Factor-based Machine/Deep Learning Strategies for China A-share Markets

Supervisor: Prof.Hongsong CHOU & **Dr.Ling LONG**
He Congxin (21024425) Boosting Model
Chen Hongxi (21025223) Deep Learning Model
Tang Jiarui (21016753) Optimization

1 Introduction

There have been a growing number of academic studies on using machine learning and deep learning methodologies to forecast financial assets' returns over different time periods and to construct optimal portfolios with the goal of achieve risk-balanced investment returns. In this project, we want to stay on the practical side and focus on investment strategies that can leverage such modeling and investment methodologies. Specifically, we focus on:

(1) with pre-selected weekly factors, we apply in-depth several machine/deep learning models to forecast stock forward returns, rigorous in-/out-sample testing shall be used, say k-fold cross validation.

(2) with step-1 forecasting result, we performed portfolio optimization with Barra factors and developed backtest system to test the performance of machine-learning models.

In this report, we use two major types of models, tree model and neural network to predict the future returns of stocks, and compare in-/out-sample results of RankIC. In the process, we also explored the effect of hyperparameters on the results. With forecasting result, back-test against major indexes to test the predicted results in a real-trade fashion, considering transaction-cost, stock-suspension, limit-up/-down, restriction on turnover, etc.

2 Data Preprocess

2.1 Data Acquisition

a) **Stock pool:** All A-shares from January 2010 to October 2024. Excluding ST stocks, excluding stocks suspended from trading in the next trading day of each cross-section period, excluding stocks listed within 3 months, each stock is regarded as a sample.

b) **Backtest interval:** 2012-01-01 to 2024-10-01, semi-annual backtest.(If the benchmark index is CSI 1000, the interval will be converted to 2014-10-20 to 2024-10-01.)

c) **Feature and label processing:** On the last trading day of each natural week, the exposure of 123 factors was calculated as the original characteristics of the samples. Calculate individual stock returns for the next entire nature week as a sample label.

2.2 Feature Preprocessing

a) **Median de-extremum:** Set a factor in all the stocks on the exposure sequence in the phase T is D_i, D_M is the median of it, and $D_{M1}$is the median of the sequence $|D_i - D_M|$. Values that all greater than $D_M + 5D_{M1}$ will be reset to $D_M + 5D_{M1}$, Values that all smaller than $D_M - 5D_{M1}$will be reset to $D_M - 5D_{M1}$;

b) **Missing value processing:** After the new factor exposure sequence is obtained, the missing factor exposure is set as the average value of the same stocks in CITIC primary industry.

c) **Industry market value neutrality:** The factor exposure after filling in the missing value is linearly regression to the dummy variable of the industry and the market value after taking the logarithm, and the residual is taken as the new factor exposure.

d) **Normalization:** The neutralized factor exposure sequence is subtracted from its current mean and divided by its standard deviation to obtain a new sequence that approximates the N(0,1) distribution.

3 Model Fit

3.1 Boosting Model

Because each model is trained in almost the same way, we only introduce the training process here. Please refer to the appendix A for the detailed introduction of the model.

3.1.1 Training Mode

In our training process, we chose rolling training, making 6 months a period. The first training used two years as the training set (2010.01.01-2011.12.31), and the following 6 months (2012.01.01-2012.06.30) as the test set. After that, Each roll rolls back the end date of the training set and the start date of the test set by one phase, retuning and fitting the model.

During boosting model training, regularization is used in two places: one is the regular term in the model parameters, that is, some regular term is added to the loss function to avoid excessive complexity of the model; the other is the early stop mechanism during model training. We divide the above divided training set into the training set and the verification set, and use the verification set to realize the early stop of the model.

The early stop mechanism is not only used in the training of the model after confirming parameters, but also in the parameter adjustment process, so our data division is shown in the figure below1.

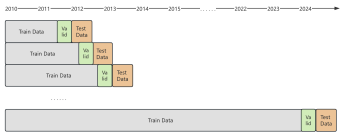


Figure 1: Data Split

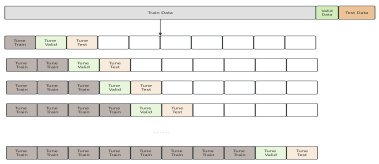


Figure 2: 10 Fold Cross Validation

3.1.2 Parameter Tunning

Since both XGBoost and LightGBM have a large number of parameters, these parameters are very important to the effect of the model, so we use 10-fold cross-validation to tune the parameters of the model every time we retrain the model.

To avoid prospective errors, our ten-fold cross-validation here uses the cross-validation of the time series model to ensure that the time of the test set is after the time of the training set, avoiding data leakage, in the format shown below2.

First, we select the range of parameters and estimated parameters, and then roughly fit ten data points within the initial selection range of a parameter3. After obtaining a more accurate range, we repeat this process until the parameter range meets our accuracy conditions, thus obtaining the final parameter range. If the change of parameters has a predictive effect on the model, That is, if the `Rank IC` does not change much, the default value of this parameter is used and is not optimized.

After that, we used optuna to adjust parameters, selected the optimal value of the average value of all tune_test sets in the ten-fold cross-validation as the optimal parameter of this training, and the trained model was used as the final model of this window. The following is an example of parameter adjustment4.

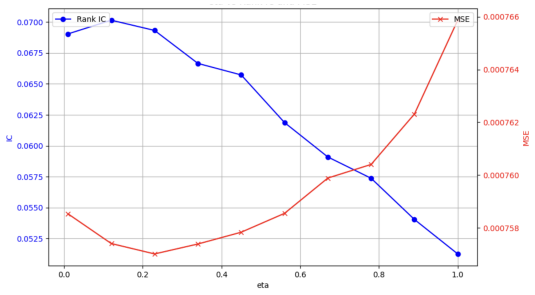
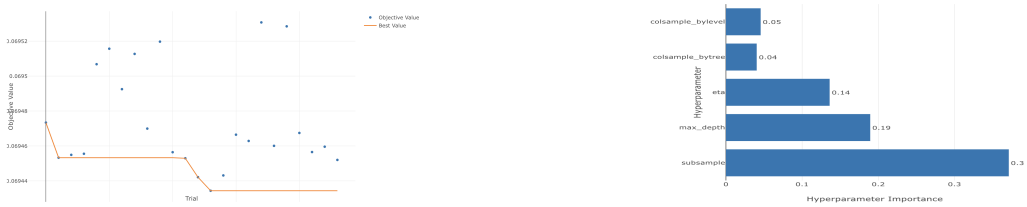


Figure 3: Parameter Pre-estimation



(a) Objective Value

(b) Hyperparameter Importance

Figure 4: Example of Tunning Parameters

Our predetermined parameters range is shown in the table below1.

Parameters	Meaning	XGBoost	LightGBM
learning_rate	Step size	(0,0.01)	(0.1, 0.001)
max_depth	Maximun depth of weak learners	(1,6)	(1,64)
colsample_bytree	The proportion of feature sampling when building the new tree	(0.2, 0.6)	-
colsample_bylevel	The proportion of column sampling when building the tree by level	(0,1)	-
min_data_in_leaf	Minimal number of data in one leaf	-	(2, 512)
feature_fraction	The proportion of feature used for random sampling	-	(0.1, 1)
bagging_fraction	The proportion of data used for random sampling	-	(0.1, 1)

Table 1: Predetermined Parameters

3.1.3 Model Training

After determining the parameter set the model is formally trained and evaluated, the loss function is set, the model training can be carried out.The details of the model are divided into the following aspects:

Loss Function: We first set the loss function of the model to `mean square error(MSE)`. The smaller the `MSE` is, the more accurate the model prediction is. But in our subsequent research on hyperparameters, we also custom `Rank IC` as a loss function.

Early Stopping: As mentioned earlier in Section 3.1.1, we used the early stop mechanism here to avoid overfitting. We set the early stop cycle to 10, that is, if the loss function of the model does not become smaller within 10 iterations, we stop the iteration immediately.

Other Hyperparameters: In both boosting models, we also have some additional hyperparameters to choose from, such as booster in the XGBoost model, which is directly specified as gbtree in this report. The following table lists the hyperparameters used by both models2.

3.1.4 Prediction Results

After the optimal parameters were determined according to the training set of the T period and before, the preprocessed features of all samples in the T+1 period were taken as the input to the model, and the predicted value $f(x)$of each sample was obtained. The predicted value was regarded as the synthesized factor and single-factor hierarchical backtest was carried out.

Hyperparameters	XGBoost	LightGBM
booster	gbtree	-
seed	777	777
device	cuda	cuda
boosting_type	-	gbdt
n_jobs	-1	-1
early_stopping_rounds	10	10

Table 2: Hyperparameters

The parameters of all models, `Rank IC` and `MSE` inside and outside the sample were obtained as shown in the table below34.

