Machine Learning Engineer Nanodegree

Capstone Project

I. Definition

Project Overview

Supervised learning is one of the most promising field in machine learning where already much   development has taken place and is currently used in real world application. The customer is the focal point of any business and is directly proportional to growth of the business. One area which affect the customers the most is the services provided and the delay in them. In order to reduce the waiting period, the business can use the machine learning algorithms. It will not only help the customers but also let businesses to focus on other things which require regular human intervention.

Supervised learning is the task of inferring a function from labeled data.it has two types of data: training data and testing data. The training data consist of a set of *training examples*. In supervised learning, each example is a *pair* consisting of an input object (typically a vector) and a desired output value (also called the *supervisory signal*). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. This inferred function is then used to predict the output value of the testing data. We already have the results of the test data. The difference between the predicted data and the results are used as a performance metric to ensure the accuracy   of   the   inferred model.

My personal motivation is this fact only that the very first use of machine learning in my mind is to reduce the frequency of redundant tasks, So that we can invest our times in better work and this problem addresses exactly that.

The link to my data source is:https://www.Kaggle.com/c/allstate-claims-severity/data

Datasets and Inputs

The dataset contains 2 .csv files with information necessary to make a prediction. They are:

1. train.csv and test.csv - contains

- id - the id of a training set question pair
- cat1 to cat116 - category variables (the range of values is not provided, neither the column names).
- cont1 to cont14 - continuous variables (the range of values is not provided, neither the column names).
- loss - the amount which the company has to pay for a particular claim. This is the target variable. In test .csv, loss is not present since we are going to predict that.

   2. In train.csv -

- Number of rows = 188318
- Number of columns = 132
- Highly relevant as this is the data we will train on.

3. In test.csv -

- Number of rows = 125546
- Number of columns = 131
- Highly relevant as this is the data we will test on.

As this was a Kaggle competition. The dataset is provided by Kaggle and Quora. They can be obtained [here](#).

Problem Statement

The **Allstate Corporation** is the second largest personal lines insurer in the United States and the largest that is publicly held. Due to its large size, they have tackle a large number of claims which takes time done by a human.

Allstate is currently developing automated methods of predicting the cost, and hence severity, of claims. The problem is to create an algorithm which accurately predicts claims severity. Basically, given the data , predict the amount to be paid by Allstate.

We want to understand the relationship between the 130(116+14) features and the loss. Although it looks straightforward but it isn't so. We have to find the relations between the features and convert categorical values from alphabets to numbers which can be used in models. Then we would test a few models to check which performs best using Kfold splitting and finally get the accuracy. The models to be used are: linear regression (as base model) and XGBoost (as trusted algorithm). This solution is a good fit for the problem as first gets a base score using naive model and later on uses complex models to attain better scores.

The end result we are looking for is the predicted amount that will be claimed.

Metrics

The model prediction for this problem can be evaluated in several ways. Firstly, our problem is regression type, so we can't use the metrics for classification or clustering like precision, accuracy, recall. We have at our disposal, mean absolute error, mean squared error, mean squared log error and r2 score. Now the r2 square checks how much variation is described by the fitted line and usually used for linear cases. Since we are more concerned as to how much do the company has to pay extra (currency wise) we should use a MAE or MSE. MSE penalizes large errors more than the small error, however, MAE gives equal weightage to every error. We could have used either of those but since the official evaluation of this project is done by Kaggle using mean absolute error (Lower it is, better the model), same will be used for evaluation of models.

Mean Absolute Error (MAE)

MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. It's the average over the test sample of the absolute differences between prediction and actual observation where all individual differences have equal weight.

