the morning’. Based on the values of these inputs, we can predict if a person is fit or not by traversing the tree from top to bottom.

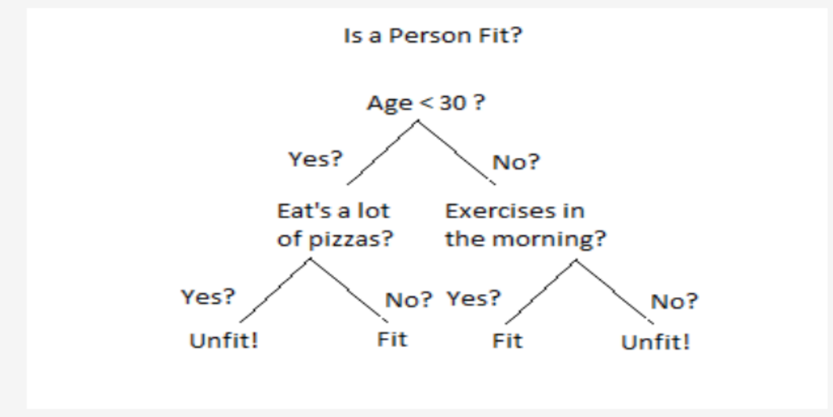


Figure 2

XGBoost

- Application: Classifying proteins, text classifying.
- Strengths: Can model complex and non linear relationships and not susceptible to noise.
- Weaknesses: Slow and computation heavy. Requires large CPU and memory. Can over fit.
- Reason for choosing: XGBoost shines when there are a lot of features which don't have to be linearly separable. It's also very accurate.
- Summary: XGBoost also known as extreme gradient boosting is derived from Gradient Boosting Algorithm. Fundamentally, it uses an ensemble of trees to predict the outcome.

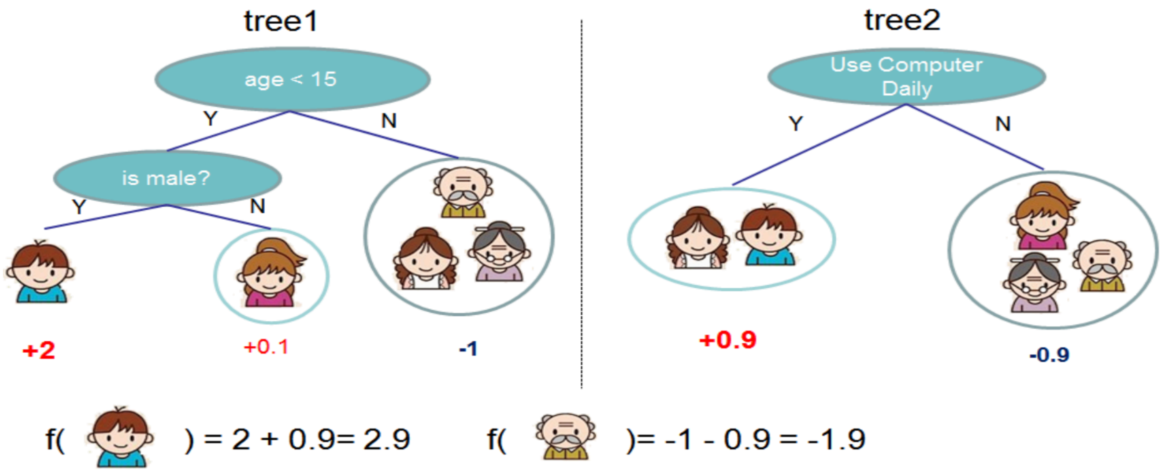


Figure 3

In figure3, we are predicting the likelihood of a person playing videogame and assigning a score. A greater score means more likely to play videogames. Tree1 predicts that the boy in blue shirt will play and assigned a score of +2, and tree2 predicted +0.9. By using ensemble of trees, we are increasing the likelihood of the boy playing video games and the newest value is 2.9.

