Fractional Brownian Motion

June 23, 2019

陳君彥, b04703091 b04703091@ntu.edu.tw
劉育嘉, b06902008 b06902008@ntu.edu.tw
黃柏豪, b06902124 b06902124@ntu.edu.tw

Division of Work

陳君彥, b04703091:

- Data initial processing and calculation tool creation.
- Initial creation and testing of XG-boost, LightGBM, and Random Forest Models

劉育嘉, b06902008:

- Indepth testing of tree based methods
- Testing and Creation of select feature models.

黃柏豪, b06902124:

- Creation and testing of neural network based models.
- Creation and testing of blending based models.

1 Introduction

The goal of this project is to reverse learn model parameters used to simulate a fractional Brownian Motion [1] simulation. The parameters used to run this simulation were alpha, mesh size, and penetration rate. We used 2 methods of evaluation, one with "Weighted Mean Absolute Error" (WMAE), and one with "Normalized Absolute Error" (NAE)

$$WMAE(Y,\hat{Y}) = \frac{1}{n_{samples}} \sum_{i=1}^{n_{samples}} \sum_{j=1}^{3} w_j \mid y_{ij} - \hat{y}_{ij} \mid, NAE(Y,\hat{Y}) = \frac{1}{n_{samples}} \sum_{i=1}^{n_{samples}} \sum_{j=1}^{3} \frac{\mid y_{ij} - \hat{y}_{ij} \mid}{y_{ij}}$$

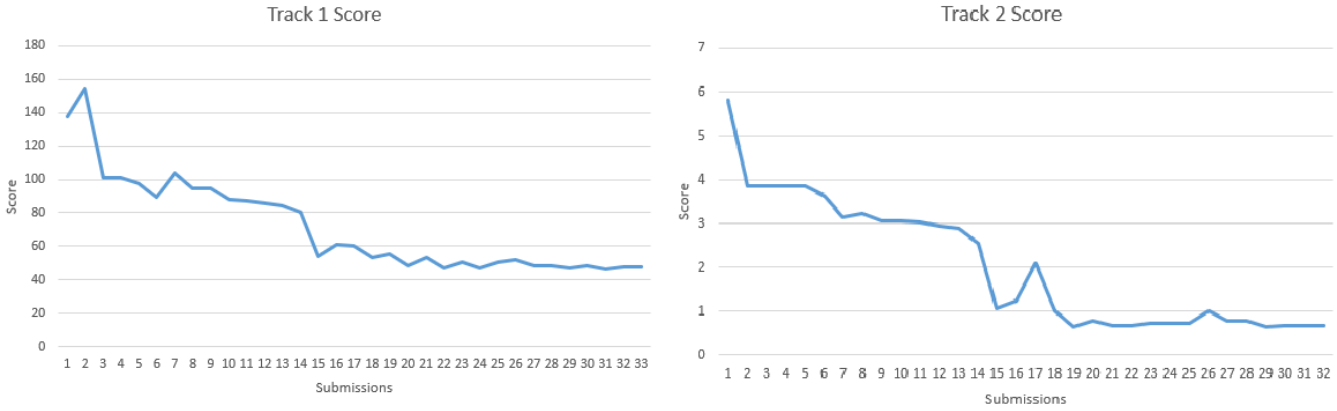

Figure 1: Submission scores of our models.

2 Features

2.1 Original Features

Our training dataset consists of 47,500 simulations with 10,000 features each. The first 5,000 features represented the mean-square displacements (MSD) of our particles ordered from time = [1,5000]. The second 5,000 features were 50 sets of velocity auto-correlations (VAC) calculated in different methods, ordered by the early VAC being more representative of instataneous velocity, with later VAC being closer to average velocity. Each sets consists of 100 calculation using the respective VAC, with time intervals from t = [1,100].

Our testing sets consists of 2,500 simulations, half of which is tested in the public score, and half hidden in the private score.

2.1.1 Feature Importance

In the interest of reducing the time need to train our models, we took a look at the feature importance determined by our earlier models and selected those that had importance higher than a given threshold. Through several rounds of testing, we found 10^{-4} to perform the best. This allowed us to greatly increase the training iteration count and deepen our original model, giving us increased accuracy with much lower training times.

Something of particular to note is the high reliance of a select few features for each parameter. Due to the lack of professional knowledge in this field and the lack of raw data, we could do little to take advantage of this except to reduce the dimensionality. However we expect that further studies can be improve by generating more features that are centered around these specific features.

