Machine Learning 2019 Final Report

Team: 3HandsomeMen!

Data preprocess

Different scale among features might have negative impact on the performance, because features with larger scale would dominate over others. Therefore, we standardize all the features to make each of them with zero mean and unit variance, by subtracting mean then dividing by variance. Furthermore, three target value also have very different scale, so we standardize them for all the experiments we made.

Feature analysis

There are total 10000 features in this dataset, so it takes significant time to apply any machine learning technique, even for linear regression. We analyze the dataset to determine which features are better, then use them in following experiments and reduce the redundant features. We fist divide all the features into 1000 groups, which has 50 features, then apply linear regression to fit three target value, evaluating by 2 tracks in the competition, the results are below (blue line represents training data, red line represents validation data):

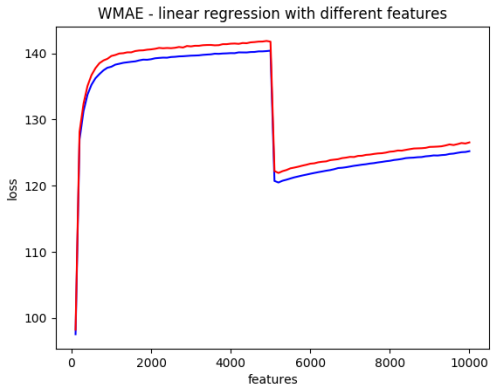


Figure 1. WMAE with different features

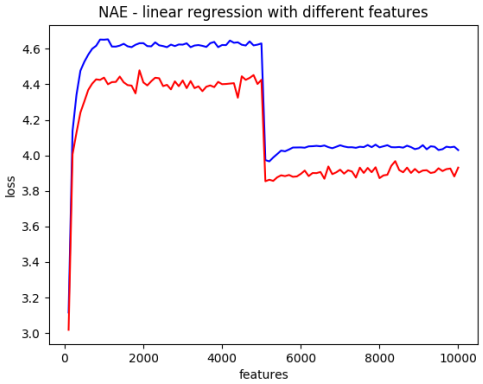


Figure 2. NAE with different features

In the figures, we discover that the first few features has much better performance than others, but overall, the last 5000 features are better than the first 5000 features, how interesting ! Next, we want to know how many features we use would lead to better performance. We experiments with first X features and apply linear regression, evaluating by 2 tracks, the results are below:

Figure 3. WMAE with first X features

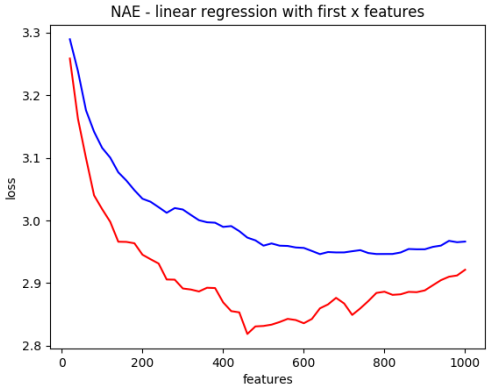


Figure 4. NAE with first X features

In figure3, 4, we discover that when we pick x from (200, 400), would have best performance (low error rate and little overfitting). As a result, we do experiments with various machine learning technique on first 200 features, and made a huge improvement in performance, compared with using all 10000 features.

Methods & Experiments

RNN Model

For the first 5000 dimensions, which are MSD as a function of time interval, are related to time sequence. We try to train our data with RNN model, whose connections between nodes form a directed graph along a time sequence. I use long-short term memory (LSTM) to construct our model, and reshape our 5000-dimensional input data into (50, 100), with 50 epochs, and WMAE customized loss function. The result of epoch versus WMAE loss is shown below.

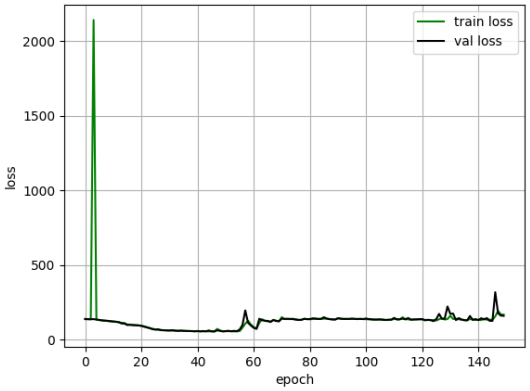


Figure 5. RNN model for the first 5000 dimension

Our machine seems to learn little from data, maybe it's challenging for LSTM to memorize the feature when the input sequence is too long. According to the linear regression analysis above, we know that the first 200-dimension data have better performance. We try to train our model with the first 200-dimensional data in the shape of (20, 10). We also try to reverse the order of the 200 data so that the data in the front dimension will influence the prediction more. The result of epoch versus WMAE loss of the two models are shown in figure 2 and figure 3 respectively. The result is quite good, and the performance of RNN model with reverse order is slightly better than another. The final WMAE loss of validation data is 45.05, and the model trained by NAE loss function has a 0.515 score.