## Benchmark Model

I will be using the average of results/MAE scores from Kaggle submissions. Since, there are few outliers that are very huge, we will remove them from our calculation of mean (where base_model_Score > 5000). Upon doing this calculation, the mean MAE score was 1222.24. I will use this score as a benchmark to compare our model's performance. This calculation can be found in metrics section above.

## III. Methodology¶

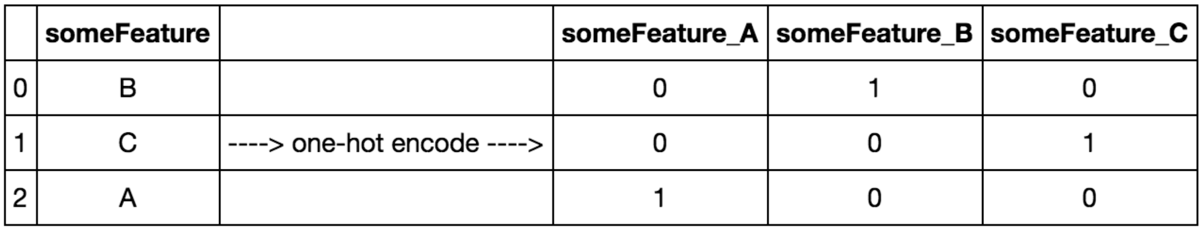
### Data Preprocessing¶

- Feature Scaling: Data is often not normally distributed (spread out). This is confirmed by the data we collected from Data Exploration section above for our target variable: loss. The difference between mean and median of scores is large indicating a large skew. As a result, we had to apply non-linear scaling for our loss variable. This is done above, and a new column is added to our data: log_loss.

- Outlier Detection: Outliers are data points which are far from the mean and median. They skew the data and often result in incorrect calculations/results. In order to avoid this, we will be applying Tukey's Method for identifying outliers, where the outlier step is calculated as 1.5 times the interquartile range (IQR). Any point outside this IQR is considered as an outlier. Upon doing this calculation, there were 521 outliers and the finally datasets had 187797 entries after removing outliers. These outliers not only increase time to train, but they will also skew the results due to abnormal values.

- In our prior calculations, we observed strong linear correlation between (cont11 and cont12), (cont1 and cont9), (cont6 and cont10). While removing these features will improve the performance, it might affect the accuracy of our model. As a result, I have decided to leave these columns/features.

- Our data contains 116 categorical features. Learning algorithms expect input to be numeric, and this means all categorical variables will have to be converted to numerical. This can be achieved by one-hot encoding scheme. One-hot encoding creates a "dummy" variable for each possible category of each non-numeric feature. For example, refer to the following illustration:



| | someFeature | | someFeature_A | someFeature_B | someFeature_C |
|---|---|---|---|---|---|
| 0 | B | | 0 | 1 | 0 |
| 1 | C | ----> one-hot encode ----> | 0 | 0 | 1 |
| 2 | A | | 1 | 0 | 0 |

- Let’s say a feature has three possible categorical value: A, B, or C. One-hot encoding will create three different columns where the value can be 1 or 0. Upon doing this calculation on 187797 data points, the columns have expanded from 130 (not including id and loss) to 1148. A subset of these include: ['cont1', 'cont2', 'cont3', 'cont4', 'cont5', 'cont6', 'cont7', 'cont8', 'cont9', 'cont10', 'cont11', 'cont12', 'cont13', 'cont14', 'cat1_A', 'cat1_B', 'cat2_A', 'cat2_B', 'cat3_A', 'cat3_B', 'cat4_A', 'cat4_B', 'cat5_A', 'cat5_B', 'cat6_A', 'cat6_B', 'cat7_A', 'cat7_B', 'cat8_A', 'cat8_B', 'cat9_A', 'cat9_B', 'cat10_A', 'cat10_B', 'cat11_A', 'cat11_B', 'cat12_A', 'cat12_B', 'cat13_A', 'cat13_B', 'cat14_A', 'cat14_B', 'cat15_A', 'cat15_B', 'cat16_A', 'cat16_B', 'cat17_A', 'cat17_B', 'cat18_A', 'cat18_B', 'cat19_A', 'cat19_B', 'cat20_A', 'cat20_B', 'cat21_A', 'cat21_B', 'cat22_A', ...]