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

Fig 1: Formula for calculating MAE

II. Analysis

Data Exploration

Data exploration is done in file **EDA_and_base_score_part1.ipynb**

The first five columns of train data looks like this. The first thing we notice is that it contains both categorical as well as continuous features. Secondly, categorical features are alphabetical and since, the models in machine learning are designed and optimized to work well with numeric vectors, we need to convert them into numeric values.

```
    id cat1 cat2 cat3 cat4 cat5 cat6 cat7 cat8 cat9 cat10 cat11 cat12 cat13  \
0    1    A    B    A    B    A    A    A    A    B     A     B     A     A
1    2    A    B    A    A    A    A    A    A    B     B     A     A     A
2    5    A    B    A    A    B    A    A    A    B     B     B     B     B
3   10    B    B    A    B    A    A    A    A    B     A     A     A     A
4   11    A    B    A    B    A    A    A    A    B     B     A     B     A

   cat14 cat15 cat16 cat17 cat18 cat19 cat20 cat21 cat22 cat23 cat24 cat25  \
0      A     A     A     A     A     A     A     A     A     B     A     A
1      A     A     A     A     A     A     A     A     A     A     A     A
2      A     A     A     A     A     A     A     A     A     A     A     A
3      A     A     A     A     A     A     A     A     A     B     A     A
4      A     A     A     A     A     A     A     A     A     B     A     A

   cat26 cat27 cat28 cat29 cat30 cat31 cat32 cat33 cat34 cat35 cat36 cat37  \
0      A     A     A     A     A     A     A     A     A     A     A     A
1      A     A     A     A     A     A     A     A     A     A     A     A
2      A     A     A     A     A     A     A     A     A     A     B     A
3      A     A     A     A     A     A     A     A     A     A     A     A
4      A     A     A     A     A     A     A     A     A     A     A     A

   cat38 cat39 cat40 cat41 cat42 cat43 cat44 cat45 cat46 cat47 cat48 cat49  \
0      A     A     A     A     A     A     A     A     A     A     A     A
1      A     A     A     A     A     A     A     A     A     A     A     A
2      A     A     A     A     A     A     A     A     A     A     A     A
3      A     A     A     A     A     A     A     A     A     A     A     A
4      A     A     A     A     A     A     A     A     A     A     A     A

   cat50 cat51 cat52 cat53 cat54 cat55 cat56 cat57 cat58 cat59 cat60 cat61  \
0      A     A     A     A     A     A     A     A     A     A     A     A
1      A     A     A     A     A     A     A     A     A     A     A     A
2      A     A     A     A     A     A     A     A     A     A     A     A
3      A     A     A     A     A     A     A     A     A     A     A     A
4      A     A     A     A     A     A     A     A     A     A     A     A

   cat62 cat63 cat64 cat65 cat66 cat67 cat68 cat69 cat70 cat71 cat72 cat73  \
0      A     A     A     A     A     A     A     A     A     A     A     A
1      A     A     A     A     A     A     A     A     A     A     A     A
2      A     A     A     A     A     A     A     A     A     A     A     A
3      A     A     A     A     A     A     A     A     A     A     A     B
4      A     A     A     A     A     A     A     A     A     A     B     A
```

```
   cat74 cat75 cat76 cat77 cat78 cat79 cat80 cat81 cat82 cat83 cat84 cat85  \
0    A     B     A     D     B     B     D     D     B     D     C     B
1    A     A     A     D     B     B     D     D     A     B     C     B
2    A     A     A     D     B     B     B     D     B     D     C     B
3    A     A     A     D     B     B     D     D     D     B     C     B
4    A     A     A     D     B     D     B     D     B     B     C     B

   cat86 cat87 cat88 cat89 cat90 cat91 cat92 cat93 cat94 cat95 cat96 cat97  \
0    D     B     A     A     A     A     A     D     B     C     E     A
1    D     B     A     A     A     A     A     D     D     C     E     E
2    B     B     A     A     A     A     A     D     D     C     E     E
3    D     B     A     A     A     A     A     D     D     C     E     E
4    B     C     A     A     A     B     H     D     B     D     E     E

   cat98 cat99 cat100 cat101 cat102 cat103 cat104 cat105 cat106 cat107 cat108  \
0    C     T     B      G      A      A      I      E      G      J      G
1    D     T     L      F      A      A      E      E      I      K      K
2    A     D     L      0      A      B      E      F      H      F      A
3    D     T     I      D      A      A      E      E      I      K      K
4    A     P     F      J      A      A      D      E      K      G      B

   cat109 cat110 cat111 cat112 cat113 cat114 cat115 cat116     cont1      cont2  \
0    BU     BC     C      AS     S      A      0      LB    0.726300   0.245921
1    BI     CQ     A      AV     BM     A      0      DP    0.330514   0.737068
2    AB     DK     A      C      AF     A      I      GK    0.261841   0.358319
3    BI     CS     C      N      AE     A      0      DJ    0.321594   0.555782
4     H     C      C      Y      BM     A      K      CK    0.273204   0.159990

      cont3      cont4      cont5      cont6      cont7      cont8     cont9  \
0  0.187583   0.789639   0.310061   0.718367   0.335060   0.30260   0.67135
1  0.592681   0.614134   0.885834   0.438917   0.436585   0.60087   0.35127
2  0.484196   0.236924   0.397069   0.289648   0.315545   0.27320   0.26076
3  0.527991   0.373816   0.422268   0.440945   0.391128   0.31796   0.32128
4  0.527991   0.473202   0.704268   0.178193   0.247408   0.24564   0.22089

     cont10     cont11     cont12     cont13     cont14     loss
0  0.83510   0.569745   0.594646   0.822493   0.714843   2213.18
1  0.43919   0.338312   0.366307   0.611431   0.304496   1283.60
2  0.32446   0.381398   0.373424   0.195709   0.774425   3005.09
3  0.44467   0.327915   0.321570   0.605077   0.602642    939.85
4  0.21230   0.204687   0.202213   0.246011   0.432606   2763.85
```