test_start_date	eta	max_depth	colsample_bytree	col_sample_bynode	colsample_bylevel	subsample	out_of_sample_ic	in_sample_ic	out_of_sample_mse	in_sample_mse
2012-01	0.009567	5	0.587397	0	0.137993	0.471500	0.062725	0.090485	0.051785	0.063708
2012-07	0.008658	6	0.210021	0	0.342508	0.583485	0.038171	0.079744	0.069239	0.062909
2013-01	0.009726	2	0.490892	0	0.331711	0.111434	0.049045	0.047404	0.064555	0.061085
2013-07	0.006722	6	0.509934	0	0.884252	0.740059	0.059887	0.080029	0.060506	0.063609
2014-01	0.005228	4	0.464638	0	0.982549	0.013151	0.030581	0.027797	0.055337	0.063204
2014-07	0.007189	4	0.249817	0	0.531568	0.029261	0.015942	0.029637	0.061312	0.062649
2015-01	0.002470	6	0.490967	0	0.409180	0.728060	0.052434	0.085187	0.111723	0.061571
2015-07	0.006165	6	0.268962	0	0.721492	0.565285	0.062980	0.085235	0.121797	0.062536
2016-01	0.008663	6	0.534990	0	0.667728	0.523848	0.075841	0.079953	0.075604	0.073506
2016-07	0.007877	6	0.354616	0	0.724870	0.319356	0.117681	0.074428	0.053696	0.075992
2017-01	0.004761	5	0.569319	0	0.608499	0.200941	0.053845	0.073630	0.056149	0.074486
2017-07	0.008392	5	0.293633	0	0.591662	0.703551	0.041212	0.056795	0.053221	0.073401
2018-01	0.009024	5	0.521386	0	0.352093	0.588351	0.041848	0.068487	0.065331	0.071820
2018-07	0.006273	6	0.419904	0	0.035646	0.794537	0.097172	0.069769	0.060671	0.071113
2019-01	0.004859	6	0.542960	0	0.365526	0.688595	0.089207	0.081853	0.068645	0.069802
2019-07	0.006437	5	0.211595	0	0.840661	0.504515	0.051134	0.067330	0.054495	0.070019
2020-01	0.006688	6	0.306023	0	0.791026	0.344117	0.031352	0.058720	0.075629	0.069556
2020-07	0.008504	6	0.204917	0	0.131183	0.638674	0.062658	0.061567	0.068826	0.069485
2021-01	0.009129	5	0.436262	0	0.607582	0.584909	0.064454	0.070860	0.066464	0.069386
2021-07	0.004189	5	0.487289	0	0.241793	0.567583	0.090428	0.058760	0.068497	0.069425
2022-01	0.004206	5	0.579005	0	0.897861	0.786699	0.064166	0.071246	0.070088	0.069245
2022-07	0.009442	5	0.261540	0	0.033429	0.491327	0.086890	0.061292	0.063844	0.069150
2023-01	0.007874	5	0.546968	0	0.687638	0.125000	0.095968	0.070345	0.054728	0.069040
2023-07	0.007356	6	0.402277	0	0.377237	0.196122	0.082989	0.072058	0.052631	0.068443
2024-01	0.007513	6	0.399397	0	0.346247	0.779819	0.066085	0.084684	0.084836	0.067530
2024-07	0.009937	6	0.593700	0	0.448443	0.571480	0.047748	0.080370	0.083237	0.068341

Table 3: Results of XGBoost Model

test_start_date	learning_rate	max_depth	num_leaves	min_data_in_leaf	feature_fraction	bagging_fraction	out_of_sample_ic	in_sample_ic	out_of_sample_mse	in_sample_mse
2012-01	0.002040	50	328	368	0.248309	0.289566	0.029208	0.130611	0.051815	0.063814
2012-07	0.001651	25	256	211	0.991667	0.753850	0.009330	0.094477	0.069334	0.061027
2013-01	0.092953	53	472	166	0.290846	0.412339	0.031829	0.096567	0.064628	0.058886
2013-07	0.035327	22	179	292	0.434067	0.167727	0.034065	0.090912	0.060599	0.062361
2014-01	0.096606	27	464	147	0.137927	0.186246	-0.031410	0.123339	0.055446	0.061893
2014-07	0.001668	10	181	249	0.422145	0.318917	-0.003798	0.185713	0.060954	0.063641
2015-01	0.019763	1	456	484	0.445341	0.844905	0.047789	0.046433	0.111243	0.059076
2015-07	0.007069	56	45	163	0.825273	0.785349	0.022956	0.071844	0.122922	0.059878
2016-01	0.006583	26	394	143	0.997179	0.944233	0.060792	0.152472	0.075937	0.088806
2016-07	0.010217	25	221	500	0.279567	0.677346	0.095046	0.129611	0.053830	0.097727
2017-01	0.002109	32	407	17	0.734046	0.830556	-0.006018	0.152261	0.056504	0.096862
2017-07	0.032275	36	274	137	0.588811	0.848152	0.035880	0.214674	0.052515	0.076822
2018-01	0.012184	55	362	80	0.183358	0.136420	0.081317	0.273623	0.064565	0.056400
2018-07	0.017743	46	444	67	0.187775	0.170811	0.050967	0.315025	0.060104	0.050722
2019-01	0.075363	14	240	411	0.959783	0.101506	0.043105	0.095048	0.069547	0.057374
2019-07	0.026123	6	504	186	0.118261	0.814740	0.025894	0.062562	0.054682	0.061286
2020-01	0.001231	24	159	69	0.721451	0.668056	0.011339	0.062487	0.075969	0.063493
2020-07	0.015726	39	115	438	0.570914	0.488620	0.028295	0.074143	0.068771	0.064580
2021-01	0.004024	3	138	276	0.904491	0.450828	0.020750	0.023851	0.066392	0.067088
2021-07	0.004325	23	369	158	0.557883	0.954273	0.005022	0.101936	0.068487	0.067126
2022-01	0.008875	12	205	456	0.926999	0.509430	-0.003362	0.087446	0.070147	0.070203
2022-07	0.012951	15	495	184	0.478078	0.541570	0.005126	0.161525	0.063852	0.067566
2023-01	0.001659	37	375	259	0.708171	0.225173	0.027007	0.140467	0.054769	0.067546
2023-07	0.038104	2	156	239	0.114916	0.201517	0.083745	0.102164	0.052683	0.064782
2024-01	0.003004	33	451	179	0.988586	0.983509	0.012296	0.177693	0.085180	0.060015
2024-07	0.004363	50	316	349	0.284244	0.192172	0.029650	0.215594	0.083995	0.063665

Table 4: Results of LightGBM Model

3.1.5 Model Exploration and Results

In order to compare whether different hyperparameters, such as loss function and training set length, will have an impact on the prediction results, we designed comparison tests respectively. Differences exist only in this hyperparameter, and other experimental conditions are the same.

The table below shows our single factor analysis of the predicted value5. According to the figure below, we have analyzed the selection of hyperparameters to some extent.

Loss Function

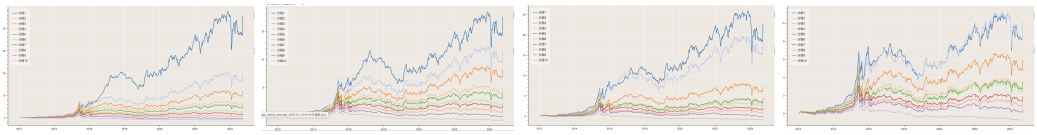
In the above model training process, `Mean Square Error(MSE)` is selected as the loss function of the model, but a small `MSE` does not completely mean the accuracy of the prediction. In the financial

	xgb_rolling_mse_P_1	xgb_rolling_mse_P_4	xgb_2y_mse_P_1	xgb_2y_mse_P_4	xgb_rolling_ic_P_1	xgb_rolling_ic_P_4	xgb_2y_ic_P_1	xgb_2y_ic_P_4
ir	5.914934 e-01	8.630556e-01	5.792810e-01	8.046646e-01	5.274276 e-01	7.535726e-01	4.345612e-01	6.340216e-01
kurtosis	2.554602 e-01	6.692715e-02	8.573073e-02	1.116335 e-01	8.453200e-01	3.449552e-01	4.461867e-01	2.623936e-01
mean	6.369107e-02	8.959708e-02	6.195201e-02	8.490513e-02	4.949839e-02	6.911793e-02	4.124564e-02	5.903189e-02
negative	1.840000 e+02	1.260000e+02	1.760000 e+02	1.280000 e+02	1.8800000+02	1.490000 e+02	2.110000 e+02	1.690000 e+02
p_value	9.839353 e-45	1.934430 e-81	3.297956 e-43	3.105514 e-73	6.492886e-37	3.732595e-66	1.635911e-26	3.705247 e-50
positive	4.730000 e+02	5.310000 e+02	4.810000 e+02	5.290000 e+02	4.690000 e+02	5.080000 e+02	4.460000 e 02	4.880000 e+02
sig_negative	1.598174 e-01	7.914764e-02	1.643836 e -01	1.263318 e-01	1.476408 e-01	1.065449 e -01	1.765601e-01	1.263318 e-01
sig_positive	5.875190e-01	6.560122e-01	5.768645 e-01	6.727549e-01	5.372907e-01	6.194825e-01	5.007610e-01	5.753425 e-01
significance	7.420814e-01	7.315234e-01	7.360483e-01	7.948718e-01	6.802413e-01	7.224736e-01	6.726998e-01	6.983409e-01
skew	-2.111259e-01	-1.2232320-02	-2.447546e-01	-2.311743e-01	-2.883674e-01	-9.631175e-02	-3.207132e-01	-2.504001e-01
std	1.076784e-01	1.038138e-01	1.069464 e-01	1.055162e-01	9.384869 e-02	9.172033e-02	9.491333 e-02	9.310706e-02
t_stat	1.516117 e+01	2.212185 e+01	1.484814 e+01	2.062517 e +01	1.351903 e+01	1.931558 e+01	1.113868 e+01	1.625125 e+01

Table 5: Rank IC of All XGBoost Models

field, `Rank IC` is used to evaluate the prediction ability of the model, which is often used to evaluate the effectiveness of quantitative investment strategies. Therefore, we changed the loss functions of parameter tuning and model training to `Rank IC` to compare the effects of different loss functions on the model.

We observe that the `Rank IC` of predicted value of the model with mse as the loss function is higher in terms of mean value than that with `Rank IC` as the loss function, but it is also relatively poor in terms of standard deviation, but the difference is small, that is to say, in terms of `Rank IC`, `MSE` is more stable as a loss function, and in the case of the data used in this report, mse performs better. Here we present a line chart of grouped cumulative returns for these model predictions5.



(a) Rolling_Mse_Week (b) Rolling_Mse_Month (c) Rolling_IC_Week (d) Rolling_IC_Month

Figure 5: Group Cumulative Return Analysis of Different Loss Function

Thoughts: We ask why the `Rank IC` of the two models are so different that it should logically be better to have `Rank IC` as a loss function than `MSE`.