- Our last step, one-hot encoding increased number of feature or columns. While this is needed by our algorithm, it will drastically increase the complexity and time to solve or problem. Principal Component Analysis (PCA) calculates the dimension which best maximizes variance in our data. PCA will explain the variance ratio of each dimension. In our calculation, we will set the variable of n_components to 250. The first 250 principal components explained 97% (.97 * 100) variance in the data. This number 97% indicates the confidence in our reduced data and this is stored in a variable called features_good_encoded_reduced.

- Our final step of processing the data involves splitting the data into training and testing set. This can be achieved by using sklearn's cross_validation.train_test_split module. We will use 80/20 ration to split our data, where 80 is the training data and 20 is the testing data. Upon doing this calculation, training set had 150237 samples and testing set had 37560 samples. Training data will be use train our model, while testing data will be used to predict the accuracy our model by predicting the cost of a claim on data that it never saw before.

- All of the code can be found in report.ipynb. Each of the code blocks can be run independently to reproduce my work. It is required that the host machine has Python 2.7, sklearn, and xgboost python packages installed. Sklearn and xgboost can be installed using python package installer pip.

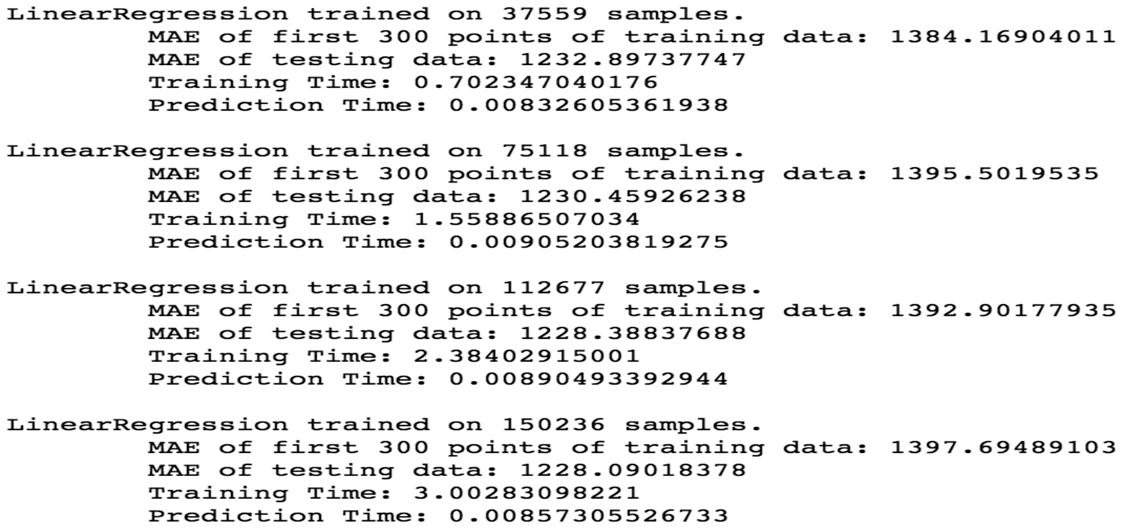
- Challenges faced during development.
    - GridSearch for XGboost took an extremely long time to train. As a result, I saved the model in a pikle file: xgboost.pkl
    - Most of the data labels had no meaning. Cont1 and Cat10 names were very ambiguous and did not reveal anything about the data.

**Implementation**

In this section, we will use three different models: Linear Regression, Decision Trees, and XGBRegressor to identify the best model. We will compare these results with the mean average MAE score of submissions calculated above: 1222.24. For each model, we will calculate the MAE of first 300 points of training data, MAE of testing data, Training time, and prediction time (which includes first 300 points of training data and testing data).

