Xgboost-How to use "mae" as objective function?

Asked 6 years, 2 months ago Modified 11 months ago Viewed 24k times

33

I know xgboost need first gradient and second gradient, but anybody else has used "mae" as obj function?

machine-learning xgboost Edit tags

Share Edit Follow Close Flag

asked Jul 10, 2017 at 7:42

Sam Qian
341 1 4 4

3 Answers

Sorted by:
Reset to default

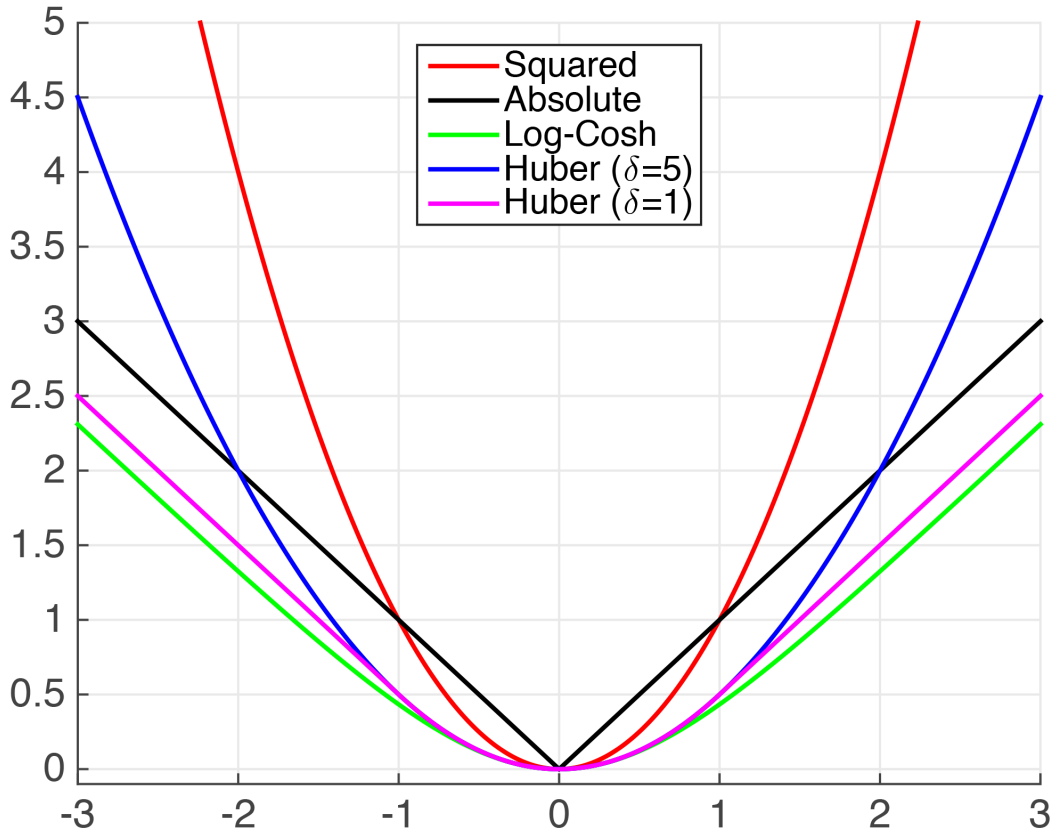
Date modified (newest first)

62

A little bit of theory first, sorry! You asked for the grad and hessian for MAE, however, the MAE is not continuously twice differentiable so trying to calculate the first and second derivatives becomes tricky. Below we can see the "kink" at `x=0` which prevents the MAE from being continuously differentiable.

Moreover, the second derivative is zero at all the points where it is well behaved. In XGBoost, the second derivative is used as a denominator in the leaf weights, and when zero, creates serious math-errors.

Given these complexities, our best bet is to try to approximate the MAE using some other, nicely behaved function. Let's take a look.



We can see above that there are several functions that approximate the absolute value. Clearly, for very small values, the Squared Error (MSE) is a fairly good approximation of the MAE. However, I assume that this is not sufficient for your use case.

