Featured Code Competition

PetFinder.my Adoption Prediction

How cute is that doggy in the shelter?

$25,000
Prize Money

PetFinder.my · 1,805 teams · 4 months ago

Benjamin Minixhofer
6th place

6th Place Solution Summary

posted in PetFinder.my Adoption Prediction 4 months ago

82

Edit: The code is now public.

First of all, thanks to Petfinder.my for hosting this great, meaningful competition. And congratulations to the winners! I did not expect to do this well in it. I started because I wanted to investigate the mismatch between CV and LB, but quickly managed to come up with a good solution and become first on the public LB. So naturally, I decided to stick with it. "Defending" this position was a challenging task and it motivated me to work a lot on this competition. And although I did not manage to do that in the end it was a great experience and I learnt amazingly much.

As a student, one challenge was the time I could spend on this competition. I tried to work on it for 1h - 2h everyday during the week, more on weekends, but that was not nearly enough to try everything I wanted to try.

I'll get right to summarizing my solution. You can see the code here. I'd recommend reading this summary first.

General

I have been stacking 5 models:

- 3 NNs using PyTorch
- one LightGBM model
- one FFM with xlearn

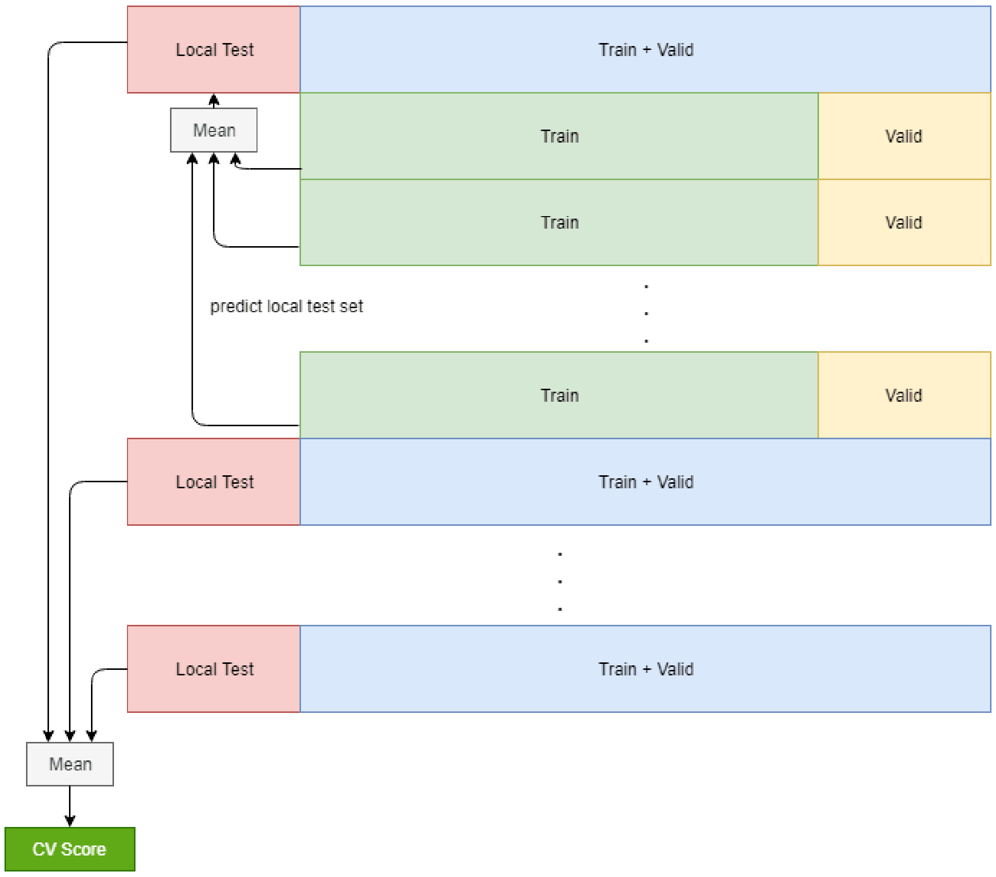
I used repeated stratified cross-validation with 5 splits and 5 repeats in my submission to reduce the effect of randomness, so each model was run 25 times overall.

