Machine Learning for Empirical Asset Pricing and Risk Premia Forecasting

Nate Huang, Zhilin Liu, Zibin Zhen

December 21, 2020

Abstract

This article aims to explore the possibility of using two state-of-the-art hybrid methods, ESRNN and Telescope, which both combine traditional statistical methods with machine learning methods, more specifically, deep learning and tree-based regression methods, in empirical asset pricing. ESRNN is proven to have the best performance among all other algorithms in the M4 Competition. Telescope is an improved algorithm inspired by ESRNN. The two algorithms both first extract the trend and seasonality, then build machine learning models on the resulting data. We conduct empirical study on the two methods and find that, although they present some validity on asset pricing, there are issues that need to be addressed.

1 Introduction

Machine learning can be applied in two problems in empirical asset pricing: factor models and time series forecasting. Traditional statistical methods often have a probabilistic interpretation but have difficulties to recognize more complex patterns. On the other hand, machine learning methods can recognize complex patterns but are hard to interpret. Machine learning techniques work well in handling data with no serial correlations. More and more researchers explore the potential methods to apply machine learning in time series forecasting. In the financial time series domain, the literature has accumulated a staggering list of predictors that researchers have argued possess forecasting power for returns. One application of machine learning in empirical asset pricing is predictor selection. Another application is to use deep learning directly in factor models. Compared to traditional statistical methods, machine learning can accommodate a much richer specifications of functional form. Gu [7] has shown that machine learning methods in empirical asset pricing yield larger economic gains than leading regression-based strategies. Adding nonlinearity to them, tree-based and neural-network-based models address the interactions from nonlinear predictors that are often missed by other methods. As function approximations, tree-based methods and neural networks do not require specifications on the functional form. However, limited literature sheds light on using machine learning in financial time series forecasting. Indeed, applying machine learning techniques directly to time series might yield spurious results. In addition, financial time series have relatively short length of history, which is not enough to obtain a good model. Recently, the M4 Competition has shown that a combination of time series techniques with carefully designed machine learning procedures beats individual statistical or machine learning method in many domains of time series forecasting. The M4 award-winning model, exponential smoothing and recurrent neural network (ESRNN) has exhibited promising results in time series forecasting. Although the model is tested with financial data, its accuracy on empirical asset pricing remains unknown. Extending from ESRNN, Bauer et al. [1] develops an automatic feature extraction and transformation approach (Telescope) in time series forecasting. Bauer et al. [1] claims that Telescope beats ESRNN in both efficiency and accuracy. We are interested in testing the predictability of these two models in empirical asset pricing. In this project, we take the following steps: 1. research the cutting-edge time series forecasting methods and select methods that we think are most promising in empirical asset pricing; 2. design and build a backtesting framework which is adaptive to different time series forecasting methods; 3. study the characteristics of model and portfolio metrics, then select the ones that we think are most relevant in wealth management; 4. backtest the two methods mentioned above and analyze the results; 5. propose potential improvements on the models for future work.

The remaining article is dedicated to four parts. Part 2 introduces the hybrid methods that we use in our project with detailed implementation. Part 3 describes the coding framework that we design to accommodate different forecasting methods, and describes our model tuning approach. Part 4 presents an empirical analysis on the two hybrid methods with a discussion on results. Finally, part 5 concludes the project.

2 Statistical and Machine Learning Models

Both traditional time series methods and machine learning models have their own limitation on time series forecasting. When the underlying structure of the time series is complicated, traditional time series methods suffer from low predictive power. In the similar way, machine learning models only work on stationary data with a long enough history of data points, which is not the case for financial time series. This makes machine learning hard to capture the underlying distribution and hence to generate robust results. Consequently, combining these two categories of methods seems to be a good way to resolve these drawbacks. In this section, we present two hybrid models that combine the power of traditional time series methods and that of machine learning techniques. Both models first extract time series characteristics from input data, transforming it into stationary time series. Then, they use the stationary data to train machine learning models. During the forecasting phase, the previously extracted characteristics are added back upon the machine learning output to produce final forecasting results. These two models have been proven to have better predictability on general time series forecasting than using traditional time series methods or machine learning models alone.