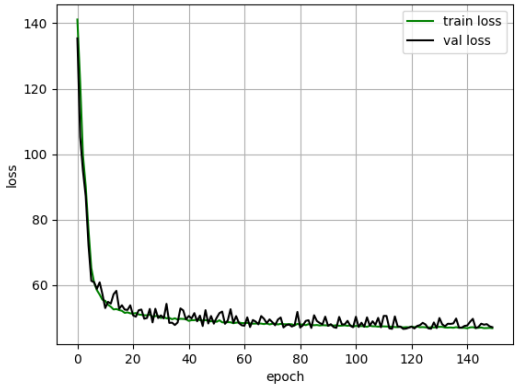


Figure 6. RNN model for the first 200 dimensions

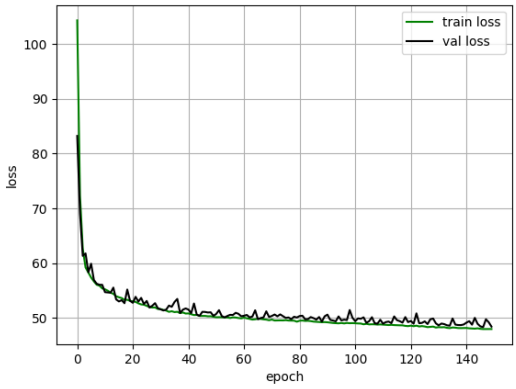


Figure 7. RNN model with reverse order

NN Model (Full-connected)

Now we know that the first 200-dimensional data can have good prediction. We built a NN model to see whether the data is really relative to time sequence. The model has 200-dimensional input, 300-500 hidden layer, 3 final output, with 150 epochs and WMAE loss function. The result of epoch versus WMAE loss is shown in figure 8. The validation loss isn't as stable as RNN model, and there is overfitting after 60 epochs. The same problem happened in the model trained in NAE loss function.

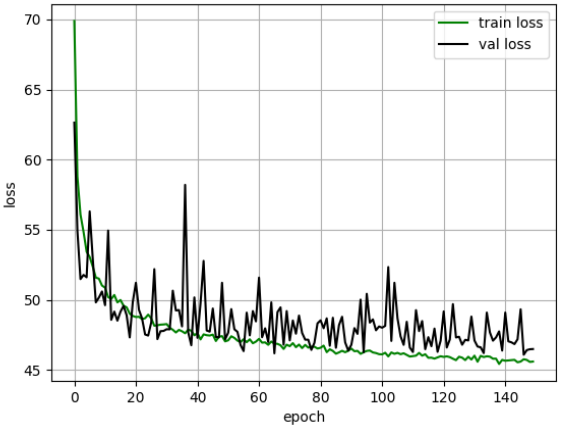


Figure 8. DNN model for the first 200 dimension with reverse order

RNN + NN Model

The remaining 5000 dimensions are composed of 50 groups of VAC data, and each group is also related to time sequence. Hence, we trained the VAC data with RNN model separately, and generate 50 RNN model. Each model has about 90 in WMAE loss and 0.8 in NAE loss, and the histogram of WMAE loss of the 50 models is shown in figure 5. After generating 51 different predictions using 50 VAC RNN model plus one MSE RNN model, we construct a neural network to do aggregation to decide the final prediction. The result of epoch versus WMAE loss of the two models are shown in figure . The final result is better than the 50 VAC model, however, overfitting happened and the validation error is quite unstable. We also try to do linear aggregation, but the result is even worse.