We guess that one of the reasons is because both XGBoost and LightGBM need to back-propagate the gradient, but `Rank IC` as a discontinuous function, there is no gradient, so our loss function is not completely accurate `Rank IC`, The calculation of `Rank IC` should be to calculate the correlation once a day and finally obtain the average value, but our loss function is to obtain the correlation at one time, ignoring the influence of time, which may be one of the reasons for this problem.

Training Mode

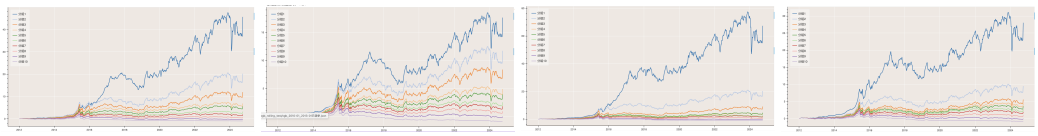
Our previous training mode was a fixed test set of 6 months, while the training set was lengthened constantly. In this process, although we gave the model more data, due to market changes, we could not guarantee the fixed distribution of data, so we also gave the model more noise. Here, we fixed the training set of 2 years10. `Rank IC` was used to compare the influence of different training modes on model prediction.

Through the results, we can observe that the prediction effect of the model with the expanding training set is higher in the mean value of `Rank IC`, and the standard deviation is also larger, which indicates that when we add more data, although more information and noise are added at the same time, according to the mean value of `Rank IC`, The positive effect caused by information is slightly higher than the negative effect caused by noise, so we believe that the training model with an ever-expanding training set is better6.

Model

In order to explore whether different models will have a greater impact on the predicted value, we only modified the model to LightGBM model, keeping other training modes and data sets unchanged, and then carried out a single factor analysis on the predicted value, `Rank IC` values are shown below6.

Thoughts: We found that not only the `Rank IC` mean and Group Cumulative Return of LightGBM during model training were much smaller than XGBoost, In the subsequent single-factor analysis of the predicted value, it is also found that `Rank IC` is much smaller than XGBoost. We guess that the



(a) Rolling_Mse_Week (b) Rolling_Mse_Month (c) 2y_Mse_Week (d) 2y_Mse_Month

Figure 6: Group Cumulative Return Analysis of Different Training Mode

	lgbm_2y_mse_P_1	lgbm_2y_mse_P_4	lgbm_2y_ic_P_1	lgbm_2y_ic_P_4
ir	$3.196681e-01$	$4.179731e-01$	$2.658829e-01$	$3.782844e-01$
kurtosis	$1.050315e+00$	$1.214855e+00$	$1.569300e+00$	$1.771824e+00$
mean	$2.906342e-02$	$3.973959e-02$	$2.057019e-02$	$2.963026e-02$
negative	$2.400000e+02$	$2.010000e+02$	$2.300000e+02$	$2.030000e+02$
p_value	$1.328228e-15$	$8.602825e-25$	$3.247763e-11$	$1.634931e-20$
positive	$4.170000e+02$	$4.560000e+02$	$4.150000e+02$	$4.420000e+02$
sig_negative	$1.902588e-01$	$1.552511e-01$	$1.751938e-01$	$1.426357e-01$
sig_positive	$4.185693e-01$	$4.901065e-01$	$3.658915e-01$	$4.403101e-01$
significance	$6.078431e-01$	$6.425339e-01$	$5.309201e-01$	$5.701357e-01$
skew	$-5.919524e-02$	$-3.028373e-01$	$-3.180850e-01$	$-5.852924e-01$
std	$9.091749e-02$	$9.507691e-02$	$7.736562e-02$	$7.832799e-02$
t_stat	$8.193736e+00$	$1.071349e+01$	$6.752587e+00$	$9.607232e+00$

Table 6: Rank IC of LightGBM

reason for this problem is not that LightGBM's model is worse than XGBoost, but that LightGBM requires longer training time. Although we have set the tuning time to 4 hours, it may not be enough to adjust the model to the optimal parameters.

Here, again, we show a line chart of grouped cumulative returns for these model predictions7.

(a) XGBoost_2y_Week (b) XGBoost_2y_Month (c) LightGBM_2y_Week (d) LightGBM_2y_Month

Figure 7: Group Cumulative Return Analysis of Different Models

3.1.6 Optimization Result

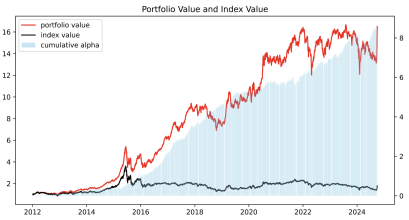
After model prediction, we used the optimization method mentioned in Section 4 to optimize the backtest process. We took the training model with the fixed length training set and the model with `MSE` as the loss function as an example to show the results, as shown in the figure below8(The benchmark is 000905.SH, barra_limit = 0.3).

We found that the XGBoost model also performed better than the LightGBM model after optimization, which matches our `Rank IC`. .

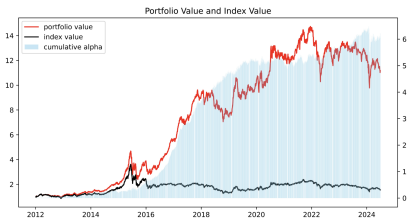
3.1.7 Which is more important, out-of-sample IC or IC difference between out-sample and in-sample?

According to the experience of other experts, the best difference between inside and outside the sample is not large, but this report finds that not every model in the sample of the `Rank IC` values meet this condition, We want to explore whether it is the difference between out of sample `Rank IC` or out of sample `Rank IC`. So we set up the following experiment.

We trained two XGBoost models in exactly the same flow, except that they differ in the parameters that need to be adjusted against cross validation each time. (Since many of our arguments are of floating-point type, the optimal arguments we get for each call are different.) Then, according to the `Rank IC` and `Difference` of these two sets of models, two sets of predicted values are constructed. One group is all the predicted values of the larger model outside the sample. The first group is all the



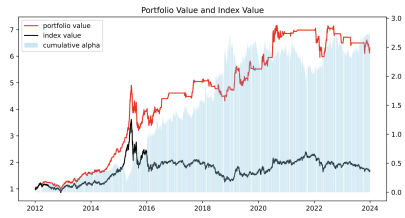
(a) Backtest Results of XGBoost



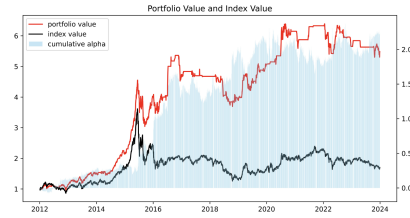
(b) Backtest Results of LightGBM

Figure 8: Backtest Results of Two Models

predicted values of the model with small difference between inside and outside the sample, and the backtest method in Section4 is used to backtest and compare, the results are as follows:



(a) Backtest Results of Large Rank IC



(b) Backtest Results of Small Difference

Figure 9: Backtest Results of Two Prediction Values

We find that the backtest effect with a small difference is worse than `Rank IC`, which conforms to the logic that `high rank IC` means the predicted value is accurate. Then we come to the conclusion that in this experiment, we think that `Rank IC` is more important, and the difference between inside and outside the sample is not as important as `Rank IC`.

3.2 Deep Learning

3.2.1 Training, Validation, and Prediction Dataset Partitioning

In this paper, the backtesting interval is set to 2010-2024, and this part is the same as the boosting model, which adopts the rolling training method, setting six months as a cycle. However, in deep learning due to the limitations of arithmetic using a fixed rolling training window, the training window period is set to two years, and the predicted window period is set to the training window period of the next six months. For example: the first training window is (2010.01.01-2011.12.31), the first prediction window is (2012.01.01-2012.06.30); the second training window is (2010.07.01-2012.06.30), the second prediction window is (2012.07.01-2012.12.31). and so on.

The training set is further divided into a training set for initial training and a validation set for model evaluation, where the optimal model is obtained by iterating the model, minimizing the loss equation and setting the early stopping condition. It is always divided into five parts, the first four being the training set and the last part being the validation set, and this division can also avoid data leakage.The following figure shows the training, validation and prediction windows.10

3.2.2 Transformation of Data Features and Labels

Before model training, the data needs to be transformed into features and labels in the form of torch.tensor so that it can be received by the data loading class TensorDataset. Since the three datasets and each stock are independent of each other, grouped by stock and traversed by date, the 123 feature data from the last five weeks are used as model features, which contain samples with a time step of 5, and the labeled data are the return labels for the next point in time entered into the model. The table below 7 shows the shape of the data for each of the data after one data transformation has been performed, which needs to be consistent with the probe question and the model inputor to ensure accuracy.

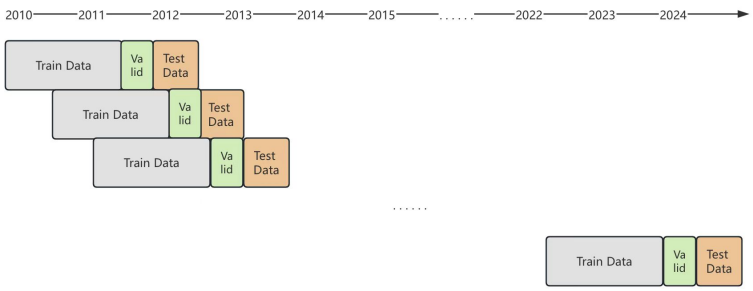


Figure 10: Training, Validation and Prediction Windows Split

Datasets	Shape	Interpretations
X_train	torch.Size([152294, 5, 123])	[Weekly Samples train, Time-steps, Features]
y_train	torch.Size([152294, 1])	[Weekly Samples train, Dimension of label]
X_val	torch.Size([36250, 5, 123])	[Weekly Samples val, Time-steps, Features]
y_val	torch.Size([36250, 1])	[Weekly Samples val, Dimension of label]
X_insample	torch.Size([188544, 5, 123])	[Weekly Samples train and val, Time-steps, Features]
y_insample	torch.Size([188544, 1])	[Weekly Samples train and val, Dimension of label]
X_test	torch.Size([49396, 5, 123])	[Weekly Samples test, Time-steps, Features]
y_test	torch.Size([49396, 1])	[Weekly Samples test, Dimension of label]

Table 7: Example of Transformed Data Shapes

3.2.3 Parameter Tunning

Because the parameters of the three deep learning models, RNN, LSTM and GRU, are more consistent, this paper will set up a unified parameter tuning board for the three models.