Huber Loss is a well documented loss function. However, it is not smooth so we cannot guarantee smooth derivatives. We can approximate it using the Psuedo-Huber function. It can be implemented in python XGBoost as follows,

```
import xgboost as xgb

dtrain = xgb.DMatrix(x_train, label=y_train)
dtest = xgb.DMatrix(x_test, label=y_test)

param = {'max_depth': 5}
num_round = 10

def huber_approx_obj(preds, dtrain):
    d = preds - dtrain.get_labels() #remove .get_labels() for sklearn
    h = 1  #h is delta in the graphic
    scale = 1 + (d / h) ** 2
    scale_sqrt = np.sqrt(scale)
    grad = d / scale_sqrt
    hess = 1 / scale / scale_sqrt
    return grad, hess

bst = xgb.train(param, dtrain, num_round, obj=huber_approx_obj)
```

Other function can be used by replacing the `obj=huber_approx_obj` .

Fair Loss is not well documented at all but it seems to work rather well. The fair loss function is:

$$c^2\left(\frac{|x|}{c} - \ln\left(\frac{|x|}{c} + 1\right)\right)$$

It can be implemented as such,

```
def fair_obj(preds, dtrain):
    """y = c * abs(x) - c**2 * np.log(abs(x)/c + 1)"""
    x = preds - dtrain.get_labels()
    c = 1
    den = abs(x) + c
    grad = c*x / den
    hess = c*c / den ** 2
    return grad, hess
```

This code is taken and adapted from the second place solution in the Kaggle Allstate Challenge.

Log-Cosh Loss function.

```
def log_cosh_obj(preds, dtrain):
    x = preds - dtrain.get_labels()
    grad = np.tanh(x)
    hess = 1 / np.cosh(x)**2
    return grad, hess
```

Finally, you can create your own custom loss functions using the above functions as templates.

Warning: Due to API changes newer versions of XGBoost may require loss functions for the form:

```
def custom_objective(y_true, y_pred):
    ...
    return grad, hess
```

Share Edit Follow Flag

edited Oct 11, 2022 at 8:42

answered Jul 28, 2017 at 10:00

Little Bobby Tables
4,496 4 30 46

2 Many thanks josh! A quick comment on the log-cosh code; `np.cosh` can overflow for some inputs, using the identity `hess = 1- np.tanh(x)**2` will avoid these issues. – chepyle Jul 23, 2018 at 3:10

1 I think the fair loss function mentioned above is wrong (though the implementation is correct). According to kaggle.com/c/allstate-claims-severity/discussion/24520, the correct 'Fair Loss' function is given by `y = c * abs(x) - c**2 * np.log(abs(x)/c + 1)` . – kadee Aug 28, 2018 at 13:51

1 @josh thanks (and I meant you and not kadee) that makes sense. Finally, could you comment on the answer by @hbar137? Shouldn't you have "d = preds - labels" instead? – zkurtz Sep 18, 2018 at 12:42

1 @josh the line "grad = d / scale_sqrt" is not compatible with "the fact d is negative does not mean that there the first derivative becomes negative". The sign of the gradient is precisely the sign of d, since scale_sqrt is always positive. Empirically, I get vastly different and better results after flipping the sign. – zkurtz Sep 18, 2018 at 17:21

4 I think that for sklearn you should invert the arguments passed to the objective function... 1st should be labels, 2nd should be predictions... it does not make much difference in this case I suppose, but it can create confusion..also get_labels -> get_label – gabboshow Jan 16, 2019 at 13:20

|

1

I am running the huber/fair metric from above on ~normally distributed Y, but for some reason with alpha <0 (and all the time for fair) the result prediction will equal to zero...

Share Edit Follow Flag

answered Feb 20, 2020 at 17:58

Philipp_Kats
3,892 3 27 44

3

For the Huber loss above, I think the gradient is missing a negative sign upfront. Should be as

```
grad = - d / scale_sqrt
```

Share Edit Follow Flag

answered Aug 31, 2018 at 6:13

hbar137
39 2

1 Please add comments to the comments section of answers. Also, what makes you think this? I cannot see where the negative comes from. Thanks. – Little Bobby Tables Sep 18, 2018 at 13:32

If correct, hbar137 deserves upvotes for pointing out something so crucial, slightly more important than your standard comment. – zkurtz Sep 18, 2018 at 17:22

1 @zkurtz it would be if it were correct, but please checkout the derivative of the psuedo-huber. – Little Bobby Tables Sep 18, 2018 at 20:16

@josh You had the correct formula in the abstract sense. But if you were to put (-x) in place of x, consider that when taking the derivative you would need to apply the chain rule, leading to a minus sign on the derivative. hbar137 was correct at the time they wrote this, but since you fixed your solution (thanks!), this answer is no longer needed. – zkurtz Sep 18, 2018 at 20:33

Thank you both for your inputs and improving the answer!! – Little Bobby Tables Sep 18, 2018 at 20:49