Imperial College London

COURSEWORK 2: ARTIFICIAL NEURAL NETWORKS

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Introduction to Machine Learning

Author:
Jiawei Mo (CID: 01899002),
Weitong Zhang (CID: 01904870),
Yunhao Zhang (CID: 01917956),
Zhenghui Wang (CID: 01988162)

Date: March 30, 2021

Contents

1 Model Description

In part 2, we decided to use the PyTorch neural network library in the code. The given example dataset (housing.csv) contains 16512 observations on ten variables. The ten variables are longitude, latitude, median housing age, total rooms, total bedrooms population, households, median income, ocean proximity and the output variables ”median house value”. To implementing a neural network architecture, there are mainly three parts in our model generation, which is Preprocessor, Constructor and Model-training method.

1.1 Preprocessor Method

Preprocessor method is the beginning step. As these data might miss values in some of the columns, we evaluate the number of data in each column to find out the column that has empty values. There currently exists multiple ways of handling missing data:

1. One existing method sets the missing data with default value 0, which is unreasonable in this dataset. It could generate noise data.

2. Another one is to remove the observation with empty value. We found that there are hundreds of empty values in the dataset. It will loss sample size and further drop generalization if we directly drop rows with empty values.

3. Another method is using the regression equation to find out a suitable value for the missing data. Unfortunately, it costs too much computation power to generate values.

In summary, we decided to fill the missing value with the average value of its column. It could maintain the basic features of these variables and avoid to generate noise data.
As the only one textual variable in this dataset didn't miss any values, we decided to apply the "fill the nan" process before the "one hot key". The one-hot key strategy applied to all the input dataset could convert all the categorical variables to numbers and assign them the same weight.
And then, it's going to be the step of normalization. There is a method that normalizes the value by dividing the max value in each column. However, dividing max negative value generates a range greater than 1. So, the program applies the formula below to map each column in the range of [0, 1].

$$\text{Normalised value} = \frac{\text{value} - \text{min}\_\text{value}}{\text{max}\_\text{value} - \text{min}\_\text{value}}$$

In this case, the value of all the variable is between 0 and 1. it can ensure every variable contributes the same impact to the parameters. And finally, we convert all variables' datatype to float64 for the convenience of data process.

1.2 Constructor Method

There indeed are added input parameters (learning rate and decay) to enhance convenience. The max and min values of each column belonging to the training data are stored for further normalization of prediction. Also, the one-hot key table is reserved. The best model and hyper-parameter will be addressed after our evaluation and the final run (on CPU). We are using GPU to rocket speed up but the lab test rejects model trained on GPU. The lists of hyper-parameters for searching will be commented out in the final version.

1.3 Model-training Method

In Model-training, we define four hidden layers in the model which have 1024, 256, 64, 16, nodes and the activation function ReLU is adopted to solve this price prediction problem. The reason that we designing model with this network architecture is that input has around 16000 features and only one output, also it's a regression problem.

2 Evaluation Setup

2.1 Prediction

Processing the input data at the beginning, recovering with max, min values, and one-hot key table to normalize data with respect to training set. Then load model to predict and multiply the result by 1000 to remap $ 20k to $ 20,000.

Training_RMSE	Validation_RMSE	Training_R2	Validation_R2
50887	58750	0.805	0.756
47943	51953	0.829	0.791
46512	51314	0.839	0.798
46153	54050	0.841	0.786
49224	48359	0.820	0.819
46564	53707	0.838	0.782
44874	56676	0.849	0.761
45985	55675	0.841	0.780
43834	51453	0.856	0.805
45114	51452	0.849	0.799

Table 1: RMSE & R2 value when learning rate = 0.02, epoch = 2000, decay =0.01

2.2 Evaluation

In evaluation, there is a score function which take the prediction result as input and generates three key outputs which are Mean Squared Error(MSE), Root Mean Squared Error(RMSE) and Coefficient of determination (R2). Meanwhile, the model uses cross validation to generate three scores of each validation under each group of hyper-parameters. We output information, ['lr', 'max_epoch', 'decay', 'train_mse', 'vali_mse', 'train_rmse', 'vali_rmse', 'train_r2', 'vali_r2'], to a text file and read this file to evaluate performance. We want RMSE to be small enough and R2 to be close to 1. If R2 is close to 0, that means the prediction is more or less the mean value of the validation set. A negative R2 indicates that the trained model is worse than just taking the average value to predict the price. A fold trial using learning rate = 0.05, epoch = 2000, and decay = 0.0001, results in both negative R2 in train and validation (-1.212369 & -1.209810) and RMSE of 172259 and 170225, revealing that the model is not stable.
The Table. 1 shows the 10-fold cross validation result of the best hyper-parameters. If the model iterates 5000 times, in Table. 2, the training R2 does approach 0.9 but the validation score would drop, leading overfitting. The RMSE will also depict the problem. Also Figure. 2 reveals it since the training loss is decreasing but validation loss shows a trend of increasing.