I used another strategy for evaluating changes in the models.

Validation Strategy

Like in my kernel about validation in the recent quora competition, I used a kind of nested K-Fold CV. In an outer split, I split the dataset into a local test set and a train+valid set. The train+valid set is then also split into K-Folds and a model is trained on each fold. The predictions of each model on the data in the local test set are then averaged to get a QWK score.

This is repeated for every outer split. I made a figure to illustrate this.



The advantage of this strategy over others is that it takes into account the effect of averaging the test predictions of each fold. The same is done in the actual submission and because the correlation between folds has been low (especially of the NNs), the score would not be accurately estimated when using regular K-Fold CV.

Features

I used all categorical and numerical features available in the .csv files, the document sentiment magnitude and score, and the image metadata features commonly used in public kernels. Additionally, I:

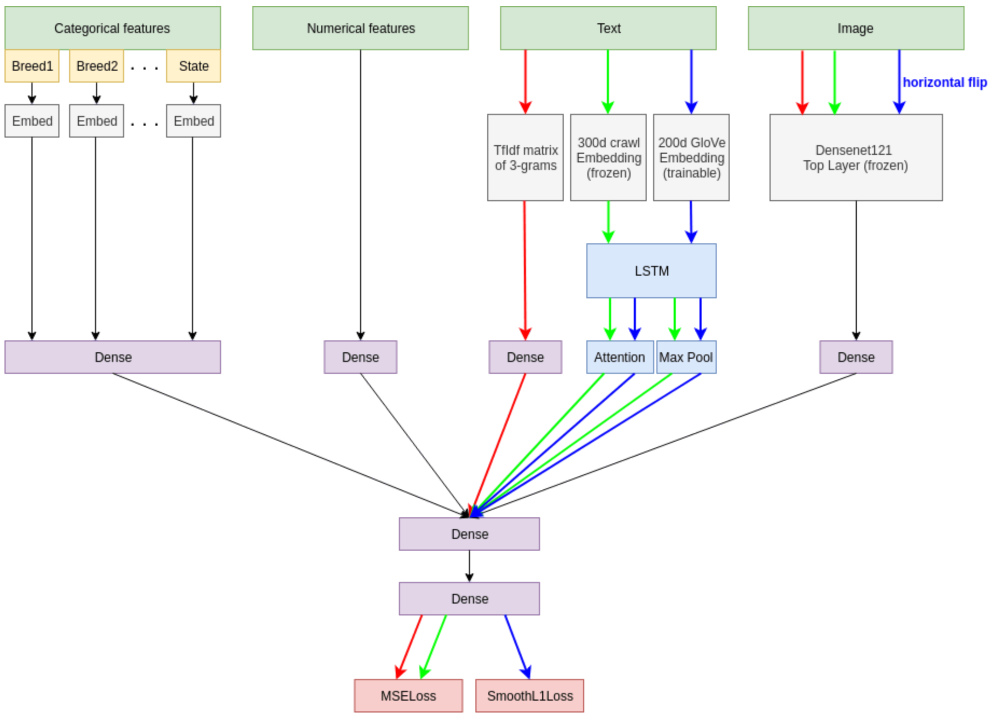
- created count features of Breed1, Breed2 and State. When using them in the NNs, they were logarithmically scaled.
- used GDP per capita of the state.
- binned RescuerID count into 10 quantile bins. This greatly increased my score (almost 0.01 on the public LB) and intuitively makes sense because rather than the exact amount of pets rescued, the *type* of rescuer should matter (e. g. whether it is corporation or a single person).

Neural Networks

All my neural networks use the same basic structure. To make the neural networks as diverse as possible, I modified:

- The loss function
- The image activations
- How the network treats text

You can see the shared structure between networks below. The black arrows represent connections in every network. The red, green and blue arrows represent connections only made in NN1, NN2 and NN3, respectively.