The model parameters are divided into general parameters which are set according to the data and literature, and hyper-parametric tuning parameters: these parameters usually have their possible values and ranges according to the literature and research papers, and the refinement of choosing the optimal values of these parameters in different training cycles is very important for the predictive performance of the model. The following table describes the general and special parameters8.

Parameters	Meaning	General (fixed-value) & Special Parameters (range)
input_size	Input dimensions of the model	123features
output_size	Output dimensions of the model	1 (label)
num_layers	Number of hidden layers in the model	2
time_steps	Number of historical input sequences used to predict the output	5
early_stop	Number of early stops	10
hidden_size	Number of neurons in hidden layer	[32, 64, 96, 128, 160, 192, 224, 256, 288, 320]
learning_rate	Step size	[0.0001, 0.001, 0.005, 0.01, 0.03, 0.05]
batch_size	Number of samples in a single training batch	[256, 512, 1024, 2048, 4096, 8192, 16384]
dropout_rate	Random disconnection of input neuron ratios (to avoid over-fitting)	range(0.1 - 0.3), step=0.05

Table 8: General and Special Parameters

In this paper, we re-tune the grid search for these particular parameters every training cycle to better select the parameters for the training data and then further select the optimal model.

The initially selected range of hyperparameters is instantiated using the $create\_study$ function in the optuna library to set the direction to minimize the loss function, and the optimize function performs five iterations of the set objective equation to determine the optimal parameter set. After obtaining a more accurate range, we repeat this process until the parameter range meets our accuracy conditions, thus obtaining the final parameter11.

The five iterations use a chronological cross-validation approach similar to the tree model, distinguishing between the training and validation sets to continuously obtain the minimum loss for parameter

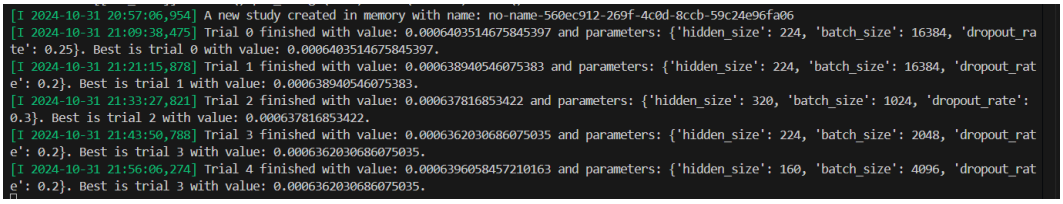


Figure 11: Example of Tunning Parameters (five iterations)

selection, where the model training function and evaluation function are consistent with the formal training, as explained in detail in the next section.

3.2.4 Model Training and Evaluation

After determining the parameter set the model is formally trained and evaluated, the loss function is set, and switching between model training and evaluating the model in Pytorch is performed. The main can be divided into the following parts:

Training function: by constantly receiving training data sent to the model for forward propagation, calculate the loss and then back-propagation to calculate the gradient, use the optimizer to update the weights of each layer of the model to complete a training step. Ensure that the model can perform well on the training set.

Evaluation function: use the validation data to calculate the value of the loss function to directly evaluate the model generalization ability, constantly update the gradient, and select the model with the smallest average loss of the entire validation set. It can help monitor the performance of the model on unknown data and avoid over-fitting.

Early stopping: In order to prevent over-fitting of deep learning models, this paper uses the regularization technique of early stop, and setting the early stop parameter to 10 means that when the model does not improve its performance in ten consecutive rounds of iterations on the validation set, it stops and saves the model in advance.

Special restrictions: to avoid gradient explosion, $torch.nn.utils.clip\_grad\_value\_()$ is used to limit the maximum value of the parameter gradient to 3.0, and the computation of the gradient is disabled by $torch.no\_grad()$ to save memory and speed up computation.

3.2.5 Model Exploration and Results

Order Stock and Random Stock

Considering the correlation between stocks it is considered that the data features and labels extraction part should not be input into the model in a data specific order every time, which may have a potential impact on the training results of the model. So in this paper, we disrupt the stock codes to take the numbers before extraction and then transform the data, expecting to explore whether the uncertainty of the correlation between stocks really has an impact on the model results9.

pred_start_date	in_sample_ic(Order)	out_sample_ic(Order)	in_sample_ic(Random)	out_sample_ic(Random)	pred_start_date	in_sample_ic(Order)	out_sample_ic(Order)	in_sample_ic(Random)	out_sample_ic(Random)
2012/1/1	0.050365778	0.041446011	0.057156344	0.045056329	2018/7/1	0.023084537	0.0416231	0.0255809	0.047673015
2012/7/1	0.047201729	0.049104103	0.049676417	0.042124868	2019/1/1	0.044473832	0.065939081	0.04275657	0.071901693
2013/1/1	0.03260107	0.02800434	0.02702266	0.030013649	2019/7/1	0.062972652	0.04537209	0.06449678	0.054569612
2013/7/1	0.022544874	0.031044274	0.030775979	0.036371979	2020/1/1	0.024798819	0.046312331	0.027805726	0.047015351
2014/1/1	0.066338285	0.05772721	0.049341377	0.054577513	2020/7/1	0.037434058	0.015606584	0.030502696	0.012263384
2014/7/1	0.036887356	0.004357893	0.056137481	0.015518826	2021/1/1	0.029688042	0.048320736	0.024013354	0.043964754
2015/1/1	0.026849501	0.060128257	0.038299459	0.041078537	2021/7/1	-0.007775665	0.034230907	-0.001720432	0.026094488
2015/7/1	0.084783621	0.021036341	0.080852274	0.018075501	2022/1/1	0.046132914	0.045475133	0.05135133	0.052806583
2016/1/1	0.044547964	0.044992447	0.035498837	0.05867099	2022/7/1	0.032901537	0.096960834	0.032568033	0.100651312
2016/7/1	0.053041516	0.041062094	0.049293334	0.043407513	2023/1/1	0.048223128	0.05970735	0.049541015	0.051641828
2017/1/1	0.041492547	0.017459154	0.052786135	0.028473122	2023/7/1	0.013989977	0.038466254	0.009254337	0.025368566
2017/7/1	0.034761012	0.037584812	0.035630558	0.045376172	2024/1/1	0.048258578	0.034633702	0.027626855	0.04761542
2018/1/1	0.07015962	0.020208608	0.071806553	0.01088888	2024/7/1	0.061167661	0.053950963	0.044885492	0.060262695

Table 9: Order and Random Stock Rank_IC Result (RNN Model)

The sampling test of the two methods on the model found that the results are basically the same12, then it can be proved that the two grouping methods do not differ much on the model, and the correlation degree of uncertainty between stocks does not significantly affect the model results. And because the data is weekly and the training window is two years, combined with the set $batch\_size$,

(a) In_sample IC Results

(b) Out_sample IC Results

Figure 12: IC Results of Order and Random Stock (RNN Model)

each training will not be too much influence of certain stocks.In the final modeling framework, the data are still transformed in the form of stock order.

Loss Function

For deep learning models, the most basic loss function is the `MSE`, but obtaining a very small `MSE` result does not fully illustrate the model's accuracy in prediction, and the `Rank IC` metric itself evaluates model performance both in and out of sample. In this paper, we set `MSE` and negative `Rank IC` on the loss function, and use different loss functions in both parameter tuning and formal running of the model. The difference between the in_sample_IC and out_sample_IC values obtained under the RNN model trained using the negative `Rank IC` as a loss function13, which we will find to be rather obviously irrational: the out_sample performs too well under most cycles and is significantly higher than the in_sample values. Whereas the results obtained by utilizing `MSE` as the loss function are more in line, this loss function, which is more suitable for time series data, is still chosen in this paper.

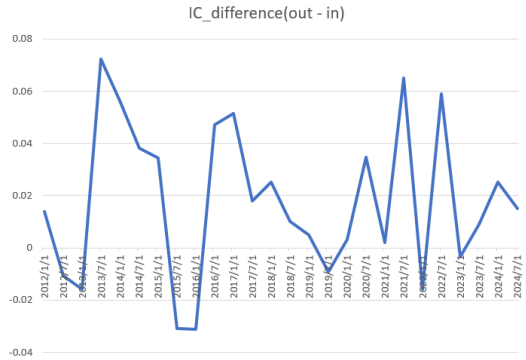
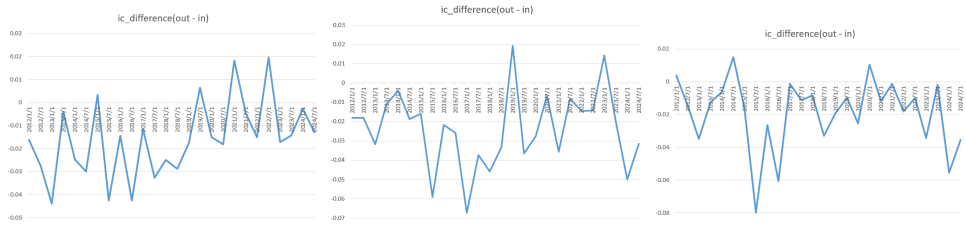


Figure 13: The difference between the in_sample and out_sample values

Model

After determining the `MSE` as the loss function, our overall deep learning framework has been built, and the written RNN, LSTM and GRU models enter the framework to run, obtain the results of the model parameter sets and `Rank IC` values inside and outside the samples for each cycle, giving the optimization part to provide the prediction results. The following figure 14 shows the differences in `Rank IC` values within and outside the sample for the three models, with specific model parameters and values shown in the Appendix.B

As can be seen from the figure14, the in-sample IC is a bit larger than the out-of-sample IC, but the difference is less in most of the time, which can reflect the model's better in-sample and out-of-sample prediction ability. In addition, through the numerical statistics in the appendixB, it is found that the in-sample IC values of RNN, LSTM and GRU are basically between 0.8-0.9, and the out-of-sample IC values are basically between 0.5-0.7, which is a more reasonable state.