Fig 2:Train_data first five columns

The test data is same with the exception of not having a loss column.

In case of continuous data, we can find the mean and standard deviation to check the outliers.

Seeing it, it doesn't look like we need to scale it, it looks quite balanced. Although, loss column needs further inspection. Also, the values are already between 0 and 1 so, we are not going to scale it.

```
                id           cont1           cont2           cont3
count  150000.000000  150000.000000  150000.000000  150000.000000
mean   234490.288887       0.493660       0.507320       0.498990
std    134937.378713       0.187537       0.207229       0.202252
min         1.000000       0.000016       0.001149       0.002634
25%    117537.750000       0.346090       0.358319       0.336963
50%    234497.500000       0.475784       0.555782       0.527991
75%    351272.000000       0.623912       0.681761       0.634224
max    467728.000000       0.984975       0.862654       0.944251

                cont4           cont5           cont6           cont7
count   150000.000000  150000.000000  150000.000000  150000.000000
mean         0.491802       0.487722       0.490615       0.484894
std          0.211373       0.209136       0.205168       0.178525
min          0.176921       0.281143       0.012683       0.069503
25%          0.327354       0.281143       0.335580       0.350175
50%          0.452887       0.422268       0.440945       0.438285
75%          0.652072       0.643315       0.653958       0.590687
max          0.952482       0.983674       0.997162       1.000000

                cont8           cont9          cont10          cont11
count   150000.000000  150000.000000  150000.000000  150000.000000
mean         0.485975       0.485215       0.497761       0.493302
std          0.199114       0.181607       0.185666       0.209816
min          0.236880       0.000080       0.000000       0.035321
25%          0.312800       0.358970       0.364580       0.310961
50%          0.441060       0.437310       0.461190       0.457203
75%          0.623580       0.558550       0.614590       0.678924
max          0.980200       0.993790       0.994980       0.998742

                cont12          cont13          cont14            loss
count   150000.000000  150000.000000  150000.000000  150000.000000
mean         0.492966       0.492626       0.496251    3040.378682
std          0.209489       0.212733       0.222599    2917.478617
min          0.036232       0.000228       0.179722       0.670000
25%          0.308395       0.315758       0.294772    1202.187500
50%          0.462286       0.363547       0.409813    2115.875000
75%          0.675759       0.689974       0.724733    3869.207500
max          0.998484       0.988494       0.844848  121012.250000
```

Fig 3: Mean, std, min, max, etc. of continuous features

We will check for skewness now.

```
id        -0.004008
cont1      0.517909
cont2     -0.312079
cont3     -0.010250
cont4      0.415957
cont5      0.680679
cont6      0.464061
cont7      0.828456
cont8      0.679399
cont9      1.074892
cont10     0.356928
cont11     0.282640
cont12     0.293526
cont13     0.385175
cont14     0.243783
loss       3.901837
dtype: float64
```

Fig 4: Skewness of continuous variables

This clearly shows that loss is positively skewed and needs to be rectified. Next, we look at the violin plots of continuous features.

It didn't make sense to copy all the 14 plots (would have increased page count for no reason). You can view them directly from the code in file - **EDA_and_base_score_part1.ipynb**.

The results drawn were -

Cont1 has many values close to 0.5.

Cont2 has a pattern where there a several spikes at specific points.

Cont5 has many values near 0.3.

Cont14 has a distinct pattern. 0.22 and 0.82 have a lot of concentration.

Loss distribution must be converted to normal.

The necessary step to transform loss.