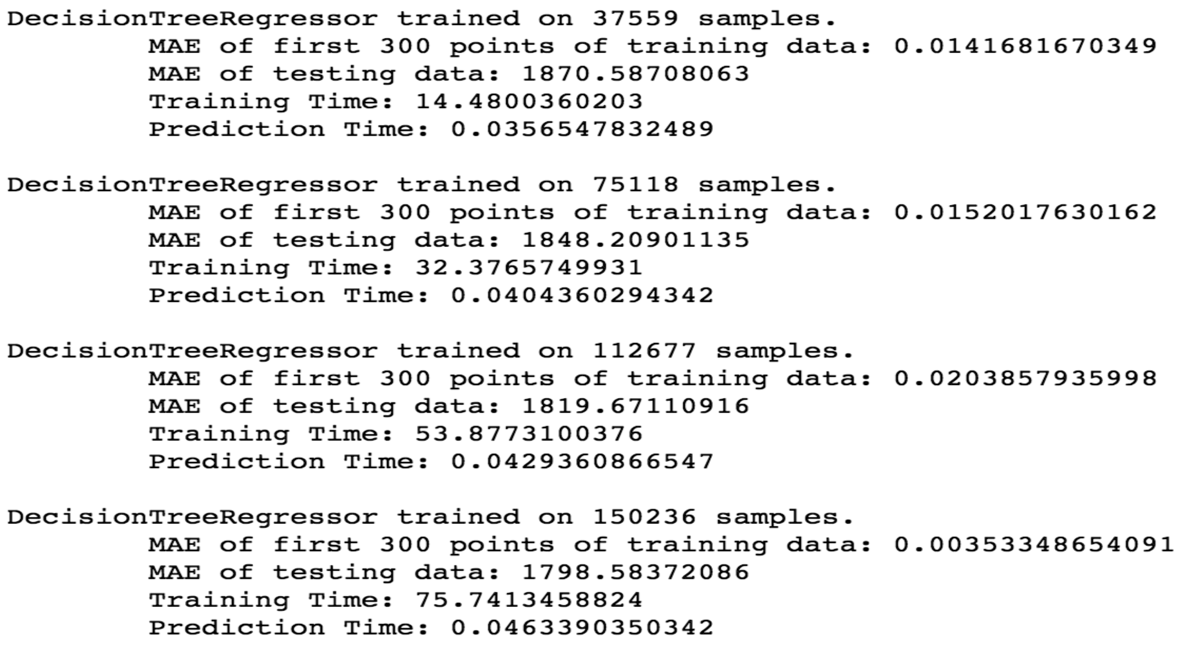
Note: Models in this section are using default parameters. We will tune them in the later section using GridSearch.

- Linear Regression: sklearn.linear_model.LinearRegression



```
LinearRegression trained on 37559 samples.
        MAE of first 300 points of training data: 1384.16904011
        MAE of testing data: 1232.89737747
        Training Time: 0.702347040176
        Prediction Time: 0.00832605361938

LinearRegression trained on 75118 samples.
        MAE of first 300 points of training data: 1395.5019535
        MAE of testing data: 1230.45926238
        Training Time: 1.55886507034
        Prediction Time: 0.00905203819275

LinearRegression trained on 112677 samples.
        MAE of first 300 points of training data: 1392.90177935
        MAE of testing data: 1228.38837688
        Training Time: 2.38402915001
        Prediction Time: 0.00890493392944

LinearRegression trained on 150236 samples.
        MAE of first 300 points of training data: 1397.69489103
        MAE of testing data: 1228.09018378
        Training Time: 3.00283098221
        Prediction Time: 0.00857305526733
```