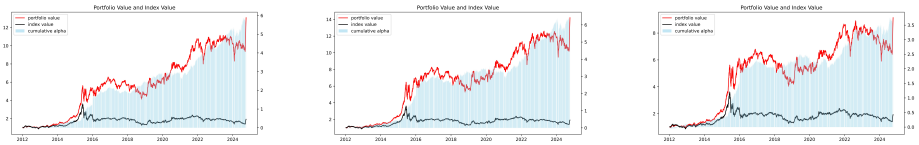


(a) RNN Model (b) LSTM Model (c) GRU Model

Figure 14: Differences in IC-value Analysis of Different Models

3.2.6 Optimization Result

After model prediction, we used the optimization method mentioned in Section4 to optimize the back-test process. We took the training model with these three models to show the results, as shown in the figure below.15(The example benchmark is 000905.SH, barra_limit = 0.5)



(a) RNN Model (b) LSTM Model (c) GRU Model

Figure 15: Backtest Results of Three Models

4 Optimization

4.1 Portfolio Optimization

The prediction model provides basic insights into which stocks are expected to perform better in the next week. However, for hedge funds and large investors, simply selecting the stock with the highest predicted return is impractical for several reasons:

Risk-Adjusted Returns: Investors, particularly hedge funds, prioritize higher Sharpe ratios over merely higher returns. This preference arises because they seek to maximize the trade-off between return and volatility, avoiding significant drawdowns typically associated with high volatility. Therefore, portfolios that balance return and risk are generally more desirable.

Portfolio Specifics: Strategies like Smart Beta aim to maintain industry and risk factor exposures similar to a benchmark index. This approach seeks to generate abnormal returns that are independent of market trends by controlling for specific factors.

Diversification: Excessive exposure to a single industry or risk factor (such as those identified by Barra) is considered risky. Investors prefer diversified portfolios to mitigate this risk, even if the prediction model suggests certain industries may outperform others in the future.

In this study, we utilize a 5-day (weekly) trading period as the adjustment interval to perform portfolio optimization using Barra factors and backtesting techniques.

4.2 Barra Factors

The Barra Risk Factor Analysis is a multi-factor model developed by Barra Inc. It is used to measure the overall risk associated with a security relative to the market. The model divides risk into two main categories: **systematic risk** which is driven by exposure to common factors such as market movements, size, value, and sector, and **idiosyncratic risk** which is pecific to individual assets and unrelated to broader market factors. In this paper, we consider the following Barra style factors:

Beta, Momentum, Size, EarningsYield, ResidualVolatility, Growth, BooktoPrice, Leverage, Liquidity, NonlinearSize

For instance, Momentum captures the tendency of high-performing stocks to continue performing well in the near term, while EarningsYield reflects the potential of undervalued stocks to generate returns. These factors explain the systematic risk of a stock and the overall portfolio. By restricting the exposure to these factors to levels similar to those of the market index, the portfolio becomes market-neutral, meaning that its performance is isolated from broader market trends.

4.3 Optimization and Backtest

A typical portfolio optimization problem consists of three parts, decision variable, objective function and constraints. In this paper, out optimization problem can be represented by:

$$
\begin{aligned}
Max_w \quad & w^T R^p && (1)\\
s.t. \quad & |B_f^p - B_f^{index}| <= barra\_limit && (2)\\
& |I_f^p - I_f^{index}| <= 0.5 && (3)\\
& w_i <= \frac{1}{num\_stock} && (4)\\
& \sum_{i=1}(w_i - w_{i,0}) <= turnover\_limit && (5)\\
& w \geq 0 && (6)\\
& \sum_{i=1}^{N} w_i = 1 && (7)
\end{aligned}
$$

The optimization problem focuses on determining the weights of all stocks in the A-share market, under the following constraints:

4.3.1 Decision Variable

The decision variable w is as n stock's weighted matrix $[x_1, x_2 \ldots . x_n]$ for adjusting the position at the adjustment date(weekly). The weight matrix before adjustment date is w_0.

We set every 5 trading days as adjustment day. The optimized w is calculated at the day before the adjustment day after close price is know, which also means w_0 is known.

4.3.2 Objective Function

Objective Function(1) defines the goal is to maximize the sum of weighted prediction return for the selected stocks.

4.3.3 Constraints

Barra Factor Exposure(2) constraints the difference in Barra factor exposure between the portfolio B_f^p and the index B_f^{index} should not exceed a predefined barra_limit. The portfolio Barra exposure is calculated by sum of weighted Barra exposure of each stock at the current date.

Industry Exposure(3) constraints the difference in industry exposure between I_f^p and I_f^{index}, which should not exceed 0.5.

Maximum Weight per Stock(4) requires that no individual stock's weight should exceed $\frac{1}{num\_stock}$. Our results show that the selected stocks often approach this limit. For instance, if the weight limit is set to 1%, the optimized portfolio will typically include around 100 stocks with weights near 1%, while the remaining stocks have weights close to zero.

Turnover(5) should be below the specified turnover_limit. Turnover is defined as the sum of difference between previous weight and optimized weight.

Weight Constraint(6) requires that the sum of all stock weights must equal 1.

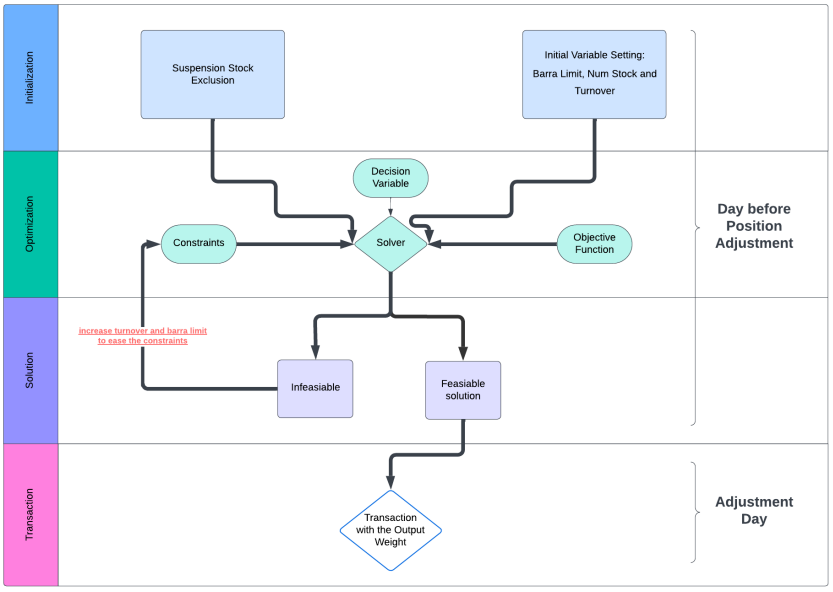


Figure 16: Optimization and Backtest Flow

4.3.4 Backtest

The backtest system is illustrated in the flow chart in figure 16.

Every 5 trading days, the portfolio's stock positions are adjusted based on the weights calculated at the end of the day prior to the adjustment day.

First, using the daily VWAP (Volume-Weighted Average Price), we exclude any stocks that are suspended on the adjustment day. In other words, the suspension status of a stock is determined the day before the adjustment day. This approach may introduce some bias by incorporating future information.

Next, with the current stock position w_0 and initial variable settings such as barra_limit, num_stock, and turnover_limit, the constraints of the optimization problem are defined. To adjust the current stock positions to the optimized weights w, stocks are bought and sold at the VWAP price plus 0.005% of total value traded as transaction fee on the adjustment date.

After the adjustments, no transactions are made for the next 5 trading days. However, if a stock is suspended during this non-trading period, the current stock position in that stock will be forcibly sold at the last available VWAP price.

5 Results

5.1 Model Performance Summary

The overall backtest results for the XGBoost and GRU models are presented in Appendix C. The table presents the annulized alpha return, maximum drawdown and Sharpe ratio for XGBoost and GRU under various variable settings. The backtesting was conducted over the period from 2015-01-01 to 2024-10-10, with 000852.SH (CSI 1000) used as the benchmark index for risk control with turnover_limit $= 0.15$. The XGBoost model exhibits better performance to the GRU model in terms of returns, and Sharpe ratio. For most barra_limit and num_stock settings, the Sharpe ratios of XGboost model are larger than 1, with the GRU model lower than 0.5 in most cases. Additionally, we observed a slight decline in the Sharpe ratio as num_stock increases, indicating that both models are effective in stock selection.

We also include the performance of an equally-weighted portfolio as a comparison, which corresponds to the results for barra_limit $= 1$ presented in Appendix C table 14. The equivalence between an

equally-weighted portfolio and barra_limit $= 1$ is explained in Constraint (4). The equally-weighted portfolio's higher annualized return comes with high double the maximum drawdown of optimized portfolio, leading to only half of Sharpe ratio compared with the optimized portfolio. Our findings reveal that optimization using the Barra factor significantly improves the Sharpe ratio by substantially reducing the maximum drawdown, while maintaining a relatively similar return compared to the equally-weighted portfolio.