Next, we see the correlation between different features and print out the top entries.

```
cont11 and cont12 = 0.99
cont1 and cont9 = 0.93
cont6 and cont10 = 0.88
cont6 and cont13 = 0.81
cont1 and cont10 = 0.81
cont6 and cont9 = 0.80
cont9 and cont10 = 0.79
cont6 and cont12 = 0.79
cont6 and cont11 = 0.77
cont1 and cont6 = 0.76
cont7 and cont11 = 0.75
cont7 and cont12 = 0.74
cont10 and cont12 = 0.71
cont10 and cont13 = 0.71
cont10 and cont11 = 0.70
cont6 and cont7 = 0.66
cont9 and cont13 = 0.64
cont9 and cont12 = 0.63
cont1 and cont12 = 0.61
cont9 and cont11 = 0.61
cont1 and cont11 = 0.60
cont1 and cont13 = 0.53
cont4 and cont8 = 0.53
```

Fig 5: Top correlated features

We see that top 5-6 entries are highly correlates which presents the opportunity to apply dimensionality reduction method like PCA.

Now, we count the number of different labels in each categorical feature.
Again, it doesn't make sense to copy all the 114 plots here so, refer to them in the file -

EDA_and_base_score_part1.ipynb.

Results were -
Cat1 to Cat72 have only two labels A and B. In most of the cases, B has very few entries.

Cat73 to cat 108 have more than two labels.

Cat109 to cat116 have many labels.

Algorithms and Techniques

The very first algorithm to be used is linear regression.

Linear regression

Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. One variable is considered to be an explanatory variable, and the other is considered to be a dependent variable. For example, a modeler might want to relate the weights of individuals to their heights using a linear regression model. For example, consider that we have data about prices of houses. We take their sizes and plot it on x axis and their price on y axis. Then we plot our data. It looks somewhat as depicted in the image below.

Housing price prediction.
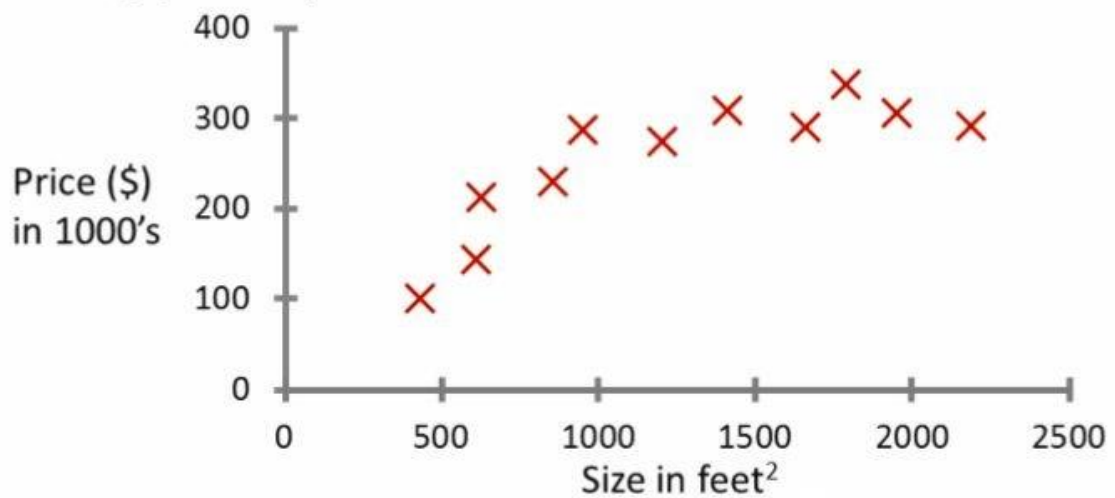
Fig 6: Graph between price and size of houses

Now we can fit a curved line passing through these points. And use that line to predict the house given its size in the future.

It's a naive classifier and won't work in our dataset since it has too much dimensions. So, it will be used as base classifier and we will try to minimize the loss it gives using a better model.

The second model to be used is XGBoost.