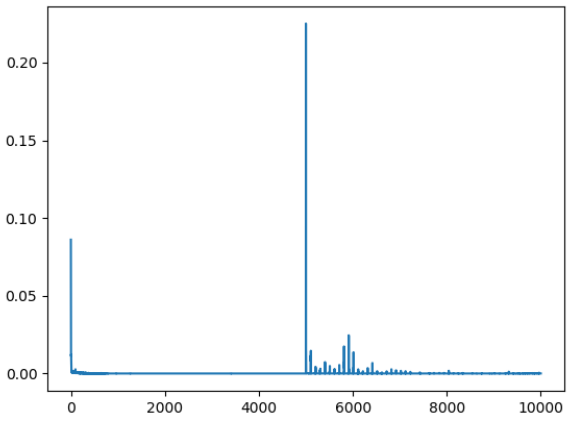

Figure 2: Feature importance of alpha.

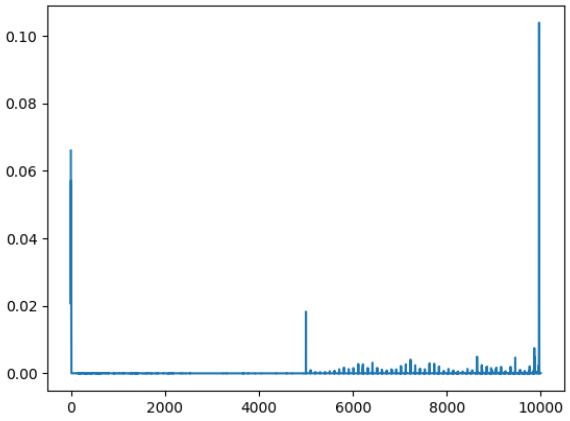

Figure 3: Feature importance of mesh size.

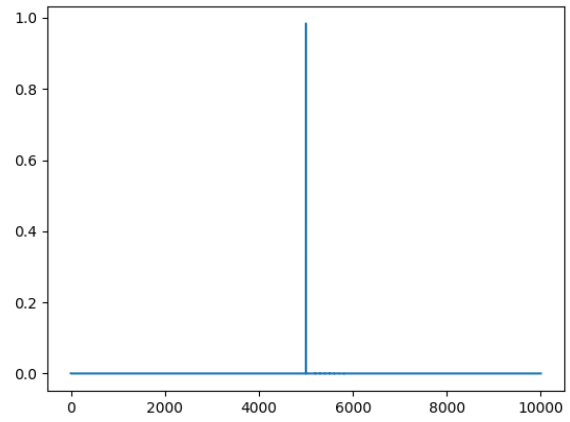

Figure 4: Feature importance of penetration rate.

2.2 Additional Features

To gather more physic based features, we also extrapolated some of our own data.

2.2.1 Average and MinMax

We were interested in having our model be less overfitting and took feature based methods to reduce the feature count. We tried taking the average of every 10 features, which would result in the

mean displacement or velocity over a larger time period. We also tried taking the minimum and maximum of each 10 features, which in essence was is a form of data pooling.

2.2.2 Square Error on Linear Regression

We assume a particle without external force will go straight forward with constant speed, so the force generated by collision might have something to do with the difference between a linear line and its real value. Therefore, we performed a linear regression compute the variance of those 5000 time intervals, substituting them for the original 5000 features.

2.2.3 Alpha feedback

Our models performed exceptionally well on alpha, and so we decided to feedback the results of our third parameter prediction back in as a feature for our other two predictions, giving a slight boost to performance.

3 Individual Model Experiments

3.1 Tree based models

3.1.1 Random Forest

We used the models provided in the python package Sci-kit Learn [2] for our random forest, with bootstrap enable, estimator count of 50, and a max depth of 9. The following shows the results of our experiments.

Tree count	Alpha	Mesh Size	Penetration
50	65.72	31.92	1.96
100	64.74	30.69	1.81
200	64.06	29.93	1.74
300	61.02	27.20	1.54
500	58.67	25.10	1.40
1250	55.07	22.40	1.23

Table 1: Performance on track 1 for random forest

As seen in Table 1, we achieve moderate performance by simply throwing in the original features into a random forest model. The performance gradually increased with each additional tree count, though the time required to train them starting to get out of hand. Each tree required around 1.5 minute to train when parallelized, meaning training each model for all three features required over 3 hours for each incrementation. Hence we tried it out with a reduced feature set selected using the previously trained model's feature importances.