3 Hyper-parameter Search

To pick the most optimal set, The program used an approach similar to the Algorithm 1, GridSearchCV, supported by Scikit-learn package but cannot be used for wrapping torch tensor without skorch package. Our method applied an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm. Set up some values to learning rate, max epochs and weight decay, and applied a greedy loop for each hyperparameter. Based on the performance of each model, find out the best model with the lowest RMSE value. Further, the 10-fold

Training_RMSE	Validation_RMSE	Training_R2	Validation_R2
42431	55918	0.866	0.758
33646	56135	0.915	0.782
36618	55067	0.900	0.766
44450	53376	0.851	0.802
29808	54881	0.933	0.783
32934	59322	0.919	0.731
33759	54414	0.914	0.786
39556	60158	0.883	0.722
35337	64384	0.907	0.669
27154	54944	0.945	0.761

Table 2: RMSE & R2 value when learning rate = 0.02, epoch = 5000, decay =0.01

cross validation method is applied and every single run's scores are recorded and dumped into a text file for examination. We take mean scores of each validation, which helps to examine the found hyper-parameters. For instance, occasionally one fold of 500 epoch run could generate a good RMSE as the same as 2000 epoch run. However, the average of 10 folds would reveal the difference.
We do store best model for programming purpose while searching the hyper-parameters. Actually, after evaluate the resulted text file, we put the best hyper-parameters to run again and the model will be overrode.

Algorithm 1: GridSearchCV - Find the best optimal hyperparameter set

```
Result: The best model with min_RMSE
for i in learning rate do
    for j in max epochs do
        for k in weight decay do
            for 10-fold cross validation do
                current_model ← model
                prediction - y_pred and y_actual
                Compute score - val_rmse
                if val_rmse<small_rmse then
                    small_rmse ← val_rmse
                    best_model ← current_model
                end
            end
        end
    end
end
```

The first trial for these three parameters is:
Learning rate = [0.001, 0.02, 0.05]
max epochs = [500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 5000]
weight decay = [0., 0.01, 0.0001]

Figure. 2 shows the comparison of training and validation among those values. It returns the current best model using hyper-parameters, learning rate = 0.02, epoch = 2000.0, and decay = 0.01, leading the lowest RMSE as 48359.149079 on the 10-fold cross validation. The assumption is that 5000 epoch might incur over-fitting as the number of iteration is so high. Learning rate of 0.001 refers to relatively small steps and contrary the overshooting of 0.05. The weight decay of 0.01 could be appropriate value to drive learning rate to decrease in an ideal extent so that the model converges well.

Further we try few learning rates:
learning rate = [0.005, 0.01, 0.03, 0.04]
max epochs = [2000]
weight decay = [0.01]

The following Figure. 1 will disclose the performance of different models under different learning rates. It's apparent that the learning rate 0.02 and 0.03 decrease the loss well on both train and validation set.

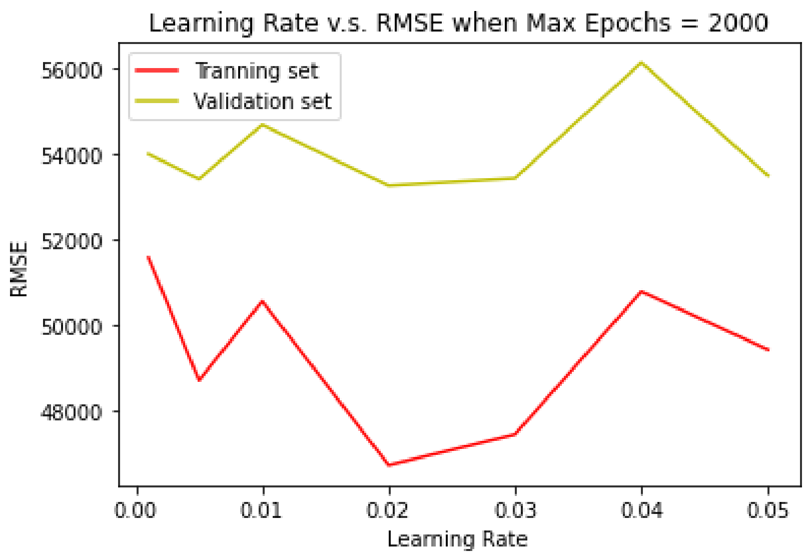

Figure 1: Learning Rate v.s. RMSE when Max Epochs = 2000

4 Final Evaluation Of Our Best Model

We obtain the best hyper-parameters with learning rate = 0.02, epoch = 2000, and decay = 0.01, which result in mean scores, train RMSE = 46709, validation RMSE = 53256, train R2 = 0.84, and validation R2 = 0.79, leading around RMSE = 44000 on the lab test. Each fold's scores are shown in Table. 1 and Table. 2 claims an overfitting with 2.5 times iteration.

