Voting Loss function

Zekun Zhao

Department of Computer Science, BSOE,
University of California, Santa Cruz

{*zzhao99*}*@ucsc.edu*

Abstract

This report presents a novel methodology for measuring loss in machine learning algorithm which using a voting mechanism to measure how bad our current estimate is in respect to smoothness and accuracy. By introducing a voting mechanism, our loss function allows algorithms built around robust loss minimization to be generalized in a high dimensional feature space, which improves performance on basic regression tasks such as house price prediction. We interpreter our loss as one regularization method to capture rich input space features and to yield a general probability distribution that contains uncertainty of our prediction result. In this work, we implement our loss function on house price data set, which improves the prediction result. To explore more possibilities, we also show how the prediction result got influenced by different shapes of the voting label. Finally, we describe and visualize each loss function and its corresponding distribution, and document several of their useful properties.

1 Introduction

1.1 What's a loss function?

The loss function is a very important knowledge point in machine learning. It is used to estimate the degree of inconsistency between the predicted value $f(x)$ of the model and the true value y. Our goal is to minimize the loss function and let $f(x)$ be as close as possible to y. A gradient descent algorithm can usually be used to find the function minimum.

There are many different types of loss functions. No loss function is suitable for all problems [1]. You need to choose according to the specific model and problem. In general, loss functions can be roughly divided into two categories: Regression and Classification. We only focus on solving regression problems in this report.

1.2 Different types and flavors of loss functions

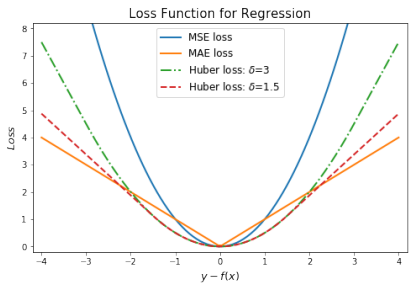
We only summarize three important loss functions commonly used in regression problems here. The three loss functions in the regression model include Mean Square Error (MSE), Mean Absolute Error (MAE), Huber Loss.

- **Mean Square Error** The mean square error refers to the average of the square of the distance between the model prediction value $f(x)$ and the true sample value y.

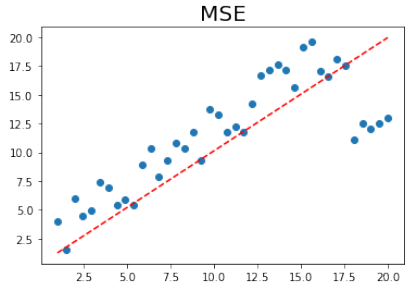
$$MSE = \frac{1}{m}\sum_{i=1}^{m}\left(y_i - f\left(x_i\right)\right)^2$$

  Where y_i and x_i represent the true and predicted values of the i-th sample, respectively, and m is the number of samples.

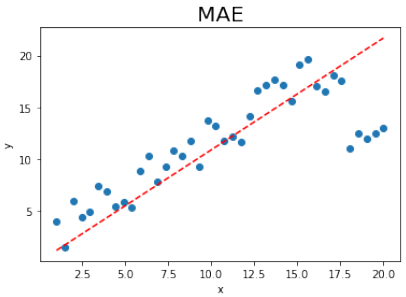
  The MSE curve is characterized by smooth continuous, steerable, and easy to use gradient descent algorithm, which is a commonly used loss function. Moreover, as the error of



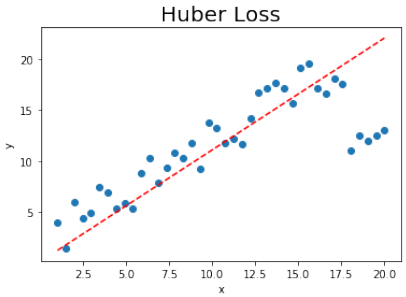
(a) Loss (Y-axis) vs. Y-f(x) (X-axis)



(b) Fitting curve using MSE loss



(c) Fitting curve using MAE loss



(d) Fitting curve using Huber loss

Figure 1: Comparison of three loss functions (To simplify the discussion for Fig 1a, we ignore the subscript i, and let $m = 1$).

the MSE decreases, the gradient also decreases, which facilitates the convergence of the function. Even if the learning factor is fixed, the function can obtain the minimum value faster.

However, if there are outliers in the sample, MSE will give outlier given more weight, but at the expense of other normal data points to predict the effect of the cost, which will ultimately reduce the overall performance of the model. In Figure 1, it can be seen that the MSE loss function is greatly affected by the outliers. Although there are only five outliers in the sample, the fitted straight lines are more biased toward the outliers.