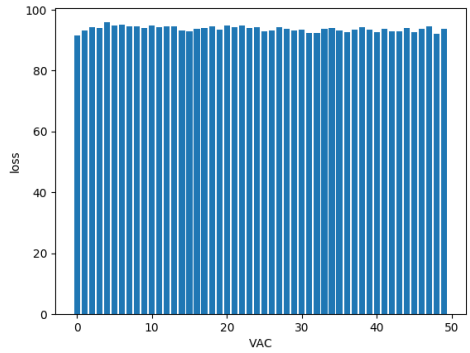


Figure 9. loss of the 50 RNN model

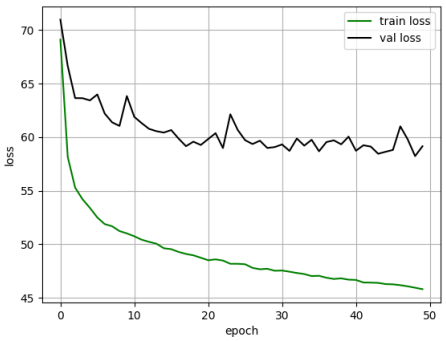
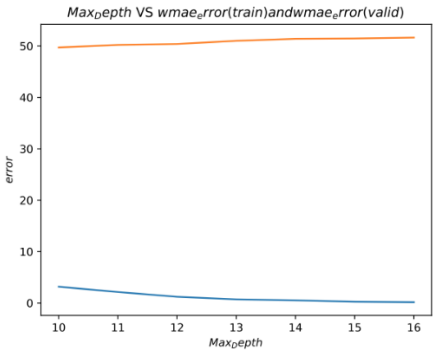


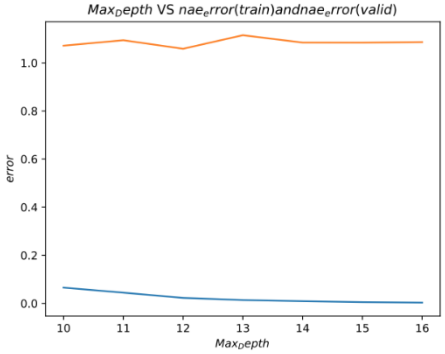
Figure 10. total NN aggregation

XGboost Model

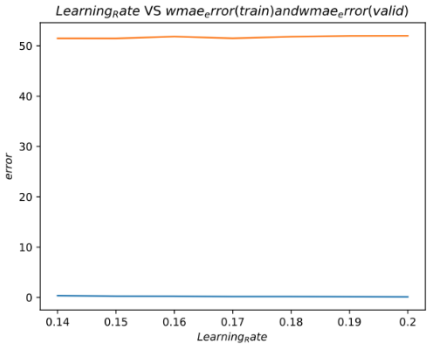
In this method, we focus on three parameters of XGboost model: ***max_depth, learning_rate, n_estimators.*** First, we keep trying to find out the performance of these parameters. Thus, we suppose to use first 2000 of training data to implement our experiment. According to previous experiment, we know that first 200 of features might dominate the performance. As a result, our experiment and final output only use first 200 features of training data. For lines on figures below, blue line represent Ein of training data and orange line indicates validation error.

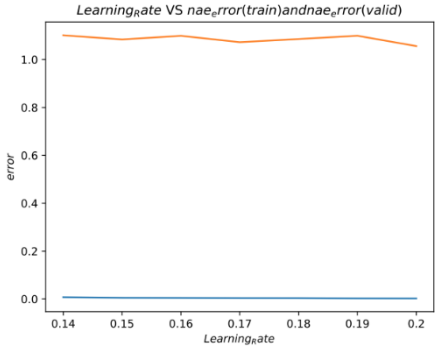
Experiment on max_depth:





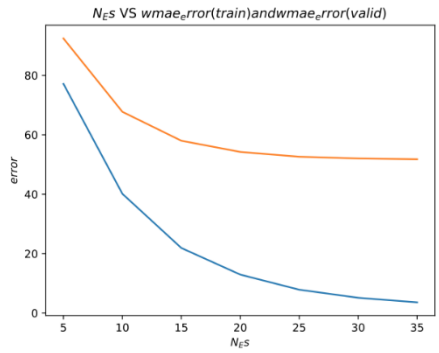
parameter max_depth, apparently, doesn't make obvious difference in the both tasks. For WMAE, we can observe that the deeper the trees are, the more probable the model might overfit the training data. As for nae, we do not observe this situation. Experiment on learning_rate:

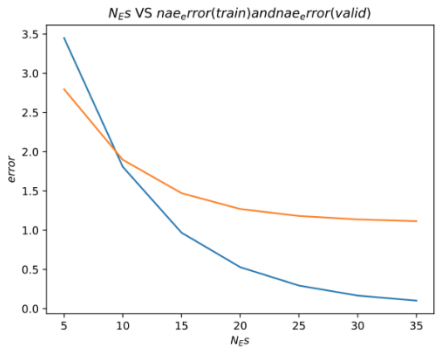




parameter learnin_rate doesn't have effective performance in both task in our experiment.