- Decision Tree: DecisionTreeRegressor

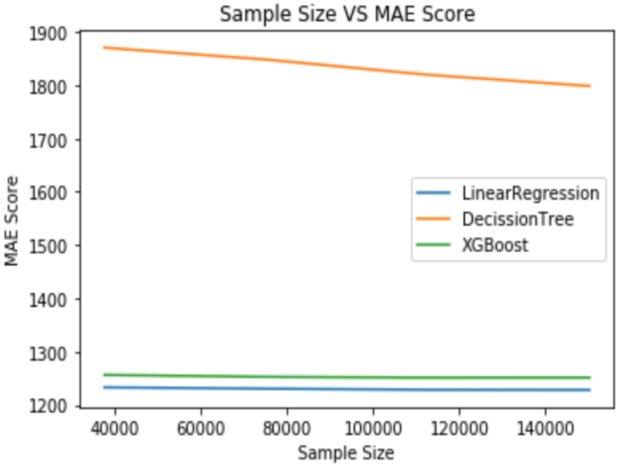


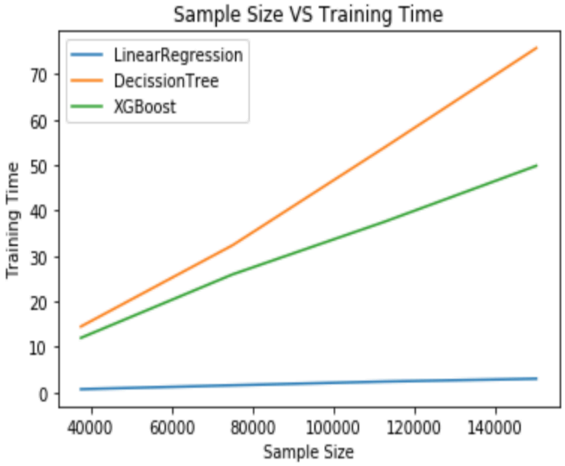
```
DecisionTreeRegressor trained on 37559 samples.
        MAE of first 300 points of training data: 0.0141681670349
        MAE of testing data: 1870.58708063
        Training Time: 14.4800360203
        Prediction Time: 0.0356547832489

DecisionTreeRegressor trained on 75118 samples.
        MAE of first 300 points of training data: 0.0152017630162
        MAE of testing data: 1848.20901135
        Training Time: 32.3765749931
        Prediction Time: 0.0404360294342

DecisionTreeRegressor trained on 112677 samples.
        MAE of first 300 points of training data: 0.0203857935998
        MAE of testing data: 1819.67110916
        Training Time: 53.8773100376
        Prediction Time: 0.0429360866547

DecisionTreeRegressor trained on 150236 samples.
        MAE of first 300 points of training data: 0.00353348654091
        MAE of testing data: 1798.58372086
        Training Time: 75.7413458824
        Prediction Time: 0.0463390350342
```

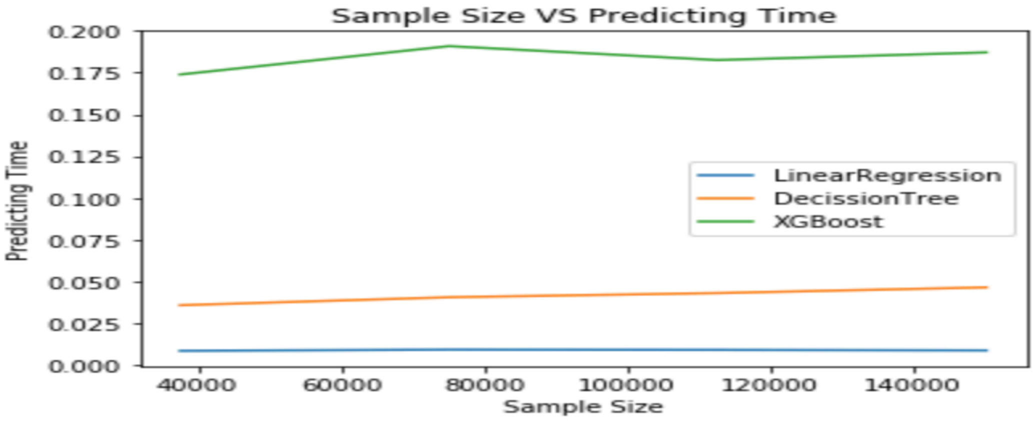
- XGBoost: xgboost.XGBRegressor

```
XGBRegressor trained on 37559 samples.
        MAE of first 300 points of training data: 1323.17621514
        MAE of testing data: 1256.21373718
        Training Time: 11.9831569195
        Prediction Time: 0.173917770386

XGBRegressor trained on 75118 samples.
        MAE of first 300 points of training data: 1340.85318756
        MAE of testing data: 1252.67059707
        Training Time: 25.9640898705
        Prediction Time: 0.19090294838

XGBRegressor trained on 112677 samples.
        MAE of first 300 points of training data: 1340.22588551
        MAE of testing data: 1251.0783475
        Training Time: 37.5314948559
        Prediction Time: 0.182554006577

XGBRegressor trained on 150236 samples.
        MAE of first 300 points of training data: 1352.30548597
        MAE of testing data: 1251.25979978
        Training Time: 49.8439881802
        Prediction Time: 0.187129020691
```

