Statistically valid way to convert training predictions to test predictions

80

posted in Quora Question Pairs 10 months ago

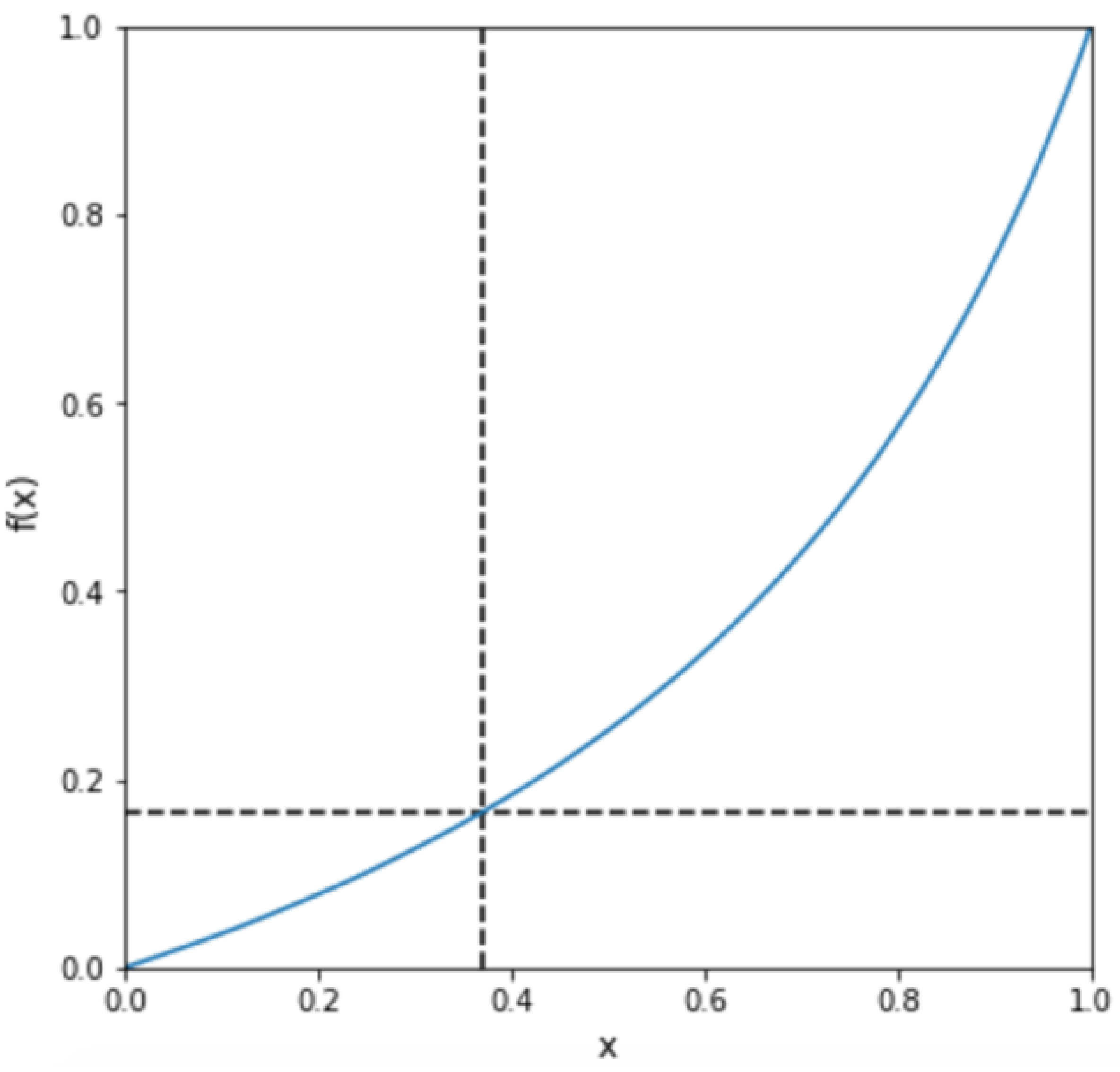
As many have noted, the class balance between the training sets and test sets appears to be different (37% positive for training, 16.5% for test). A lot of people have been asking what's the best way to convert training predictions to test predictions. I can't guarantee this is the **best** way, just what seems to me the most valid way from an stats standpoint.

let a = 0.165 / 0.37, b = (1 - 0.165) / (1 - 0.37)

function to convert is f(x) = a * x / (a * x + b * (1 - x))

For a full explanation of where this comes from - check out section 3 of my blog post about this : https://swarbrickjones.wordpress.com/2017/03/28/cross-entropy-and-training-test-class-imbalance/

Function looks like this (lines are to show f(0.37) = 0.165) :



Alternatively, you can train use a weighted version of the log-loss function as your objective :

a * y_true * log (y_pred) + b * (1 - y_true) * log(y_pred)

this is easy to do in tensorflow, a bit more of a mission to do in xgboost as far as I can see.

Other approaches I've seen talked about :

- oversampling - this approach is morally the same as oversampling, but is more exact, and will be quicker for complex models. Note that oversampling (and so this approach) has problems, your model will underestimate the variance of the oversampled class.
- linear transformation - e.g. can't find a single linear function that sends 0 to 0, 1 to 1 and 0.37 to 0.165

Hope this, or something like it will be useful to you.