XGBoost

XGBoost is an algorithm that has recently been dominating applied machine learning and Kaggle competitions for structured or tabular data.XGBoost is an implementation of gradient boosted decision trees designed for speed and performance.XGBoost stands for e **X**treme **G**gradient **B**oosting. The term "Gradient   Boosting" is proposed in the paper Greedy Function Approximation: A Gradient Boosting Machine, by Friedman.
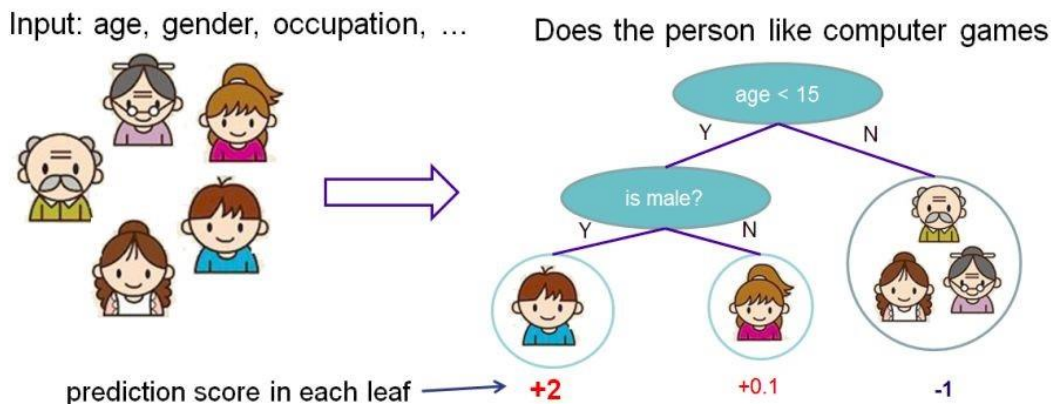
Fig 7: Depiction of trees

A tree is simply a series of decisions. For example, as seen in the diagram, to check if a person like computer games, given features like age, gender, occupation etc. we construct a tree as if age is less than 15 then furthermore check if the person is male and outputs the probability. In gradient boosting, we take several weak learners(models whose performance is just better than chance) and combine them to get a strong model. To take our previous example again, let's say that the decision tree shown in diagram have a performance just better than chance(>50% accurate). Let's say we have constructed another decision tree model which asks whether the person is student or employed and its performance alone is also just better than chance. Then, we use both the trees and to combine their result, we use voting(give a weightage to their individual decision). This is tree boosting and taking many trees and combining them to give a greater performance than any tree individually.

Xgboost just uses the full extent of the machine it is running on.

The two reasons to use XGBoost are also the two goals of the project:

1. Execution Speed.
2. Model Performance.

XGBoost is openly available to download.
Before implementing both the algorithms, we also have to preprocess the data. This will be discussed later.

Benchmark

As this is a Kaggle competition a benchmark model would be the best Kaggle score for the test set, which comes in at 1109.70772 mean absolute error (lower is better). However, due to hassle and limitations (like 5 entries per day), for academic purposes, we would use a part of training data as testing data. More precisely, we would use the last 38,318 entries of training set as testing set.

We will run the linear regression model to get a base MAE. Then, we can compare our next model with it to see of it can beat it and by how much extent. We will take the best model and for satisfying the curiosity, run it on test set provided by Kaggle as test dataset. The submission file will be uploaded on Kaggle website to check the score. Then, we can also compare our model with the benchmark model hosted by Kaggle. A personal goal would be to be in the top 20 % i.e. less than 1121.21401 error of the Kaggle Private Leaderboard.

First we run the linear regression in **final_model_part1.ipynb**. Train it on 150000 entries and then find the mae for the rest around 30000 entries. The result comes to be **1267.48**

This is our base score and our aim is to beat this score, hopefully by a good margin.

III. Methodology

Data Preprocessing

First, we will normalize the loss column since during our exploratory analysis, we found that it is positively skewed. Apart from this, we perhaps does not require anything else to be done.

Next, we store the loss column in target variable and drop the id and loss column. We will take log of the target + shift.

Shift is another hyper-parameter. It sometimes gives better performance. I have taken it from the discussion linked here -

https://www.Kaggle.com/c/allstate-claims-severity/discussion/24611#141269

After taking log, our loss is normalized.
Since we can't use alphabetical categories, we factorize the columns cat1 to cat114.

We are going to use a part of the train data as test data. So we take 150000 entries as train data and the rest 38,318 entries as test data. In other words, roughly 20 percent of training data is taken as test set.

Later on, we also have to check at Kaggle , what score we get, but there's another problem in it. Since, there are entries which are in train data and not in test data and vice versa, we will first join them and then factorize and then divide them again into train and test data.

It doesn't look promising to remove outliers because there aren't much of them and those which are present, are not affecting much.