- **Mean Absolute Error** The average absolute error refers to the average of the distance between the model prediction value f(x) and the sample true value y. Its formula is as follows:

$$MAE = \frac{1}{m}\sum_{i=1}^{m}|y_i - f(x_i)|$$

It is worth mentioning that MAE has an advantage over MSE that MAE is less sensitive and inclusive to outliers. Because MAE calculates the absolute value of the error, there is no square term effect, the penalty strength is the same, and the weight is the same.

Using the MAE loss function is less affected by the outliers, and the fitted line can better characterize the distribution of normal data. At this point, MAE is better than MSE. The comparison of the two is in Figure 1.

- **Huber Loss** Huber Loss 's formula is as follows:

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2, & |y - f(x)| \le \delta \\ \delta|y - f(x)| - \frac{1}{2}\delta^2, & |y - f(x)| > \delta \end{cases}$$

Huber Loss is a synthesis of above two loss functions, including a hyperparameter δ. The magnitude of the δ value determines the emphasis of Huber Loss on MSE and MAE. When $|y-f(x)| \leq \delta$, it becomes MSE; when $|y-f(x)| > \delta$, it becomes similar with MAE. Huber Loss combines the advantages of MSE and MAE to reduce the sensitivity to outliers and achieve a versatile function. In general, the hyper parameter δ can be chosen to achieve the best value by cross-validation. We take $\delta = 0.1, \delta = 10$, and draw the corresponding Huber Loss in Figure 1.

Huber Loss When $|y-f(x)| > \delta$, the gradient is always approximated as δ , which ensures that the model updates the parameters at a faster rate. When $|y-f(x)| \leq \delta$, the gradient gradually decreases, which ensures that the model obtains the global optimal value more accurately. Therefore, Huber Loss has the advantages of the first two loss functions. It can be seen that using Huber Loss as the activation function still has good immunity to outliers, which is stronger than MSE.

2 Voting mechanism

2.1 Motivation

- **Rich features [2]** If a task is very noisy or data is limited and high-dimensional, it can be difficult for a model to differentiate between relevant and irrelevant features. The neural network can extract features very well and help the model focus its attention on those features that matter for the task. Thus, if we capture more relevant features from our training samples in the learning process, the prediction would generate a better result.

- **Regularization [3]** Regularization is one of the many means to prevent over-fitting and is very common. By limiting the spatial parameter range, explicitly control the complexity of the model, thereby avoiding over-fitting.

- **Bias** As for the bias, this motivation comes from two previous experiments [4] done by David Parks. In the Astronomy work, it had a strong bias at the tail ends of the data. In another instance (a simple face detection model) where it outputs the X and Y coordinates of a bounding box center which tracked a face in the image, it performed progressively worse the closer to the edges the faces got. In both of the cases above, there are alternative explanations that might be the actual root cause. However, it was enough circumstantial evidence for us to consider the possibility that square error was the issue. It it does seem logical that bias might exist because the network will not make errors in the output above or below the maximum range it ever saw in training, so we should expect such a bias.

2.2 Main idea

Normally, we use y_i and x_i represent the true and predicted values of the i-th sample. But we cannot use only one value to have a voting effect in the learning process. Thus, in the voting loss function, we have converted our label to a high dimension. According to our motivations, there are several aspects we need to consider when we convert our label dimension: 1, Richer features. 2, Preventing over-fitting. 3, Eliminating the bias in the model. It is always better to use examples to illustrate new ideas. Here, we pick number 600 as our label (true values of the i-th sample).

Example of converting label to higher dimension in voting loss function first, we initialize a vector of length n ($n = 2 * mask + r$) , where "r" is the range of possible values of our samples and "$mask$" is a hyperparameter which we choose mask as $0.3 * r$. This vector will become our converted label after the later process and the initial vector is all zero for each index value.

Then, we set values for this vector. Around the index y_i, value of index y_i in our vector is zero and value of index $y_i + d$ in our vector is d, where $-mask \leq d \leq mask$. In addition, we need to set a special value k around the end of each two side of index $y_i - mask$ and $y_i + mask$. This process is because we do not want to penalize the predicted value in these index.

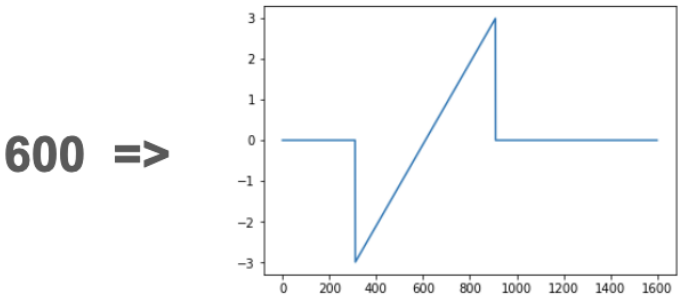


Figure 2: Example of converting label to higher dimension in voting loss function.

