Bonus Project Report

Huan Ma
1000955120

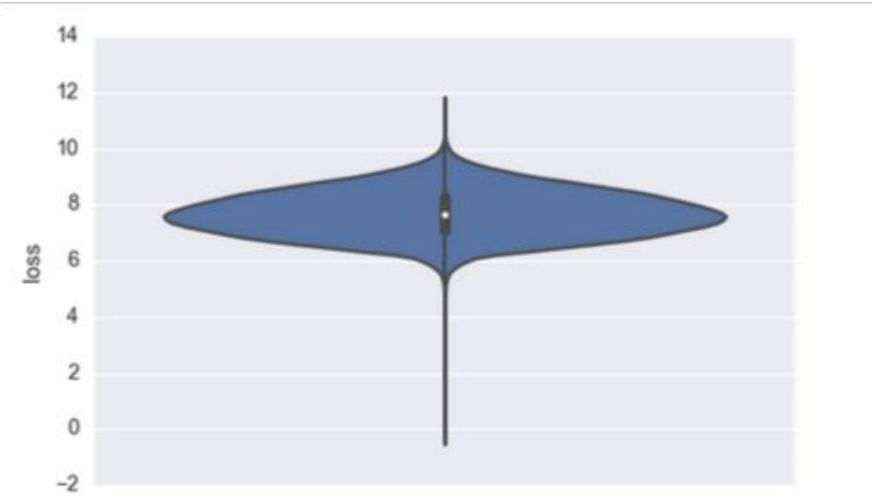
Problem description

The problem is called Allstate Claims Severity which is available on Kaggle and provided by Allstate. Allstate is developing automated methods of estimating the cost and the severity of a claim. As a result, in this task, we need to predict severities of claims based on existing data.

My baseline method

This is a regression problem since we need to predict on the loss value. I chose Linear Regression and K-Nearest Neighbor as the possible baseline methods which one is linear regression and one is non-linear regression. Before implementing these two methods directly, I did some data analysis and data transformation to make sure that our data would cause as little bias on the result as possible.

Through the analysis step of data, I notice that the data has a very high skewness on the loss column. After data transformation, we can notice from the plot below that the high skewness problem no longer exists.



Linear Regression

Linear Regression is a method where we can learning a mapping from one real-valued space to another. It has the mathematical form as follows.

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = \sum_{j=1}^{D} w_j x_j + b \quad .$$

where **w** vector are the weights need to be learned, **b** is the offset that need to be learned, and **x** vector are the features of a dataset, f(**x**) is the predicted label value of the corresponding features. Usually, we find a solution of the form above by Least Square Error(LSE) estimation which will provide us that $\mathbf{w}^* = (\tilde{\mathbf{X}}^T\tilde{\mathbf{X}})^{-1}\tilde{\mathbf{X}}^T\mathbf{y}$. But when $\tilde{\mathbf{X}}^T\tilde{\mathbf{X}})$ is singular, we cannot compute a result by this solution, in this task, unfortunately, the matrix is singular, so we use gradient descent to find a solution instead.

K-Nearest Neighbor

K-Nearest Neighbor can be a non-linear regression algorithm. The idea of K-Nearest Neighbor is trying to find the K nearest neighbors(shortest Euclidean distance) to X in the training set and

our prediction on the new data is simply the average of the output of these nearest neighbors. The mathematical form is as follows.

$$\mathbf{y} = \frac{1}{K} \sum_{i \in N_K(\mathbf{x})} \mathbf{y}_i$$

Conclusion

After implementing the two methods on our data, I decided to choose --- as my baseline because it has smaller mean absolute error. (Sadly, the data is too large that I cannot finish running before midnight Dec 20th) I think K-Nearest Neighbors should have a better performance than Linear Regression because K-Nearest Neighbors algorithm takes the average of the output of the nearest several data points of the new data point while linear regression tries to estimate the weights on each feature.

Advanced baseline from the Kaggle community

I chose the work from Ekaterina Aladyeva. He uses the XGBoost algorithm to predict the loss value. XGBoost is short for Extreme Gradient Boosting which is a popular and efficient method on Kaggle. It is a type of boosting algorithms. This is the loss function of XGBoost:

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$
$$\text{where } \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

And we take first gradient and second gradient:

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} [l(y_i, \hat{y}^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

$$\text{where } g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}) \text{ and } h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

Finally, we try to find the best split by

$$\mathcal{L}_{split} = \frac{1}{2} \left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma$$

There is a XGBoost library for Python which needs to be installed separately. The usage of this XGBoost is very similar to scikit-learn package.

My improvements

I was trying to improve XGBoost algorithm by implementing 5-fold cross validation and grid search on some parameters of XGBoost to find the optimal values for each parameter that minimizes the mean absolute error rate, namely, max_depth and n_estimators.

```
tuned_parameters={'max_depth': [2, 5, 8], 'n_estimators': [100, 200, 300, 500]}
gbm = GridSearchCV(xgboost.XGBRegressor(learning_rate = 0.05), tuned_parameters, cv=5, scoring= scoring_fnc)
```

This improved model should have a better performance than the advanced baseline from the Kaggle community. After model selection, values of the parameters of our model will be more precise for this dataset rather than random selected values of parameters.

Anything else that you learned in the process

I learned a new algorithm called XGBoost and it is very powerful. I know more about Kaggle and that running code might be time consuming if the dataset is not small.