2.1 ESRNN

ESRNN is the winner of the M4 Competition, an international series of competitions whose purpose is to bring the most accurate forecasting method(s) for different types of time series datasets. ESRNN, short for Exponential Smoothing and Recurrent Neural Network, is a hybrid method that effectively shows the predictability of machine learning combined with statistical models. The internal architecture of ESRNN has three main elements. First, it performs exponential smoothing to de-seasonalize and normalize input data; then it uses a global LSTM network to generate forecast results; and in the final stage, the ensemble stage, ESRNN aggregates all the results that are made in previous steps.

2.1.1 Exponential Smoothing

Exponential smoothing is a technique to smooth time series data using exponential decay weights. In ESRNN, a simplified version of Holt-Winters' seasonal method is used as follows:
For non-seasonal models:

$$l_t = \alpha y_t + (1-\alpha) l_{t-1} \tag{1}$$

For single seasonality models:

$$l_t = \alpha y_t / s_t + (1 - \alpha) l_{t-1} \tag{2a}$$

$$s_{t+K} = \beta y_t / l_t + (1 - \beta) s_t \tag{2b}$$

For double seasonality models:

$$l_t = \alpha y_t / (s_t u_t) + (1 - \alpha) l_{t-1} \tag{3a}$$

$$s_{t+K} = \beta y_t / (l_t u_t) + (1 - \beta) s_t \tag{3b}$$

$$u_{t+L} = \gamma y_t / (l_t s_t) + (1 - \gamma) u_t \tag{3c}$$

where y_t is the value of the series at point t; l_t, s_t, and u_t are the level, seasonality, and second seasonality components; K is the number of data point per seasonal period; L is the number of data point per second seasonal period; α, β and γ are the smoothing coefficients between 0 and 1.

The level and seasonality components are calculated for all points in each series using above formulas. These components are then used to de-seasonalize and adaptively normalize the data during preprocessing stage of ESRNN.

2.1.2 Recurrent Neural Network

Once the input time series data is smoothed, the NNs come to make early-stage forecasts. The output results are generated as following:
For non-seasonal models:

$$\hat{y}_{t+1..t+h} = exp(NN(x)) * l_t \tag{4}$$

For single seasonality models:

$$\hat{y}_{t+1..t+h} = exp(NN(x)) * l_t * s_{t+1..t+h} \tag{5}$$

For double seasonality models:

$$\hat{y}_{t+1..t+h} = exp(NN(x)) * l_t * s_{t+1..t+h} * u_{t+1..t+h} \tag{6}$$

where x is the smoothed input vector; $NN(x)$ is the NN output vector; l_t means the level at time t; h is our forecasting horizon.

The neural network architecture built by Smyl [11] is illustrated in Figure 1. The basic idea of ESRNN's network is a dilated-LSTM stack that consists of several blocks. For each block, it involves one to four dilated-LSTM layers. Figure 1 shows an example of this architecture, where we start from an input size of seven at the bottom, and then use a two-block structure in the vertical direction. The first block consists of two layers with dilations of 1 and 2, respectively. Another block are layers with dilations of 4 and 8. For each dilated-LSTM layer, shown in the horizonal direction in Figure 1, it uses part of its input the hidden state from previous, but not necessarily the nearest cells. For example, when we have a layer of dilation 4, the hidden state at time t is taken from the cell at $t - 4$. This dilated LSTM structure has been proven by Smyl [11] as the most effective and efficient design in improving long-term memory performance in time series scenarios.

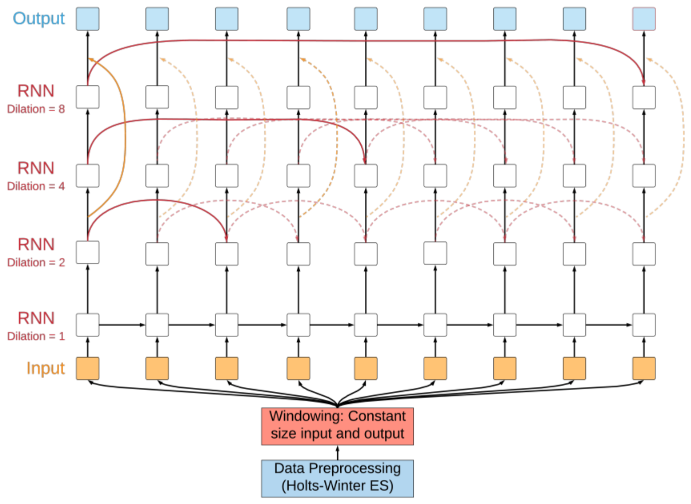

Output
RNN
Dilation = 8
RNN
Dilation = 4
RNN
Dilation = 2
RNN
Dilation = 1
Input
Windowing: Constant size input and output
Data Preprocessing (Holts-Winter ES)

Figure 1: ESRNN architecture

2.1.3 Ensemble Strategy

The last main technique used by ESRNN is ensemble. In the M4 Competition, the given input time series come from a wide range of data sources in different frequencies and in different industries. This makes the ensemble part reasonable and necessary.