Implementation

Although, the implementation is pretty straightforward, I would still walk through it. We have already preprocessed the data as described above. Now that we have our training and testing data, next step is to implement our base classifier (linear regression). We simply import linear regressor from sklearn and also mean absolute error. Then we train the model. Then, we predict the results of the test set. One thing to note here is that, since the target variable was transformed as log(loss+shift), we have convert the predicted result back. This could be done using the exponential. So we predict the loss and take their exponential and then subtract shift from it. Then we calculate the MAE between these values and the actual values.

Next we have to implement the XGBoost model.
First we does what we did for linear regression model, i.e. Preprocess the data and divide it into training and testing dataset.

Then, a simple XGBoost classifier is built using usual parameters without any tuning and compare the results.

There was no major part in the code which needed to be heavily documented. I have already documented bits and pieces almost everywhere.

Naive XGBoost classifier is built in file - **naive_xgboost_part2.ipynb**

t

use Bayesian Optimization in file, **bayesian optimization (XGBoost).ipynb**

Since, it requires a lot of computing power , I ran it on google cloud, but for some reason , the server stopped before the code could fully run. I tried three times but same result. Please note that i used the trial where 300 dollars were provided but limitations were imposed like maximum of 8 cores were available and no GPU.

Next , I decided to use grid search, in file, **XGBoost_grid_search.ipynb**. It worked on a dataset on 10 or 100 elements in my local machine, however when I ran it on full dataset on google cloud, it always gave an error of bad_alloc memory and even after hours, I could not find a solution and it's also possible that this is a bug since the XGB classifier is relatively new and may contain more bugs than usual. I also asked about it in Udacity one on one but no solution could be found.

So , at last I decided to fine tune manually using xgb.cv . I confirmed regarding its use on conference and was given a go ahead. It's done in file **manual_tuning_part3.ipynb**

Refinement

The result obtained by naive xgboost was **1153.071**, which is really good improvement over our base classifier (**1267.48**).

Next, We need to find the optimal parameters to use in the model. To do that, we first use Bayesian Optimization. it is implemented in file-**bayesian optimization (XGBoost). ipynb** . The things I could infer from the partial result I obtained were:

It did give an idea of what parameters to try like max_depth should be in range 10-15 and alpha less than 5, col_sample bytree around 0.8 and gamma around 2, man-child weight 10 or less, subsample, not so much so have to try that between 0.5 to 1.

Next , I decided to use grid search, in file, **XGBoost_grid_search.ipynb**. The error I received is already discussed in the implementation section.

In file, **manual_tuning_part3.ipynb**, the first two parameters which I am going to tune are max_depth and min_child_weight. From bayesian, I already had an idea. Started with list with gap of two with max_depth 10 to 14 and min_child_weight 1 to 7. Also, set the learning rate 0.1 to make the process faster. We will adjust it later.

The result comes out to be max_depth=10 and min_child_weight=7. Now, we check one above and below values, so run again with values max_depth 9,10,11 and min_child_weight 6,7,8.

The result comes out to be max_depth=9 and min_child_weight=8.

since , max_depth is lower, we should check other values also , so we ran it for the third time with max_depth in range 5 to 9 and the best performance is for **max_depth = 6** and **min_child_weight = 8**. These are the final values for them.

Next, we tune gamma. We first give it the values from [0.0,0.1,0.2,0.5,0.8,1.0].the best value comes at **gamma = 0.0**. We fix that. Next, we check for col_sample(0.3,0.5,0.7,0.9) and subsample(0.3,0.5,0.7,0.9) and get 0.5 and 0.9. Now, check for col_sample(0.45,0.5,0.55) and subsample(0.85,0.9,0.95,1.0). Results come as **col_sample = 0.45** and **subsample = 1.0.**

At last, we use these parameters and reduce the **learning rate to 0.01** to find the score on the test set now.

IV. Results

Model Evaluation and Validation

Our fine-tuned model gives us good results. It's evident how we have chosen it. After training the model with the good parameters, we find the MAE between loss and predicted test set.

Since we transformed the loss using log, we have to take the exponential of it before finding the MAE.

The final score obtained using fine-tuned XGBoost model is **1132.54788745**. This is better than both the base classifier score and the untuned XGboost score. It's done in file - **Tuned_XGBoost_part4.ipynb**

Next, we want to check its Kaggle score. So, we train our model on the whole training dataset and then make a submission file. We submit that file on Kaggle to find the loss. Its done in file - **XGBoost_Kaggle_submission_part5.ipynb**.