5.2 Implication on Barra Constraints

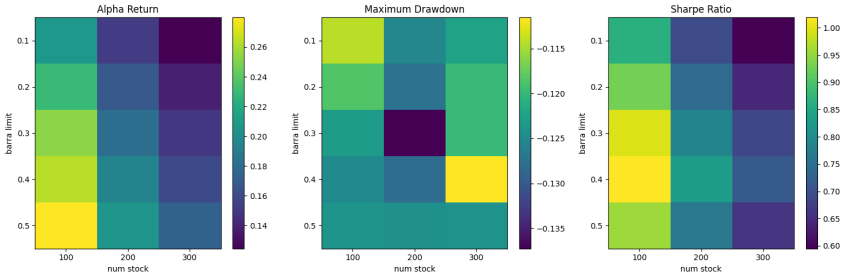


Figure 17: Heatmaps for XGBoost Alpha Return, Drawdown and Sharpe

The heatmaps in figure 17 illustrate the annualized alpha return, maximum drawdown, and Sharpe ratio of XGBoost model over the same period, with CSI 1000 as the index. The first heatmap (on the left) displays the annualized alpha returns across different settings of barra_limit and num_stock. A higher barra_limit indicates that the portfolio's Barra factor exposures do not need to align closely with those of the index, allowing for greater flexibility. This increased flexibility enables the optimization process to select more stocks with high predicted returns, rather than being constrained to maintain the portfolio's Barra exposures close to the index. As a result, portfolios with higher barra_limit values tend to achieve higher alpha returns.

However, higher barra_limit values also result in increased maximum drawdowns, as more flexible portfolios are generally associated with higher volatility. The final heatmap (on the right) reveals that the highest Sharpe ratios are not observed in settings with the largest alpha returns. Instead, the settings that achieve the highest Sharpe ratios are typically around barra_limit $= 0.4$.

Moreover, as num_stock increases, the portfolio becomes more diversified by including a larger number of stocks. This diversification helps to spread risk, which explains why both alpha returns and maximum drawdowns tend to decrease as num_stock increases.

These findings align with Markowitz's efficient frontier theory, which suggests that higher returns are generally accompanied by higher risk. In practice, investors should focus on constructing portfolios positioned along the "efficient frontier," rather than simply pursuing higher returns. Our results also highlight that the Barra factor is an effective tool for assessing the risk of individual stocks and portfolios. By carefully balancing risk and return, portfolio optimization can deliver superior performance while minimizing exposure to unnecessary volatility.

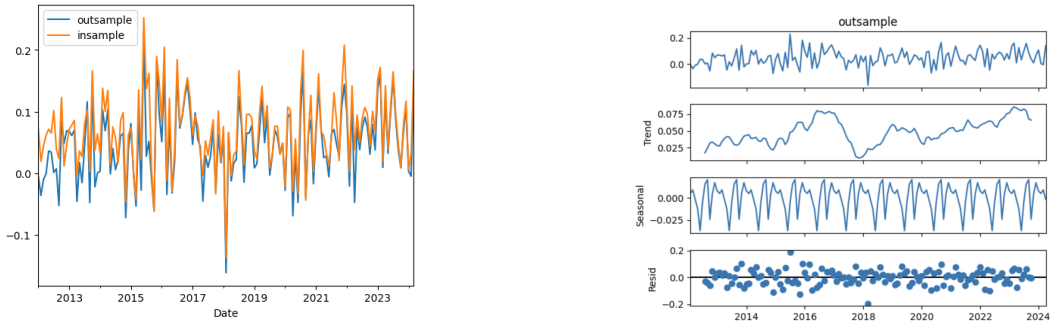
6 Future Work

Relatively high yields in exceptional years: Whether it is a tree model or a deep learning model, our newspaper reports that the annualized rate of return after optimization is relatively high in 2015, 2016, and 201710. In this report, we do not know whether this is logical or not, and we have not found the reason. We will continue to think about this in the future.

Seasonal effect: In the tree model, we calculated the monthly `Rank IC`, drew their line charts, and carried out the seasonal decomposition. We observed that our predicted value does have a certain seasonal cycle effect, so we can consider adding the time factor into the current 123 features in our subsequent work.

trade_year	return_XGBoost	sharpe_XGBoost	max_drawdown_XGBoost	return_LSTM	sharpe_LSTM	max_drawdown_LSTM
2012	0.11918	0.740640	-0.038730	0.096400	0.923909	-0.053919
2013	0.15455	0.966687	-0.050489	0.124040	0.716576	-0.083152
2014	0.06264	0.475344	-0.055644	0.177280	1.071503	-0.053396
2015	0.45836	1.275056	-0.093817	1.217030	2.124645	-0.108051
2016	0.60513	2.164034	-0.060351	0.415250	1.409707	-0.078922
2017	0.15854	1.277698	-0.053369	-0.107270	-0.776849	-0.114260
2018	0.12050	0.738311	-0.065685	0.180090	1.009295	-0.152649
2019	0.18121	1.106733	-0.039180	0.051490	0.369820	-0.059729
2020	0.04557	0.326112	-0.056896	-0.050300	-0.108976	-0.117000
2021	0.06083	0.539288	-0.061586	0.166171	1.119931	-0.160126
2022	0.13602	0.830638	-0.058751	0.328690	1.603337	-0.064629
2023	0.16346	1.500393	-0.027018	0.154750	1.297281	-0.053321
2024	-0.00556	0.073016	-0.124035	0.039560	0.286484	-0.156275

Table 10: Annualized Return of XGBoost and LSTM



(a) Monthly Rank IC of XGBoost

(b) Seasonal Decomposition

Figure 18: Seasonal Effect

Problems in special years: In the tree model, we find that Rank IC of the predicted value is relatively low in some years, and Rank IC of some years is relatively high. In the subsequent work, we will continue to optimize the parameters and control the parameters to prevent over-fitting.

In addition, in 2016, the predicted value outside the sample is much higher than the predicted value inside the sample. We do not know the cause of this problem, and we will carefully study this problem in the follow-up work.

Negative IC loss function in deep learning model: It is possible that the negative IC loss function is not applicable to the current model for the following two reasons:

a) Limitations of the information coefficient: The IC loss function itself measures the linear correlation between the model predictions and the true values. Its core assumption is that the model predictions can have some linear relationship with the true values, but for deep learning models, it is usually more capable of capturing more complex non-linear relationships.

b) Data distribution bias: there may be distributional differences between the training data and the test data (out-of-sample data), especially in the fields of finance and time series, where the patterns of historical data may change over time. Negative IC as a loss function may not adequately account for these distributional changes, and thus the model will outperform the in-sample data on the out-of-sample data. Out-of-sample data may not be inherently as complex or noisy as in the training data, thus allowing the model to perform better on these data.

The research in this paper has identified possible problems with the negative IC loss function, and in future research it will be possible to think more deeply about how to improve and thus introduce the use of the negative IC loss function in different ways, potentially achieving better results.

Quadratic objective function: In this paper, we use simple linear objective function for optimization. For further research, risk term such as covariance matrix of past return and covariance matrix of past Barra exposure could be added to replicate the risk of portfolios.

Appendix

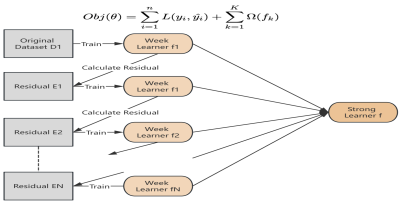
A Model Introduction

A.1 Boosting Ensemble Learning Model

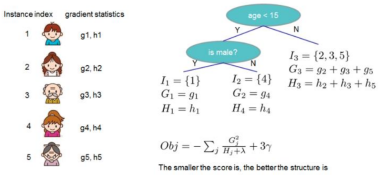
A.1.1 XGBoost

XGBoost is an efficient implementation of Gradient Boosting method and also an improvement and enhancement of GBDT algorithm. Compared with the traditional GBDT algorithm, XGBoost has been improved in the aspects of loss function, regularization, split-point finding and parallelization design, making it more than 5 times faster than the common toolkit.

XGBoost makes the final objective function only depend on the first and second derivative of the loss function at each data point by doing second-order Taylor expansion of the objective function, so that it is easy to achieve parallelism. Here we shows the whole process and the single iteration learning process of XGBoost19.



(a) Learning Process of XGBoost



(b) Single Iteration Learning Process of XGBoost

Figure 19: XGBoost

This report mainly introduces XGBoost algorithm based on CART tree learner. Give a sample point and calculate the score of the sample on each CART, which adds up to the final score of the sample, from which it can be predicted or classified. The mathematical description and objective function of the XGBoost model are as follows:

$$\hat{y}_i = \sum_{k=1}^{K} f_(k)(x_i), Obj(\theta) = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k).$$

Where n is the number of samples, x_i represents the $i-th$ sample, y_i and $\hat{y}_i$ is the true and predicted value of the ith sample; K is the number of carts, f_k represents the kth CART, which can be regarded as the mapping from sample point to score; $L(y, \hat{y})$ is the loss function, $\Omega(f_k)$ is the regularization term.

When training the $k-th$ tree, it is equivalent to minimizing the objective function of the $k-th$ tree:

$$Obj^{(t)} = \sum_{i=1}^{n} L(y_i, \hat{y_i}^{(t)}) + \sum_{k=1}^{t} \Omega(f_k) = \sum_{i=1}^{n} L(y_i, \hat{y_i}^{(t-1)} + f_t(x_i) + \Omega(f_t)) + const$$

The regular part of the objective function (equivalent to the complexity of the tree) is defined as follows:

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} \omega_j^2$$

Where f_t represents the $t-th$ tree, T represents the number of leaf nodes in the tree, and ω_j represents the fraction on the $j-th$ leaf node. γ and λ is the penalty factor, and the larger the penalty is for the complexity of the tree.

The next step is to refine the f_t by splitting the tree into the structure part q and the weight part ω:

$$f_t(x) = \omega_{q(x)}, \omega \in R^T, q : R^d \hookrightarrow 1, 2, ..., T$$

Where ω is a T-dimensional vector corresponding to the fraction on T leaf nodes; q is a mapping that maps the sample point $x \in R^d$ to one of the leaf nodes. Therefore, as long as we determine the structure of the tree q and the score on each leaf node ω, we can get this tree.

A.1.2 LightGBM

Another advanced implementation of GBDT is LightGBM, which uses a leaf-wise growth strategy to select the leaf with the maximum delta loss to grow, leading to deeper trees and potentially better fit. The algorithm also incorporates a technique called Gradient-based one-sided sampling (GOSS), which retains instances with large gradients while randomly sampling those with smaller gradients, ensuring that the model focuses on harder-to-predict cases.