All network were trained using Cyclic LR and the Adam optimizer. I achieved the best results training for 10 epochs.

NN 1 - TfIdf matrix

My first and best performing network uses a kind of ridiculous structure. Text is encoded using 3-grams of all words with document frequency >= 2. That amounts to ~120k features. It is fed to the NN through a linear layer with 8 neurons, so there are 8 weights per 3-gram which amounts to ~1 million weights just in the text layer. To prevent this from miserably overfitting I used dropout of 80% and completely dropped the text features for 40% of all samples.

I used MSE loss to train it.

NN 2 - Non-trainable 300d crawl embeddings

The second network uses a regular RNN structure: Text is fed to the NN via an embedding layer pretrained on crawl embeddings and non-trainable. It is then processed using a bidirectional LSTM. Max-pooled output and output of an attention layer is then concatenated with the other features.

This network is also trained using MSE loss.

NN 3 - Trainable 200d GloVe embeddings + flipped image activations

The third network uses the exact same architecture as the second, but with 200d GloVe embeddings that I froze for the first 6 epochs and started training afterwards.

Also, instead of using the regular image features I used image features extracted from the same densenet121 model on flipped images to increase diversity. I also tried multiple other augmentations (rotating, zooming, ...) but none of them worked. Flipping the images significantly increased my score though.

This network was trained using SmoothL1Loss.

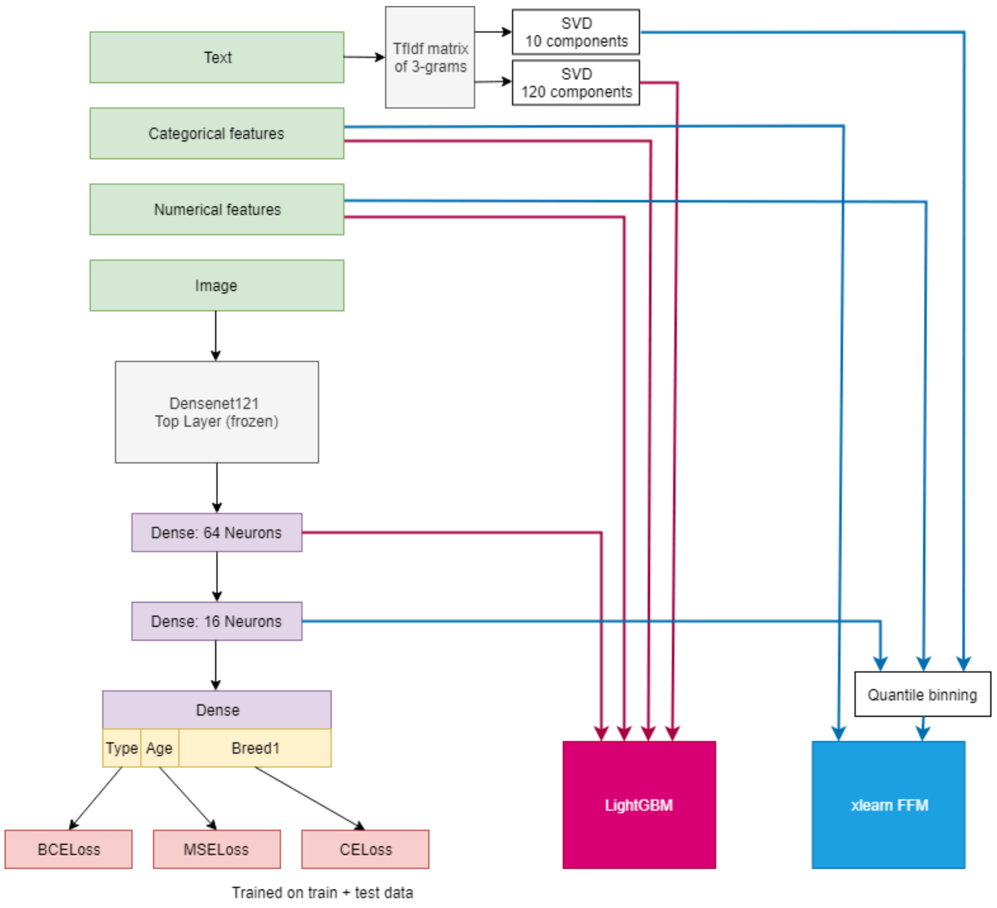
Other models

Additionally, I used LightGBM and xlearn. Text is encoded using SVD of a TfIdf matrix with 3-grams. The SVD has 10 components for xlearn and 120 components for LightGBM.