Looking at these numbers, we can quickly come to a consensus that DecissionTreeRegressor is slower and less accurate than LinearRegressor and XGBRegressor. While XGBRegressor was more accurate than Linear Regressor, it also took longer. Lets analyze this in graphs below.







**Analysis + Trends**

- Decision Tree Regression had highest MAE score (worst accuracy) than Linear Regression and XGBoost
- Decision Tree also took the most time to train the data, followed by XGBoost, and Linear Regression.
- XGBoost took the most time to predict the test data, followed by Decision Tree, and Linear Regression.
- As the number of samples increased, the MAE Score decreased indicating our algorithm is not underfitting.
- As the number of samples increased, training time increased linearly indicating the time to process more data.
- As the number of samples increased, predicting time remained constant for the most part.

From the results above, we can see that Decision Tree is neither fast nor accurate. However, XGBoost and LinearRegression are fairly promising and we can continue our investigation by tweaking the parameters using Grid Search CV algorithm.

## Refinement

In this section, we will fine tune parameters of the following models: LinearRegression and XGBoost using sklearn.GridSearchCV module.

- Linear Regressor
  - Linear Regressor Final MAE testing data: 1228.0901

```
parameters = {'normalize': [True, False],
              'n_jobs': [1, -1],
}
```

- XGBRegressor
  - XGBRegressor Final MAE testing data: 1168.5284

```
parameters = {'max_depth': [6, 7, 8],
              'learning_rate': [0.03, .3, 1],
              'n_estimators': [100, 250, 500],
}
```

Linear Regressor's model accuracy remained same despite tweaking its parameters. However, XGBRegressor accuracy increased from 1251 to 1168 after tuning its parameters.

Note: Running GridSearch on XGBRegressor with the above parameters took very long time to complete. To save time, I saved the model into a pikle file: xgboost.pkl.

# IV. Results

### Model Evaluation and Validation

We initially started off with three models: LinearRegressor, DecissionTreeRegressor, and XGBRegressor. From those three, we have decided to eliminate DecissionTreeRegressor, because it was slow and not very accurate. We used GridSearch to optimize the parameters for both LinearRegressor and XGBRegressor. Linear Regressor did not improve accuracy despite tweaking its parameters; however, it was very quick. On the other hand, XGBRegressor improved a lot, but took longer time.

- Benchmark score: 1222.24 (from average submissions)
- Linear Regression: 1228.0901
- XGBRegressor: 1168.5284

From the above results, we can see that Linear Regression was very close to the Base Score we setup. XGBRegressor shined as it was better than the Base Score by 1222.24 - 1168.5284 = 53.71.

GridSearch on XGBRegressor took extremely long time. As a result, I saved the model into a pikle file. In order to test the reproducibility of the model, I have reloaded the model again and ran our model on both training and testing data to predict the MAE. Results are shown below:

- XGBRegressor Final MAE Training data: 1000.0910
- XGBRegressor Final MAE Testing data: 1178.2932

Optimum Parameters used for this model are as follows:

```
XGBRegressor(base_score=0.5, colsample_bylevel=1, colsample_bytree=1, gamma=0,
       learning_rate=0.03, max_delta_step=0, max_depth=8,
       min_child_weight=1, missing=nan, n_estimators=500, nthread=-1,
       objective='reg:linear', reg_alpha=0, reg_lambda=1,
       scale_pos_weight=1, seed=0, silent=True, subsample=1)
```

The time to train XGBoost increased as the number of training data samples, max_depth, and n_estimators increased. However, this also increased the accuracy of our predictions on testing data. Similarly, as the learning_rate decreased, our model’s accuracy increased. Given more time, I would have tweaked these values a little more to see when the accuracy converges.