Compared with XGBoost20, LighTGBM has the advantages of fast training speed, low memory consumption, efficient parallel training support, better accuracy, distributed support and fast processing of massive data.

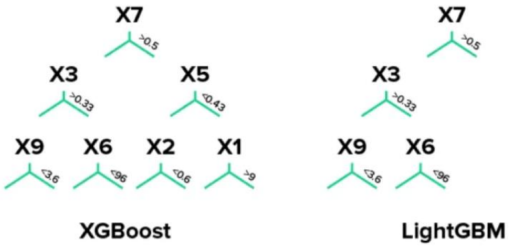


Figure 20: Comparison of XGBoost and LightGBM

A.2 Deep Learning Model

Neural network is the core technology of artificial intelligence which has developed rapidly in recent years, and recurrent neural network has strong time series prediction ability, suitable for stock forecasting. In this paper, we select three kinds of recurrent neural network models, namely, RNN, LSTM and GRU, for the study of training and prediction direction. The following section describes the basic principles and differences between these three models, their main parameters are basically the same, this paper sets the data processing, parameter tuning, training and prediction framework consistent under the comparison of the degree of prediction of each model on the stock. And the meaning of each parameter and the reason for selection will be explained in detail in the parameter tuning section of the model in the article.

A.2.1 Recurrent Neural Network

RNN as a recurrent neural network adds the processing of time-series data to the traditional neural network model, which can maintain the persistence of the information by using the long-term factor data of the stock as the time series, and using the data in the past period of time as the input value.

The unfolded representation of the RNN21 shows that U, V, and W are the three matrix parameters of the model, and their sharing in the model can reflect the idea of circular feedback in the RNN model.

The forward propagation algorithm for RNNs focuses on obtaining hidden states and using activation functions to obtain predictions:

$$h_t = \sigma(z_t) = \sigma(UX_t + Wh_{t-1} + b), o_t = Vh_t + c, \hat{y}_t = \sigma(o_t)$$

Backpropagation uses gradient descent to find the optimal solution for the three parameters U, V, and W when letting the loss function be minimized:

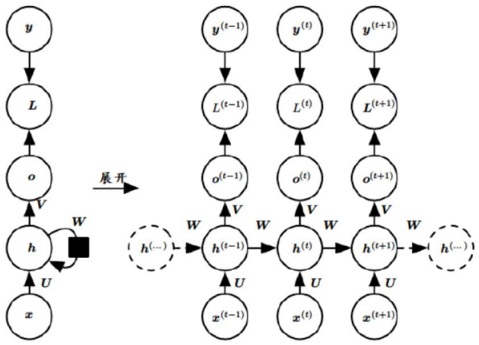


Figure 21: RNN Unfolded Representation

$$\frac{\partial L}{\partial c}=\sum_{t=1}^{\tau}\frac{\partial L_t}{\partial c}=\sum_{t=1}^{\tau}\frac{\partial L_t}{\partial \sigma_t}\times\frac{\partial \sigma_t}{\partial c}=\sum_{t=1}^{\tau}\hat{y}_t-y_t,\frac{\partial L}{\partial V}=\sum_{t=1}^{\tau}\frac{\partial L_t}{\partial V}=\sum_{t=1}^{\tau}\frac{\partial L_t}{\partial \sigma_t}\times\frac{\partial \sigma_t}{\partial V}=\sum_{t=1}^{\tau}\left(\hat{y}_t-y_t\right)\left(h_t\right)^T$$

$$\delta_t=\frac{\partial L}{\partial h_t}=\frac{\partial L}{\partial \sigma_t}\times\frac{\partial \sigma_t}{\partial h_t}+\frac{\partial L}{\partial h_{t+1}}\times\frac{\partial h_{t+1}}{\partial h_t}=V^T\left(\left(\hat{y}_t-y_t\right)+W^T\delta_{t+1}\operatorname{diag}\left(1-h_{t+1}^2\right)\right.$$

$$\frac{\partial L}{\partial W}=\sum_{t=1}^{\tau}\frac{\partial L}{\partial h_t}\times\frac{\partial h_t}{\partial W}=\sum_{t=1}^{\tau}\operatorname{diag}\left(1-h_t^2\right)\delta_t h_{t-1}^T,\frac{\partial L}{\partial b}=\sum_{t=1}^{\tau}\frac{\partial L}{\partial h_t}\times\frac{\partial h_t}{\partial b}=\sum_{t=1}^{\tau}\operatorname{diag}\left(1-h_t^2\right)\delta_t$$

$$\frac{\partial L}{\partial U}=\sum_{t=1}^{\tau}\frac{\partial L}{\partial h_t}\times\frac{\partial h_t}{\partial U}=\sum_{t=1}^{\tau}\operatorname{diag}\left(1-h_t^2\right)\delta_t x_t^T$$

However, as a neural network model with high model complexity, RNN has the problems of gradient vanishing and gradient explosion, both of which affect the convergence of model training, reduce the convergence speed, and increase the difficulty of network learning for long-distance dependence, and the next LSTM and GRU models are improved on the RNN model to alleviate these two problems of the traditional RNN model.

A.2.2 Long Short-Term Memory

The essential reason for the gradient vanishing is that the hidden layer states are computed in such a way that the gradient is directly represented in the form of a concatenated product, whereas Hochreater and Schmidhuber proposed the LSTM model in 1997, which mitigates the gradient vanishing problem of the traditional RNN by hidden layer neurons when going from the output to the acquisition of intermediate state h_t. The following figure shows the hidden state structure of the LSTM model22. Compared with RNN there is one more new hidden state as cell state and forgetting gate, input gate and output gate three gating structures.

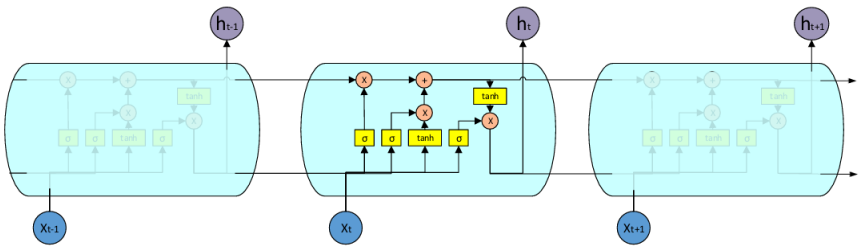


Figure 22: Hidden State Network Model Structure for LSTM

The forget gate is responsible for controlling whether or not to forget the hidden cell state of the previous layer with a certain probability, which also represents the probability of forgetting the cell state of the previous layer.

$$f_t = \sigma\left(W_f h_{t-1} + U_f x_t + b_f\right)$$

The input gate is responsible for controlling whether or not to incorporate the current moment's input x_t into the cellular state, and also represents the probability of remembering this layer of input information.

$$i_t = \sigma\left(W_i h_{t-1} + U_i x_t + b_i\right), \tilde{C}_t = \tanh\left(W_C h_{t-1} + U_C x_t + b_C\right)$$

And before the information goes to the output gate, the current cell state is updated based on the results of the forget gate and the input gate, where the old state is multiplied by the probability of needing to forget, plus the new candidate value multiplied by the rate of needing to update.

$$C_t = C_{t-1} \odot f_t + i_t \odot \tilde{C}_t$$

The output gate is responsible for the filtering of the current cell state, with the aim of generating the hidden state h_t from the cell state that has been affected by the forget gate and the input gate, and also to represent which parts of the cell state need to be output.

$$o_t = \sigma\left(W_o h_{t-1} + U_o x_t + b_o\right), h_t = o_t \odot \tanh\left(C_t\right)$$

The specific derivation process of the hidden state ht can be obtained by the above equation, and it can be found that h_t in the LSTM model needs to be mentioned not only by h_{t-1}, but also by the cell state. This changes the form in which the gradient is directly represented as a concatenated product, and as long as the forget gate is opened, the gradient of the cell state can effectively be passed backwards to the previous cell state, thus alleviating the problem in the RNN model.

$$h_t = o_t \odot \tanh\left(C_{t-1} \odot f_t + i_t \odot \tanh\left(W_C h_{t-1} + U_C x_t + b_C\right)\right), C_t = C_{t-1} \odot f_t + i_t \odot \tilde{C}_t$$

A.2.3 Gated Recurrent Unit

The GRU model is an improvement of the LSTM model in 2014, which combines the forgetting gate and the input gate in the LSTM model into a single update gate, which is simpler to construct compared to the LSTM, and reduces a gating structure thus reducing a lot of matrix multiplication operations, and the following figure shows the hidden state structure of the GRU23.