I found that raw image activations were too high dimensional to work well, so I trained an image activation extractor NN model. This model has the top layer activations from Densenet121 as input and tries to predict the Type, Age and Breed1 of the pet. It is trained on the train + test set.

This model has two hidden layers with 64 and 16 neurons. The activations of the first hidden layer are used as input for LightGBM, the activations of the second layer for xlearn. This greatly increased my score.

For xlearn, it worked best to treat all features as categorical, so I binned the image extractor activations and the SVD representation into 10 quantile bins. You can see the structure of the input features for the two models below.



Stacking

At first, I used a linear regression to stack the models, but it did not work well. There were mainly 2 problems:

1. Taking into account the increase when averaging predictions of each fold from the NN.
2. Mismatch between train and test distribution.

I solved the first problem by dividing the coefficients of the linear regression by the squared correlation between folds of the model. This somewhat accounts for the increase in score from averaging the predictions. I chose to use the squared correlation instead of just dividing by correlation because it slightly increased local CV score.

The second problem was that neural networks were weighted significantly lower than expected. Of course, I didn't want to overfit to the public LB so adjusting the coefficients manually was definitely a bad idea. What i came up with is setting the sample weights of the linear regression to the out-of-fold predictions of an adversarial validation model. As such, the samples which were seen as closer to the test distribution were weighted higher and the coefficients of the neural networks increased strongly.

When these two problems were solved, I arrived at a solid stacking strategy in which CV and LB score were closely related.

To convert the output of the stacking model to classes, I just followed the train distribution of classes. I tried the public OptimizedRounder and various genetic algorithms but they performed on par with following the train distribution, so I just sticked with it.

Trouble with the Leaderboard

Like everyone else, I had problems with the mismatch between CV and LB. My validation strategy largely solved mismatches when just tuning hyperparameters of the models, but some significant discrepancies remained.

Binning Age

Binning the pet age into 20 quantile bins greatly increased my CV score and it intuitively makes sense to me because rather than looking at the exact age of a pet, adopters will typically divide pets into categories like young, middle aged or old. I guess there is little difference between chance for a pet that is e. g. 6 months old and another one that is 8 months old. However, this decreased my score on the public LB significantly.

Extreme sensivity to small changes

Another problem I faced was that the LB score was extremely sensitive to small changes in input data. Just changing the number of SVD components from 120 to 256 decreased my LB score by 0.007, although it slightly increased CV score.

Using ordinal regression

This competition is an ordinal regression problem. There are different classes but the classes have an ordinal relation. So why not treat is as such? Neural networks can directly handle ordinal regression by using 4 output neurons and encoding classes as:

- 0: [0 0 0 0]
- 1: [1 0 0 0]
- 2: [1 1 0 0]
- 3: [1 1 1 0]
- 4: [1 1 1 1]

More information on this kind of encoding here.

The model can then be trained using binary cross entropy loss. The 4 outputs are summed to convert them back to a single scalar for each pet. This also greatly increased my CV score (by about 0.01) but decreased the public LB score.

A mistake increasing the LB score

For a long time, I used the following two lines of code in my kernel to create the gdp per capita feature:

```
train_df["gdp_vs_population"] = train_df["state_gdp"] /
train_df["state_population"]
test_df["gdp_vs_population"] = test_df["state_gdp"] /
train_df["state_population"]
```

When taking a closer look at this, you can see that it is wrong because the state GDP of pets in the test dataframe is divided by the state population of pets in the train dataframe. I didn't notice this until the last day in the competition. And I discovered that this wrong code gave me boost of an insane 0.007 on the leaderboard. So the LB score of my best correct submission was 0.485. This must have been just noise and it surely wouln't have held up in the private LB.