Tree count	Alpha	Mesh Size	Penetration
50	67.31	30.69	3.13
100	67.42	30.60	3.12
200	67.35	30.65	3.11
300	67.35	30.64	3.11

Table 2: Performance on track 1 for random forest with reduced input.

However, as seen in Table 2, the reduced feature set performed poorly, while giving us a very low training time, it failed to beat even the 50 tree count on the regular RF model at 300 trees. We gave up and tried a different approach, reducing the dataset by picking averages of every 10 features.

Tree count	Alpha	Mesh Size	Penetration
50	64.57	23.98	2.37
100	64.56	23.94	2.36
200	64.56	23.86	2.36
300	64.60	23.85	2.35

Table 3: Performance on track 1 for random forest with averaged features.

The results as shown in Table 3 shows that this performed better than the feature selection model, and slighly better than the original model, with a combined score of 90.81 on track 1 and 2.29 on track 2. However, we notice that stop in decrese of the validation error, and recognized that further training would do the model no good.

3.1.2 XGBoost

We tried the very popular XGBoost [3] often used in machine learning competitions. XGBoost is a gradient boosting tree model that has an emphasize on reduced tree depth and higher parallelism. This allows us to train a model much faster and allows easier batching on the massive dataset.

Notes	Track 1	Track 2
base	137.23	6.22
+ filtered data &alpha feedback	48.86	0.66
+ auto tree method	43.49	0.60
+ tweedie regularizer	41.70	0.38
+ boot round = 5	40.85	0.33

Table 4: Performance of xgboost with each additional parameter tuning.

As seen in Table 4, the xgboost relies heavily on custom tuned parameter, with the base parameters model performing really bad. However, with careful tuning, and the filtered data fed into our model, xgboost provided very respectable data on the validation set. After submitting the results, we found out that the final rounds of tuning had led to the model being overfitted, with the submission score actually being worse than the previous round. We thus stopped our experiments on xgboost here and moved on to a different model.

3.1.3 LightGBM

LightGBM [4] is a very new contender to the scene, with a focus on limiting leaf count rather than xgboost's method of limiting tree depth.

In our instant, lightgbm performs much better with base parameters, with learning rate set to 0.01. As seen in Table 5, it is already in clear competition with xgboost. We also went through the same process of using the predicted alpha to feed back into the model, as well changing to use 'dart' as a boosting method with more iterations. It can be seen from Table 6 that the submission score is also dropping at a steady rate, but then shows signs of overfitting with the last submission, we then started performing regularization to combat the issue.

Notes	Track 1	Track 2
base	56.80	1.52
+ alpha feedback	55.87	1.20
+ 1000 iterations + dart	38.25	1.07
+ 2500 iterations	30.84	0.36

Table 5: Validation performance of lightgbm with each additional parameter tuning.

Notes	Track 1	Track 2
base	60.76	2.94
+ alpha feedback	59.87	2.10
+ 1000 iterations + dart	48.23	0.92
+ 2500 iterations	50.48	0.86

Table 6: Submission performance of lightgbm with each additional parameter tuning.

We tried using the square error on linear regression as a feature, and averaging them out, cutting our feature set down to one tenth of the size, and as seen in Table 7, shows that the overfitting issue has been somewhat allieviated.

Notes	Validation	Submission
500 iterations	45.52	49.17
1000 iterations	43.27	48.13

Table 7: Submission performance of lightgbm with each additional parameter tuning.

3.2 Neural network models

We used a dense neural network of size 2048-50-1 to predict our three parameters. As the number of features is too big for a neural network, the features were first passed through an autoencoder of size 5000-2048-1024 and train with 'relu' activation on a 5000-2048-1024-2048-5000 model.

```
def encoder(x,
    actvation_list = ['relu', 'relu'],
    layer_list = (2048, 1024), # dimension
        for each layer
    name = 'temp', # to specify model file
    use_old = False): # load existing model
        or not
```

3.3 Blending

4 Conclusion

References

[1] Wikipedia contributors, "fractional Brownian Motion," 2019, [Online; accessed 22-June-2019]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Fractional_Brownian_motion&oldid=895902440

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[3] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 785–794. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939785

[4] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. USA: Curran Associates Inc., 2017, pp. 3149–3157. [Online]. Available: http://dl.acm.org/citation.cfm?id=3294996.3295074