I saved this model in a pickle file and predicted our model’s performance on multiple sets of testing data. In all cases, our accuracy was very high and better than our benchmark metric. This made me conclude that our model was very robust.

## Justification

We ran our most optimum model again to predict the target variable: loss. Our model performed extremely well on training data as the MAE is roughly 1000. This is a lot better than our benchmark score of 1222.24. Similarly, it did a great job in predicting unseen data as well. The MAE score on testing data was 1178.3, which was also better than our benchmark score. This is indicating that our model is neither under-fitting nor over-fitting. If our model was under fitting, our MAE scores would not have been as high. For example, a model which finds the average of ‘loss’ column is under-fitting, because this model is very simple. Similarly, if our model was overfitting, our model would not have performed well on unseen testing data.

## V. Conclusion

### Free-Form Visualization

Lets visualize features which are the most important for our XGBRegressor model. We can get them by using the models helper function. We are sorting them by their value and plotting a bar chart below. X-axis represents the Feature #, while the y-axis is the feature weight.

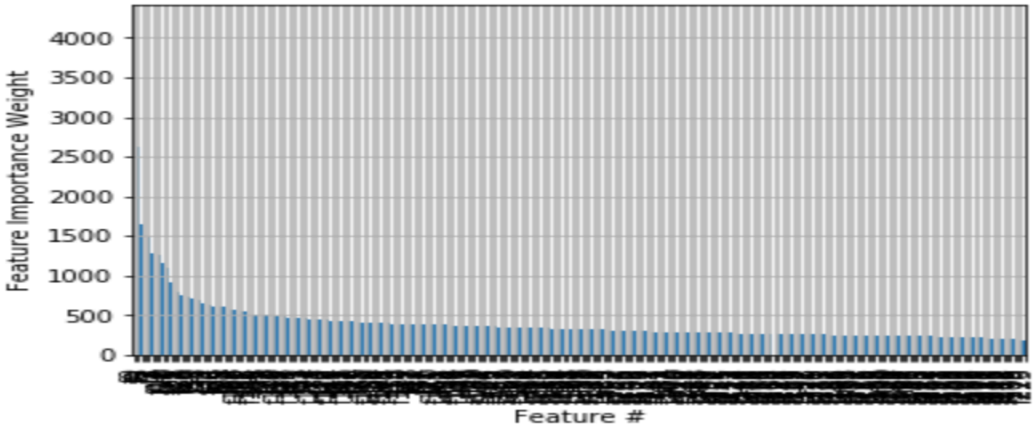


Figure 4

From figure 4, we can see that the first few features contribute a lot more than rest of other features. Lets analyze this a little further by only plotting the first 20 features.

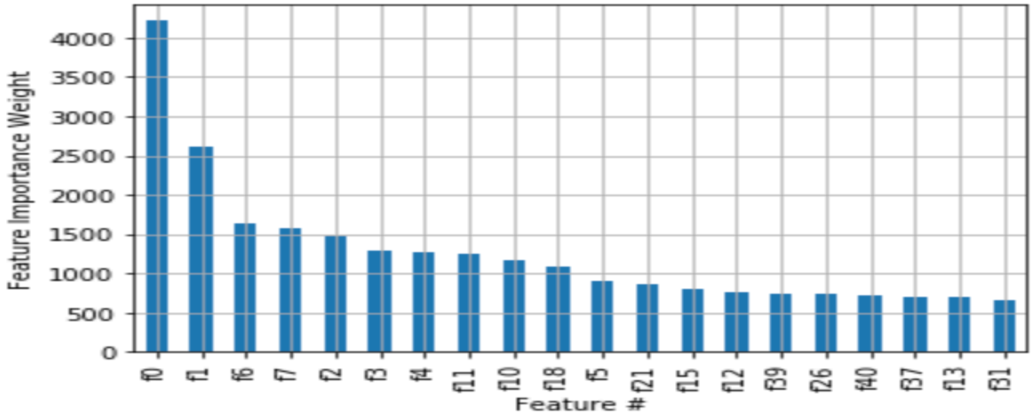


Figure 5

From figure 5, we can see that f0 is the most import feature in our model with feature importance weight of 4200. The second most important feature, f1, has a feature importance weight of 2500. The difference between first and second feature is close to 100%.