Experiment on n_estimators:



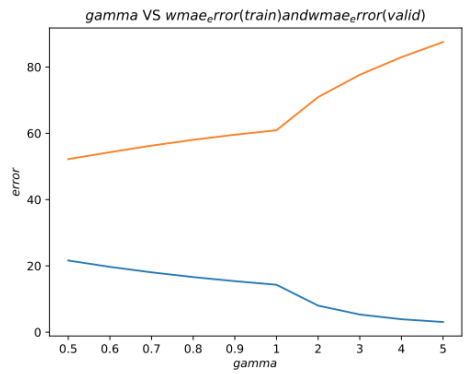


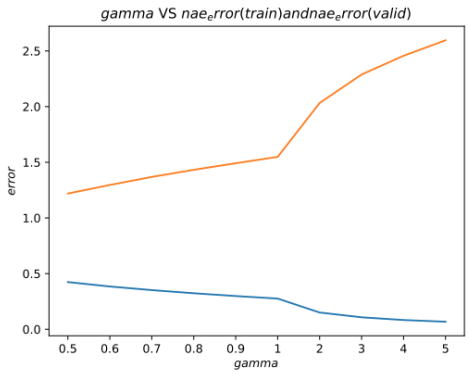
parameter n_estimators, in our experiment, obviously dominate the performance in both of the tasks. The result shows that the size of the forest is quite important, if we generate more trees in the process, we can get more precise prediction.

Concluding information all above, we try to implement results from our experiment on generating models for the tasks. Our result by implement XGboost with max_deph=15, learning_rate=0.15 and n_estimators=1000 is very well, let our team occupy second place in private and third in public of track1. Though it seems that XGboost gives well performance on track 1, it doesn't perform well on track 2. Another discovery is that while XGboost method perform quite well, the model always seems to be overfit training data in contrast with other method.

RBF kernel ridge regression Model

Another method we suppose is RBF kernel ridge regression, which was taught in the course. In experiment of this method, we also only focus on first 200 features of the data. Mainly, we focus on the performance of gamma. For the experiment result below, blue lines and orange lines represents training error and validation error respectively.





we discover that in this data, RBF model will overfit after gamma exceed 1 and no matter which track we experiment on, the error grows very fast. As a result, our final result implement kernel ridge regression is not as well as former method, a main reason is that the time kernel ridge regression model takes to fit the model is quite long. Therefore, we don't have time to adjust the parameters to give us a better result.

Recommendation for each track

track 1 – WMAE (public: 41.854501 , private: 42.492305)

Data process: Select the first 200 features to be training data, and normalize all data and denormalize prediction at the end of the process.

Machine learning technique: We choose XGboost model to be our best method, with learning rate = 0.15, n_estimator=1000, max_depth=15. we fit every target once to make all three target features

Pros: XGboost method works quite well in this track, we even didn't use any of domain knowledge, simply use machine learning method and data processing. It's easy to implement, but also works efficient.

Cons: In the beginning of experiment, we can't adjust our parameters well since only parameter n_estimator gives us obvious improvement. Another problem is that although the outcome seems to be very well, the is still a huge gap between validation error and training error (Ein), which indicate there is a problem of overfitting training data.

track 2 – NAE (public: 0.424 , private: 0.461)

Data process: We select the first 200 features as training data, and reshape to (20, 10), where 20 represents time sequence, and we also add quadratic term as feature transform, so the final input for each sample is in shape of (20, 20)

Machine learning technique: We use a two-layer LSTM model, 1024 units per layer, following by one-layer NN (1024 * 3), trained by Adam optimizer with learning rate 0.001 for 100 epochs, batch size = 125, We use Absolute Distance as our loss function.

Pros: The method can achieve good performance in both tracks, and has no problem of overfitting. And during training, we can select the best one among all epochs.

Cons: Need GPU for accelerating training process, and it takes some effort and time for parameter tuning.

Cooperation

student id / name	contribution
B40901069/林志皓	Data analysis, do experiments on linear regression, DNN, RNN
B05902050/黃子源	Do experiments on DNN, RNN, total training for all data
B05902134/楊仁傑	Do experiments on XGboost and RBF kerenl ridge regression