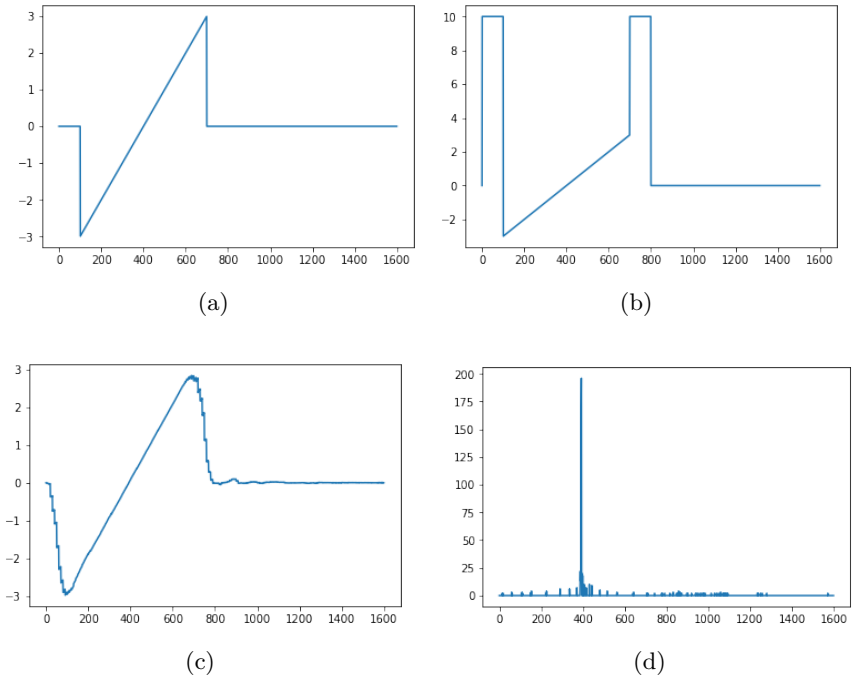


Figure 3: Process of converting label and making decision

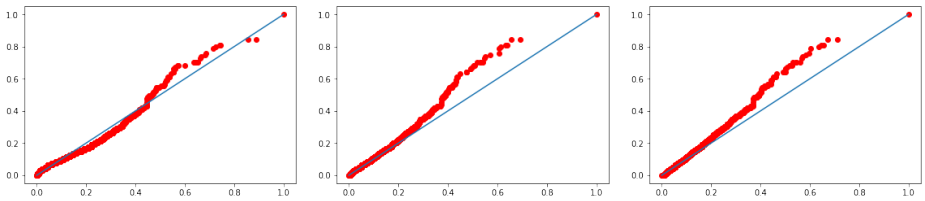
Finally,one last thing need to do is that we have to make sure that we can implement one voting function to convert the high dimension label back to our original label.

3 Experiment

In our experiment, we choose a house price data set, which contains 10 features: “bedrooms”, “bathrooms”,“sqft_lot”,“sqft_living”,“sqft_above”,“lat”,“long”,“sqft_lot15”,“sqft_living15”,“zipcode” and 21605 smaples after we delete abnormal data values.

3.1 what is the benefit of voting?

- **Smoothness** Since our loss function has considered a higher dimension value from voting, the result is smoother than just using point estimate evaluation.



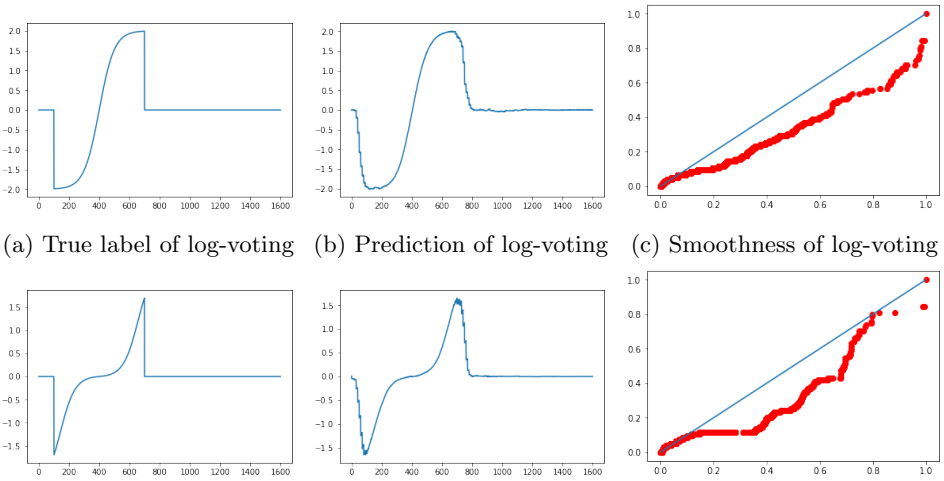
(a) Voting loss (b) MSE loss (c) MAE loss