To improve forecasting accuracy, ESRNN put forecast results all together at three levels of ensemble. The first level involves multiple independent model trains, which aims to mitigate the effect of randomness from parameter initiation. The second ensemble level is called "ensemble of specialists," which is designed to reduce the effect of uncertainty from the data source variety. At this level, the output forecasts are produced by averaging those generated by a pool of models with specialized subsets of series as their input and with specialized parameters for each run. The final level of ensemble is at the final stage of the entire training process, where the output forecasts are generated and finalized by averaging the results from the two previous levels.

2.2 Telescope

Telescope extracts and transforms features from its input time series, and then uses them to generate an optimized forecast model. There are three phases in Telescope's framework. The first phase is a preprocessing phase. In this phase, Telescope takes a time series and extracts features from it. The second phase is called model building phase, during which Telescope takes the extracted features and trains a XGBoost model. Forecasting is done in the third phase. The key step in this final phase is adding the previously extracted characteristics back to the predicted output from the XGBoost model.

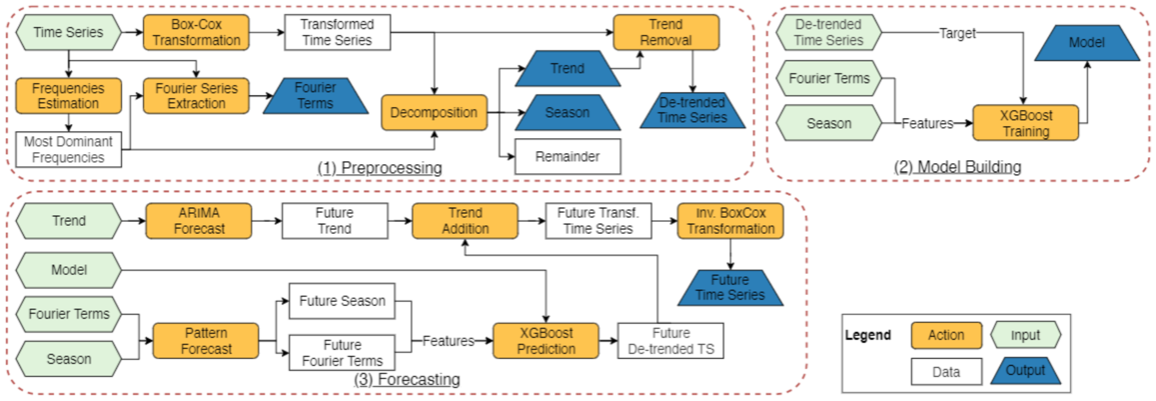


Figure 2: Telescope workflow

2.2.1 Preprocessing

The preprocessing phase starts with taking a time series as an input and performing frequency estimation on it. More specifically, a periodogram is applied on the input time series to retrieve all the frequencies. Then the framework will match the frequencies to the most reasonable natural time units frequencies. At the same time, frequencies with the threshold $\geq 50\%$ of the spectral value are defined by Telescope as the most dominant frequency. Only the most dominant frequencies that match the natural frequencies within a small tolerance are extracted as Fourier terms, the terms that serve as the input for model building and forecasting.

Because machine learning methods cannot handle changing variance and multiplicative effect, the Telescope framework performs a Box-Cox transformation to reduce these effects. In addition, Telescope expects that the time series present with trend and/or seasonality patterns, thus non-stationary, and hence it uses STL(Seasonal and Trend decomposition using Loess) to split the transformed time series with the most dominant frequency into three components: trend, season, and remainder. The trend is used as input for the forecasting phase, and the season is used for both the model building and forecasting phases. The trend is removed from the transformed time series to make the time series stationary. The resulting de-trended time series is used as input for the model building phase.