But there is another strange thing about this. Why did the code even run? At first glance, I would think that it has to fail because the test dataframe and train dataframe have a different length so division can not be done correctly. It turns out that pandas automatically pads the series with NaN values to allow for a division to work. And that it then only uses the elements of this series up to the length of the test dataframe to make the assignment operation work. In my opinion, that is confusing behaviour and should probably be removed.

Anyway, this is a testament for how noisy to public LB was, and how noisy the private LB probably also is because of the very little data that is available.

Options

Comments (42)

Sort by
Hotness

Click here to comment...

QuanTran • (492nd in this Competition) • 4 months ago • Options • Reply 3

Congrats on the solo gold and thanks for sharing your validation strategy! A quick question: there are several parts of your solution when you said the added tweaks improve your local CV but lowered your LB score, and you ended up didn't trust your local CV in the end. Do you have an insight on why your CV strategy does not work with this dataset, and what would be a piece of advice (maybe for future competition) to deal with this type of situation (where LB and local CV strategy are not consistent)?

Benjamin Mi... Topic Author • (6th in this Competition) • 4 months ago • Options • Reply 4

Thanks! And great question. One problem with my validation strategy is that in the case of 5 test splits and 5 "regular" splits I only use 80% * 80% = 64% of the data to train each model compared to the 80% of the data which the model in my submission is trained on in each fold. This might account for some mismatches.

However, especially in this competition, there is a lot of unavoidable noise because we have very few samples to work with.

In a future competition, when you find that CV and LB are not consistent, it is worth putting a lot of effort in making your CV as consistent as possible. Although my CV-tuned submission scored worse in the end, I only arrived at my "LB-tuned" submission because I was able to reduce the mismatches between CV and LB to a minimum and thus get a - in most cases - relatively accurate estimate of my LB score through local validation.

It's important to note that when writing of "LB-tuned" I mean a submission which scored well in CV and LB, as opposed to CV-tuned where the submission scored *better* in CV but worse on LB. I did not just submit to the leaderboard and see what works.

Dieter • (9th in this Competition) • 4 months ago • Options • Reply 4

Great write-up. I need to admit I was not able to construct a NN architecture similar to yours (although I tried hard after I used that successfully in the avito competition and there it worked great). Reason was I couldn't find a solution for the overfitting of the text part. Luckily I found a different NN solution which relies heavily on de-noising autoencoders as used by @mjahrer shared in this post . Will share more details about that in a few hours. I also want to ask how much would you say weighted the 3 different methods (NN,LGB,FFM) in your final solution?

Dieter • (9th in this Competition) • 4 months ago • Options • Reply 1

and btw I also used tfidf for text in the end :D

Benjamin Mi... Topic Author • (6th in this Competition) • 4 months ago • Options • Reply 0

You can see the exact weights here. I'd say NN and LGB were very important (my score with only LGB on the public LB was ~ 0.467). FFM helped but didn't contribute that much.

RDizzl3 • (9th in this Competition) • 4 months ago • Options • Reply 4

wow @bminixhofer this is an incredible solution and I was really rooting for you the whole time! This is a very well deserved finish and I will certainly be dissecting your solution. Thank you for the in depth post. Also, would you be willing to make your kernel public? If you haven't already :)

Congratulations again :)

Benjamin Mi... Topic Author • (6th in this Competition) • 4 months ago • Options • Reply 2

Thanks! I'm working on making the kernel public. I had to run it again in a separate kernel because I chose my final solution from my main kernel and it is not the most recent version there, so it would be confusing to publish it. But it should be finished soon.

I'm looking forward to your solution too if you'll publish it :)

hanfu • (319th in this Competition) • 4 months ago • Options • Reply 1

Thanks for your sharing! I noticed that you convert categorical features to labels using label encoding. Will ordinal encoding function properly in gbm tree models?