Figure 4: True (Y-axis) vs. Predictions (X-axis)

- **Multitasking** In our label space, we do not only use the right index to predict our true value, but also use the information that which index is more relevant for making the final decision.
- **Uncertainty** After train our model, our prediction is also an n-length vector. For each sample label result, each voting index has a prediction, which actually can be desecrated as a distribution for our result. In this case, we can show our model uncertainty for each sample. One thing we need to mention here is that this uncertainty is not exactly the measurement of uncertainty of model, but there is some relationship between them.

3.2 Other voting strategy

Except for the linear voting strategy we have mentioned previously, we also test two other nonlinear voting strategy. Their converted shape are showing below:



(a) True label of log-voting (b) Prediction of log-voting (c) Smoothness of log-voting

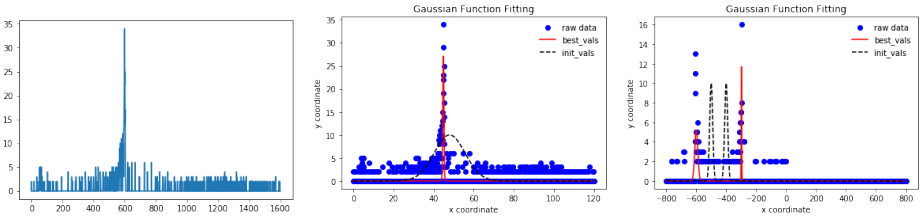
(d) True label of exp-voting (e) Prediction of exp-voting (f) Smoothness of exp-voting

Figure 5: Comparison of log-voting loss and exp-voting loss functions (Fig 5c and Fig 5f show True (Y-axis) vs. Predictions (X-axis) of log-voting and exp-voting loss)

From these different experiment results, we can see that in the same setting, the log_voting is better than exp_voting. It makes sense for the neural network model, that dramatic value change is hard for the neural network to learn. And this is also one part of the reason why we use a mask parameter in our voting loss function.

3.3 Model fitting method

To evaluating and comparing our model result with base line model, we have to convert our result back to point estimate. Since the result generated from our model is a distribution, we have to pick one most possible value from it. The easiest thing to do this is by using the majority voting strategy. However, when the result gives us some same voting number for several indexes, it seems not fair to just pick the one which has little advance than others. Thus, we use the Gaussian Mixture Model(GMM) to pick the right value for us. The idea shows in []. It is worthy of mentioning that sometimes, the voting result may generate two peaks for prediction, which can only be detected from GMM rather than majority voting. Ar voting strategy we have mentioned previously, we also test two other nonlinear voting strategy. Their converted shape is showing below:



(a) Voting result (b) One peak voting (c) Two peaks voting

Figure 6: Gaussian Mixture Model

4 Future work

In this project, we used fixed parameters (mask, length), but ideally, those parameters could be adapted by the distribution of our data set. For example, outliers in the sample should have a longer mask length since our model is less confident about predicting those value. Also, for bias, our result dose not provided a demonstration that the bias exists in square error, or is improved with the multi-task loss. More experiments need to be done to confirm this idea.

A code description

We successfully test our voting loss function in a multilayer perceptron model for one house price data set[1]. Here is the link to our code: `https://github.com/kriszhao/Voting-loss-function`.

References

[1] Christian Hennig. Some thoughts about the design of loss. *Main*, 5(276):1–19, 2007.

[2] Sebastian Ruder. An Overview of Multi-Task Learning in Deep Neural Networks. (May), 2017.

[3] Jonathan T. Barron. A General and Adaptive Robust Loss Function. 2017.

[4] David Parks, J. Xavier Prochaska, Shawfeng Dong, and Zheng Cai. Deep learning of quasar spectra to discover and characterize damped Lyα systems. *Monthly Notices of the Royal Astronomical Society*, 476(1):1151–1168, 2018.

[1] `https://github.com/kriszhao/Voting-loss-function/blob/master/kc_house_data.csv`

[5] Bernhard Scholkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[6] Cam Davidson-Pilon. *Bayesian Methods for Hackers*. 2014.