2.2.2 Model Building

In the model building phase, the extracted Fourier terms, seasonality terms, and de-trended time series terms from the preprocessing phase are passed in. Then, the framework applies XGBoost as its default machine learning regression method on these features. XGBoost is a tree-based method that can capture the nonlinear structure of the data. Moreover, it is efficient, accurate, easy to interpret, and is proven to outperform other machine learning techniques in time series forecasting. With XGBoost, the framework models the de-trended time series based on the Fourier terms and seasonality, and finally uses this model to generate predictions in the forecasting phase. Note that Telescope assumes the remainder to be the noise in the data, and assumes that the trend increases the variance, making the time series violate the stationary assumption. Therefore, the remainder and trend are excluded in the model building process for a purpose of forecasting error reduction.

2.2.3 Forecasting

In the forecasting phase, the trend, seasonality and Fourier terms from the preprocessing phase and the XGBoost model from the model building phase are used as the input. As discussed in the Model Building section, the future seasonality and Fourier terms are passed to the XGBoost model to generate future prediction. To get the future seasonality and Fourier terms, as they are recurring patterns by definition, the model recognizes the patterns and generate them. The future seasonality and Fourier terms are then passed to the XGBoost model. The output of the XGBoost model is a de-trended forecast.

Then the framework rebuilds the true forecasts by adding the trend back. A future trend must be forecasted. Since there is no recurrent pattern on future trend, an ARIMA model is used to forecast the future trend. Then the future trend is added to the output de-trended forecast. Because Box-Cox transformation is applied on the time series in the preprocessing phase, the inverse Box-Cox transformation is applied to the output, and the result is the final forecast.

3 Model Calibration and Testing

3.1 Code Framework

We adopt object orientated programming paradigm and design a testing framework which allows us to test on a variety of data and forecasting models. In general, our code framework, shown in Figure 3, consists of three main categories: 1) data objects in green and blue boxes; 2) model objects in red boxes; and 3) evaluation modules in orange and yellow boxes. All classes and modules implemented are independent with each other, which is helpful for our further work. In other words, we do not need to make huge modifications, while new models can be easily tested and the overall functionality works properly.

In this regard, data flows in the following way: firstly, raw data is downloaded as csv files. For each csv file, it contains one time series for a specific asset ticker. The class DataSeries is used to store the data from each ticker, in which we provide functionalities for basic operations that we will be using for later procedures. To aggregate all the data series, another class, DataCollection is created as a basic unit of model calibration and testing parts. In other words, a DataCollection contains a composition of different tickers or time periods; and after collecting all DataSeries into one DataCollection object, we use it as a global input to all the forecasting model objects. Within each model object, we add specific reformatting methods to connect DataCollection with the models we examine, playing a role as a model interface. The output of model objects is stored as a new DataCollection, which is in turns used as an input of evaluation modules. At the end of the entire flow, plots and tables of metrics can be visualized clearly.