Sheldon • (712th in this Competition) • 10 months ago • Options • Reply 0

Thanks for sharing. But I have a naive question, how to get the result of 16.5% positive for test.

sweezyjee... • (1034th in this Competition) • 10 months ago • Options • Reply 1

if you're asking why do we think 16.5% are positive in the test set, see here :

https://www.kaggle.com/davidthaler/quora-question-pairs/how-many-1-s-are-in-the-public-lb/comments

mezoganet • (409th in this Competition) • 10 months ago • Options • Reply 0

In XgBoost you have a parameter called **max_delta_step**. This can help when the class is unbalanced, according to the official doc.

https://github.com/dmlc/xgboost/blob/master/doc/parameter.md

> max_delta_step [default=0] Maximum delta step we allow each tree's weight estimation to be. If the value is set to 0, it means there is no constraint. If it is set to a positive value, it can help making the update step more conservative. Usually this parameter is not needed, but it might help in logistic regression when class is extremely imbalanced. Set it to value of 1-10 might help control the update range: [0,∞]

sweezyjee... • (1034th in this Competition) • 10 months ago • Options • Reply 1

Interesting, though I don't think they mean that it will help where there is a difference between the training set and test set - they're still optimising to log-loss, so it will have the same problem.

mezoganet • (409th in this Competition) • 10 months ago • Options • Reply 0

But we don't know what is the *real/full* test set... I don't think that oversampling is the key, wait and see the final LB.

And, remember the Allstates competition, when all was one and all fell down :-)

Jared Turk... • (3rd in this Competition) • 10 months ago • Options • Reply 0

I've never seen a dataset where max_delta_step has helped. I've also seen datasets much more imbalanced than this one, which really isn't that unbalanced.

CPMP • 9 months ago • Options • Reply 1

max_delta_step gave me 0.01 boost in the Bosch competition.

Laurae • 8 months ago • Options • Reply 4

`max_delta_step` is useful if you want to control how big the gradient updates can be. It allows to avoid exploding gradients, which also happen frequently with extremely unbalanced datasets (we are talking in magnitudes of 1:5000 or worse) and large learning rates.

There are also objectives prone to exploding gradients, like Poisson regression. They have a preset `max_delta_step` of 0.7.

In typical scenarii, not using a default of 0 for `max_delta_step` will hurt more than improve performance, even on unbalanced datasets, because you are essentially preventing gradient descent to perform its natural job to deep dive into a local minima quickly. Therefore, it should be never used unless you know what you are doing.

Also, this will not help at all if class distributions are different between train and test set. In a ranking scenario (Accuracy, AUC, F1 score, MCC, etc.), it might help because you are not allowing lots of (potentially wrong) observations to fly around quickly and become unreachable by the right observations during gradient updates (like Bosch competition as @CPMP mentioned).

CPMP • 8 months ago • Options • Reply 0

@laurae, I concur. in Bosch, unbalance was around 1:200, and max delta step was useful. Glad to see you again here :)

Bing Bai • (6th in this Competition) • 10 months ago • Options • Reply 9

In fact we don't need to oversample negative pairs for XGBoost. There is a parameter called "scale_pos_weight". If you set it to 0.360, then the share of positive examples in the training set comes to about 0.165.

John22 • 10 months ago • Options • Reply 1

My I ask why you choose 0.36? ... Since the original share is 0.37, shouldn't the weight be 0.445 to obtain a 0.165 share?

Bing Bai • (6th in this Competition) • 10 months ago • Options • Reply 3

Since the share of positive pairs in train set is 0.3692, and it in test set is 0.1746 (see this), if we want to make the weight of positive/negative pairs in train/test set to be the same, we have

(0.3692 * x) / ((1 - 0.3692) * 1) = 0.1746 / (1 - 0.1746)

Then we can get x is about 0.36

PavelTroshen... • (352nd in this Competition) • 10 months ago • Options • Reply 2

If someone looks for ways implement it in python xgboost, then look at this piece of code:

```
def kappa(preds, y):
    score = []
    for pp,yy in zip(preds, y.get_label()):
        score.append(a * yy * np.log (pp) + b * (1 - yy) * np.log(1-pp))
    score = -np.sum(score) / len(score)

    return 'kappa', float(score)

bst = xgb.train(params, d_train, 10000, watchlist, early_stopping_rounds=5,
verbose_eval=10, feval = kappa)
```

Maggie • (785th in this Competition) • 9 months ago • Options • Reply 0

Hello, it seems that the model is not optimized due to the new log-loss function, and it just show us the new log-loss.

timoeller • (356th in this Competition) • 10 months ago • Options • Reply 3

Schmidhuber would have set it was already done in the 90s. I found a paper from 2002, look at formula (4) on page 6 in "Adjusting the Outputs of a Classifier to New a Priori Probabilities: A Simple Procedure"
https://pdfs.semanticscholar.org/d6d2/2681ee7e40a1817d03c730d5c2098ef031ae.pdf