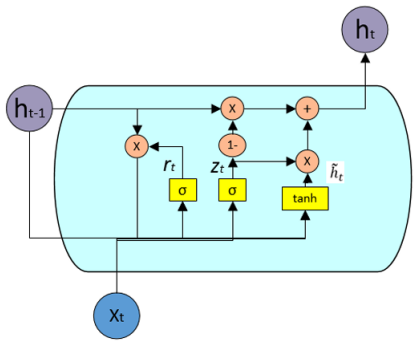


Figure 23: Hidden State Network Model Structure for GRU

The reset gate is used to control the effect of the previous moment's hidden state on the current hidden state, while the update gate is partly used to decide whether to ignore the current input variables, when open the current hidden state is determined by the previous moment and the current input variables, and when closed the current hidden state is determined by the previous moment only.

$$r_t = \sigma\left(W_r h_{t-1} + U_r x_t + b_r\right), z_t = \sigma\left(W_z h_{t-1} + U_z x_t + b_z\right)$$

$$\tilde{h}_t = \tanh\left(W r_t \odot h_{t-1} + U x_t + b_f\right), h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

B Predicted Results under the Three Models

pred_start_date	hidden_size	learning_rate	batch_size	dropout_rate	in_sample_ic	out_sample_ic	ic_difference(out - in)
2012-1-1	224	0.01	8192	0.2	0.07795349	0.061665752	-0.016287738
2012-7-1	160	0.05	16384	0.2	0.09872215	0.071299257	-0.027422893
2013-1-1	32	0.03	8192	0.2	0.098573332	0.054785701	-0.043787631
2013-7-1	128	0.001	4096	0.2	0.076147307	0.072113854	-0.004033453
2014-1-1	320	0.001	512	0.25	0.067458798	0.042745674	-0.024713125
2014-7-1	32	0.005	512	0.1	0.082708614	0.052802561	-0.029906053
2015-1-1	192	0.01	512	0.15	0.062797323	0.066134233	0.00333691
2015-7-1	128	0.05	2048	0.25	0.128217603	0.08578346	-0.042434143
2016-1-1	128	0.005	4096	0.3	0.078110977	0.063706088	-0.014404889
2016-7-1	224	0.005	8192	0.3	0.102920583	0.060408881	-0.042511702
2017-1-1	160	0.05	4096	0.25	0.119760758	0.108288145	-0.011472613
2017-7-1	64	0.01	1024	0.3	0.121399903	0.088850753	-0.03254915
2018-1-1	32	0.001	2048	0.1	0.121378241	0.096511312	-0.024866929
2018-7-1	192	0.03	16384	0.1	0.135177158	0.106325879	-0.028851278
2019-1-1	128	0.05	8192	0.25	0.111512954	0.094015557	-0.017497397
2019-7-1	96	0.0001	1024	0.1	0.047789896	0.054259575	0.006469678
2020-1-1	96	0.01	16384	0.2	0.090219603	0.075121737	-0.015097866
2020-7-1	224	0.001	512	0.1	0.062308638	0.044136826	-0.018171812
2021-1-1	320	0.03	4096	0.3	0.09206786	0.110301421	0.018233561
2021-7-1	288	0.005	512	0.25	0.054243888	0.049502258	-0.00474163
2022-1-1	320	0.05	2048	0.25	0.105654249	0.090579719	-0.01507453
2022-7-1	160	0.03	512	0.2	0.064844943	0.084475198	0.019630254
2023-1-1	288	0.01	8192	0.3	0.100311955	0.083121555	-0.0171904
2023-7-1	224	0.0001	1024	0.15	0.083368089	0.069038249	-0.01432984
2024-1-1	96	0.03	512	0.25	0.086442311	0.083878487	-0.002563823
2024-7-1	288	0.001	16384	0.2	0.10274551	0.089789735	-0.012955775

Table 11: Predict Results (RNN Model)

pred_start_date	hidden_size	learning_rate	batch_size	dropout_rate	in_sample_ic	out_sample_ic	ic_difference(out - in)
2012-1-1	224	0.03	1024	0.2	0.065636481	0.047566922	-0.01806956
2012-7-1	256	0.03	8192	0.1	0.056365656	0.038391296	-0.01797436
2013-1-1	320	0.0001	512	0.1	0.092642763	0.060930355	-0.031712408
2013-7-1	96	0.001	4096	0.25	0.075544981	0.064705377	-0.010839603
2014-1-1	256	0.03	1024	0.15	0.049013965	0.04497081	-0.004043155
2014-7-1	32	0.03	4096	0.1	0.077755395	0.059064126	-0.018691269
2015-1-1	288	0.01	512	0.1	0.106749553	0.090903399	-0.015846154
2015-7-1	96	0.05	512	0.15	0.136843194	0.077785797	-0.059057397
2016-1-1	128	0.05	2048	0.3	0.07214761	0.050383298	-0.021764312
2016-7-1	192	0.03	4096	0.25	0.072892328	0.047023247	-0.025869081
2017-1-1	288	0.01	1024	0.1	0.130218012	0.062892399	-0.067337326
2017-7-1	288	0.03	1024	0.2	0.093024254	0.055778821	-0.037245433
2018-1-1	32	0.005	2048	0.1	0.094367851	0.048534196	-0.045833655
2018-7-1	192	0.01	16384	0.25	0.127825259	0.094658134	-0.033167125
2019-1-1	224	0.01	8192	0.25	0.073956956	0.093111551	0.019154596
2019-7-1	256	0.01	512	0.15	0.07569795	0.039308376	-0.036389574
2020-1-1	96	0.05	1024	0.15	0.08183611	0.054330933	-0.027505176
2020-7-1	192	0.0001	4096	0.1	0.05352338	0.046997159	-0.006526221
2021-1-1	64	0.01	4096	0.2	0.086911424	0.05131511	-0.035596313
2021-7-1	192	0.03	8192	0.1	0.059793518	0.051632258	-0.00816126
2022-1-1	64	0.05	1024	0.2	0.060345064	0.045855807	-0.014489257
2022-7-1	128	0.03	2048	0.15	0.084178545	0.07004559	-0.014132955
2023-1-1	320	0.03	4096	0.1	0.048504733	0.062838828	0.014334095
2023-7-1	96	0.01	16384	0.25	0.082493678	0.061974032	-0.020519646
2024-1-1	320	0.0001	2048	0.1	0.121996583	0.072030832	-0.049965751
2024-7-1	288	0.05	256	0.3	0.095543756	0.064137699	-0.031406057

Table 12: Predict Results (LSTM Model)

pred_start_date	hidden_size	learning_rate	batch_size	dropout_rate	in_sample_ic	out_sample_ic	ic_difference(out - in)
2012-1-1	288	0.05	2048	0.3	0.063784441	0.067751449	0.003967007
2012-7-1	288	0.05	256	0.1	0.072864251	0.058122844	-0.014741407
2013-1-1	64	0.03	512	0.25	0.06034471	0.025461735	-0.034882975
2013-7-1	96	0.005	4096	0.1	0.063692968	0.05110554	-0.012587428
2014-1-1	64	0.01	4096	0.15	0.060235996	0.054064	-0.006171996
2014-7-1	32	0.001	256	0.2	0.084005093	0.09899738	0.014992287
2015-1-1	96	0.001	256	0.25	0.093164598	0.077196987	-0.01596761
2015-7-1	64	0.03	256	0.25	0.141045688	0.06123151	-0.079814178
2016-1-1	320	0.05	16384	0.2	0.13675201	0.110281051	-0.026470959
2016-7-1	64	0.01	4096	0.3	0.08914639	0.028444573	-0.060701817
2017-1-1	320	0.03	256	0.3	0.018191545	0.016951286	-0.001240259
2017-7-1	128	0.001	16384	0.2	0.082516423	0.071212839	-0.011303584
2018-1-1	160	0.01	8192	0.2	0.079260994	0.070857655	-0.008403339
2018-7-1	224	0.01	4096	0.15	0.112670369	0.079654106	-0.033016263
2019-1-1	160	0.005	4096	0.2	0.081016819	0.061904621	-0.019112198
2019-7-1	96	0.05	16384	0.2	0.109083283	0.099329437	-0.009753846
2020-1-1	256	0.0001	512	0.2	0.079227444	0.053713867	-0.025513577
2020-7-1	160	0.05	8192	0.3	0.093124695	0.103480939	0.010356244
2021-1-1	32	0.05	8192	0.1	0.066043352	0.054235782	-0.01180757
2021-7-1	256	0.05	512	0.25	0.061480193	0.060203978	-0.001276216
2022-1-1	160	0.03	2048	0.3	0.056905132	0.038825204	-0.018079928
2022-7-1	320	0.0001	16384	0.3	0.077209459	0.067293408	-0.009916052
2023-1-1	320	0.005	1024	0.2	0.090241589	0.055916512	-0.034325077
2023-7-1	128	0.05	512	0.2	0.016501809	0.014773336	-0.001728473
2024-1-1	192	0.01	8192	0.2	0.122007397	0.066621713	-0.055385684
2024-7-1	160	0.03	4096	0.25	0.091037171	0.055637883	-0.035399288

Table 13: Predict Results (GRU Model)

C Backtest Results

Note: This table presents the results of XGBoost and GRU model backtest in different variable settings. The first column is $barra\_limit$ and the first row is num_stock. The three data in each setting are $return$, $(sharpe\ ratio)$ and $[maximum\ drawdown]$

	XGB **100**	XGB **200**	XGB **300**	GRU **100**	GRU **200**	GRU **300**
0.1	0.21	0.15	0.12	0.09	0.09	0.09
	(0.86)	(0.69)	(0.6)	(0.41)	(0.43)	(0.43)
	[-0.11]	[-0.13]	[-0.12]	[-0.09]	[-0.1]	[-0.1]
0.2	0.23	0.17	0.14	0.09	0.09	0.09
	(0.93)	(0.74)	(0.64)	(0.41)	(0.43)	(0.4)
	[-0.12]	[-0.13]	[-0.12]	[-0.14]	[-0.11]	[-0.11]
0.3	0.25	0.18	0.15	0.09	0.1	0.09
	(1.0)	(0.79)	(0.68)	(0.42)	(0.44)	(0.41)
	[-0.12]	[-0.14]	[-0.12]	[-0.14]	[-0.13]	[-0.12]
0.4	0.26	0.19	0.16	0.09	0.1	0.09
	(1.02)	(0.83)	(0.71)	(0.43)	(0.44)	(0.41)
	[-0.13]	[-0.13]	[-0.11]	[-0.13]	[-0.13]	[-0.13]
0.5	0.28	0.21	0.17	0.1	0.1	0.09
	(1.06)	(0.86)	(0.76)	(0.43)	(0.44)	(0.42)
	[-0.12]	[-0.12]	[-0.12]	[-0.14]	[-0.15]	[-0.14]
1	0.49	0.42	0.40	0.23	0.22	0.22
	(0.46)	(0.43)	(0.80)	(0.23)	(0.24)	(0.22)
	[-0.22]	[-0.22]	[-0.16]	[-0.24]	[-0.25]	[-0.24]

Table 14: Backtest Results of XGB and GRU in Different Variable Settings