### Reflection

I have used the following steps to solve our problem:

- Data Exploration
  - 1) Started off by identifying the types of input variables and dependent variables i.e. whether its continuous or discrete/categorical.
  - 2) Found relevant statistics about the target data like: mean, mode, median, etc. Here, I realized the loss variable was positively skewed and had to use log(loss) to normalize the data.
  - 3) I tried to find any continuous features that are linearly related. Despite finding these, I decided not to remove them as my focus was to have an accurate model as opposed to a quicker model.
  - 4) Removed outliers
  - 5) The categorical data had to be one-hot encoded, because machine learning algorithms are only capable of handling numbers and not special characters.
  - 6) Used PCA to reduce the dimensions of the data for quicker processing, while not sacrificing accuracy.

- Training the Datasets
  - 1) Split the data into training and testing
  - 2) Pick three different models: Linear Regression, Decision Trees, and XGBoost and use our training data to come up with a model.
  - 3) Validate our model performance measured by MAE on testing data (new data that our model has not seen).

- Testing and Optimizing
  - 1) At this point, I have identified that Linear Regression and XGBRegressor were very accurate.
  - 2) Used GridSearch to run our models through various parameters to optimize our results. This took extremely long time for XGBRegressor, but it gave very accurate results.
  - 3) Our final model had a MAE of 1000.0910 on training and 1178.2932, which is a lot better than our base score of 1222.2.

I was extremely impressed by my results. A score of 1178 is way better than the average score on Kaggle, especially when the best score was 1109.7. I think having a faster hardware would have encouraged me to experiment with more aggressive parameters for XGBRegressor. We have reduced our features from 1148 to 250 using PCA. I would have used a higher n_components, which could potentially improve our accuracy as well. Our feature labels had no meaning to them. Despite this fact, I think my model was able to perform well. Even though training time was long and increased proportional to the size of data, our prediction time remained constant, and quick.

## Improvement

- While our results were very accurate and had a low MAE score, it was extremely time consuming to train an optimum model using GridSearchCV for XGBRegressor model. I could have chosen fewer data points or rows for this step, but this would not have guaranteed low MAE score or high accuracy.
- For PCA, I reduced 1148 features to 250. While this gave a variance of 0.9724, I believe using more dimensions would resulted in better accuracy.
- Our optimum XGBRegressor model used highest max_depth and n_estimators. This makes me believe using higher values would have improved our model's accuracy. Similarly, our model also used the lowest learning_rate, and using an even smaller learning_rate would have improved our model's accuracy.
- If time permitted, I would have used other Regression Models Keras.
- A faster CPU or better hardware could have improved the algorithms performance, specifically reduce the time of computation.

## Reference

- https://www.kaggle.com/c/allstate-claims-severity
- http://asirt.org/initiatives/informing-road-users/road-safety-facts/road-crash-statistics
- http://scikit-learn.org/stable/supervised_learning.html
- https://en.wikipedia.org/wiki/Mean_absolute_error
- http://www.lauradhamilton.com/machine-learning-algorithm-cheat-sheet
- https://github.com/ctufts/Cheat_Sheets/wiki/Classification-Model-Pros-and-Cons
- https://www.quora.com/What-are-the-advantages-of-different-classification-algorithms
- http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html
- Previous Submissions: https://github.com/siddartha1992/machine-learning/tree/master/projects/
- https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/
- https://people.duke.edu/~rnau/compare.htm
- https://www.xoriant.com/blog/product-engineering/decision-trees-machine-learning-algorithm.html
- http://xgboost.readthedocs.io/en/latest/model.html