5 Appendix

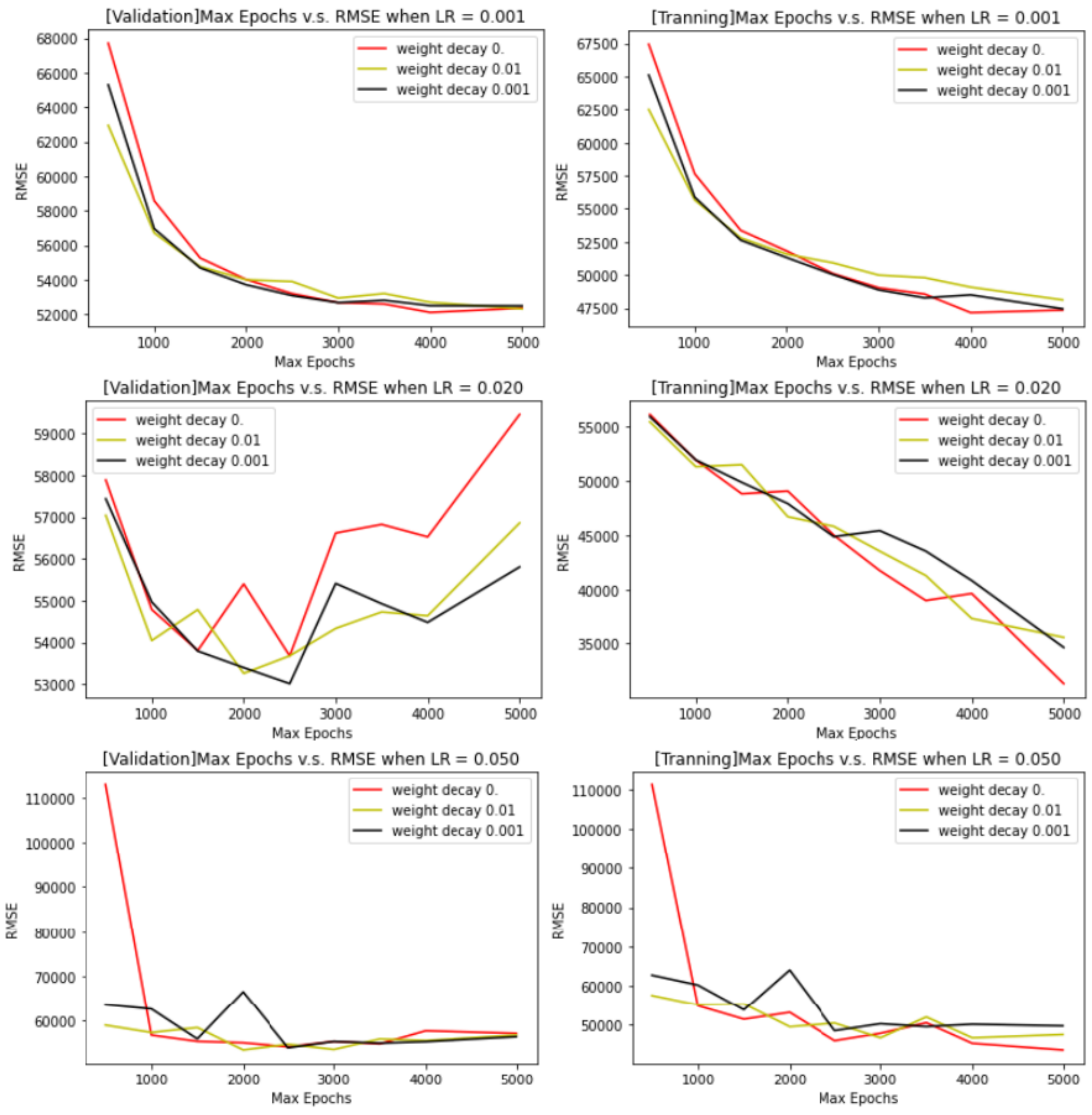

Figure 2: Max Epochs v.s. RMSE