The loss comes out to be **1122.14212**.

Justification

The model passes our base classifier's performance.

As we don't have enough knowledge about the domain the problem, we can't say for sure if we have solved the problem. However, we does have Kaggle scores.

As compared to them, we can say that our model is quite near to the 1st rank (1109) model so it can be said usable.

V. Conclusion

Free-Form Visualization

The violin plot for loss was plotted and it can be clearly seen that it is positively skewed, concentrated only on the lower end of the graph and therefore, as long as we don't transform it. We cannot have a low error.
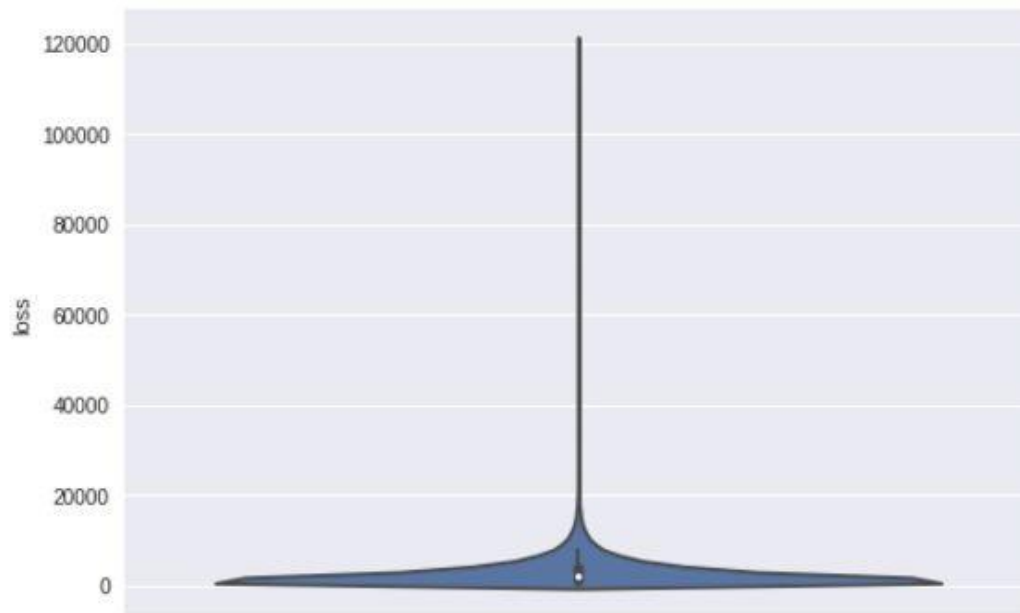
Fig 8: Violin plot of loss

Reflection

After doing the whole project, I have developed a model which is going to take about 130 features of any potential claim and gives the amount the insurance company may have to pay, although I can't say for sure it is ready to be used due to lack of domain knowledge, however, the MAE is quite fine as compared to other Keglers scores. It could have been better if we had more computing resources, but for the sake of academic purposes, I'd say we can stop for now.

This was the black box approach.
To go inside, I have created a simple linear regressor and used its MAE as base score, however, the aim was to beat that and after looking at the discussion forums, I deduced that XGboost is working much better than other models like SVM. Also, SVM is quite time taking so I went with XGBoost as next model. Even without parameter tuning it gave a better result. Next, we find tuned the hyper-parameters. It was particularly difficult since bayesian optimization died on google cloud and grid search always presented bad memory allocation error. At last, I decided to use xgb.cv with manual parameter tuning which finally worked. Lastly, I used those parameters to get a final score on test set (the last 38318 elements) of the training dataset.

I then trained this model on full training data and submitted it on Kaggle and got the score of - 1122.14212.

Improvement

1. I could have used to make a function for xgb.cv instead of typing it again and again.
2. Keras looks a potentially good model, however it would take even more resources than the current model with people on Kaggle reporting a training time of more than 10 hours and that too on good hardware. Therefore, I don't believe we need it for academic purposes.
3. 3) PCA or t-SNE could have been tried for feature reduction, however, since don't have column names and there's no point in visualizing the data in 2d (converting130 features to 2 or 3 features), and it will result in loss of data, giving higher MAE, I didn't tried it based on this intuition.

References

Intro to XGBoost -

https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/
Comparing MAE and RMSE - https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac 3bde13d
Fine tuning XGBoost -

https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/
Kernel by Vladimir - https://www.Kaggle.com/iglovikov/xgb-1114