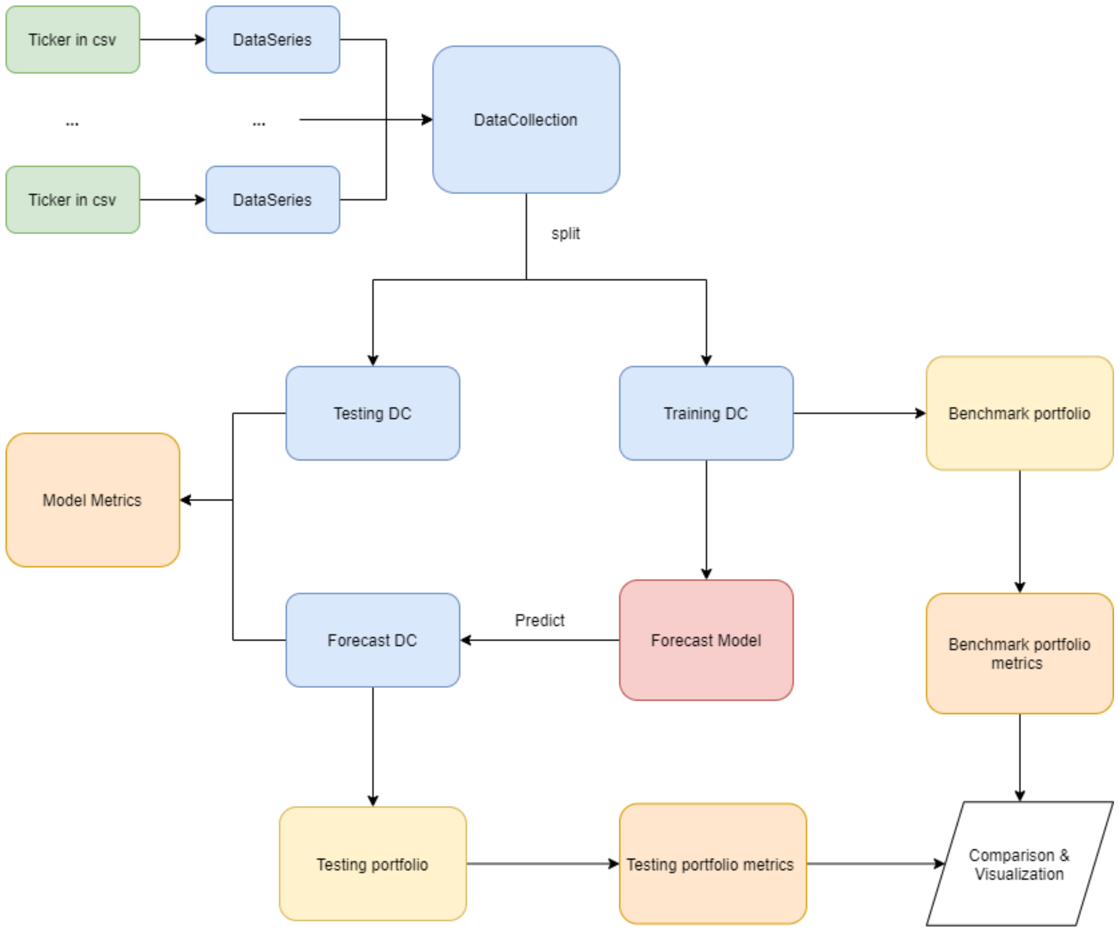


Figure 3: Data-flow diagram

3.2 Hyperparameter Tuning

In machine learning, hyperparameter tuning is a process of choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is used to control the learning process. In other words, hyperparameters are values that must be specified outside of the training procedure.

The traditional way to optimize hyperparameters is using K-fold cross validation, which is proved to be an effective approach for independent series of data. It is, however, not well-suited for time series with correlation involves in. Hence, a more robust method called Forward-Chaining Nested Cross Validation, shown in Figure 4, is used in this project. The main idea is that, for each set of hyperparameters, we first perform train-test split for several times on a rolling basis, then we train the model on the training subset from each split, and obtain the model's out-of-sample performance by the testing subset that immediately follows that training subset. The model performance on each split is evaluated based on mean absolute percentage error (MAPE). The average performance upon all splits provides us the estimated predictability of that model with that specific set of hyperparameters. Finally, the optimal set of hyperparameters is picked by balancing the model performance score and running time.

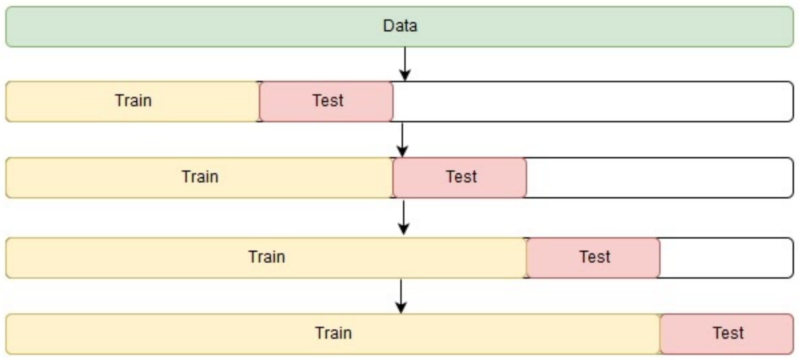


Figure 4: Forward-chaining nested cross validation

In ESRNN, there are more than 30 hyperparameters to tune. Besides the ones that are clearly specified by Smyl [11], we need to optimize the remaining 7 of them, which are maximum epoch, batch size, learning rate, dilations, seasonality, input size, and output size. Final choices of these parameters are shown in Appendix Section C.

4 Data Information

We retrieve historical prices of 108 tickers covering four asset classes including ETFs, commodity futures, mutual funds, and real estate, from Yahoo Finance based on trading volume. The period examined is from 01/03/2005 to 12/31/2019, a total of 3,863 trading days.

The data set is cleaned by the following steps. First, if a ticker has more than 120 missing values (3%), this ticker will be removed from our analysis. Second, we exclude trading days on which more than 60 tickers have missing values. Finally, missing data is imputed using the mean of a rolling window with a size of 5. After this cleaning procedure, our final dataset contains 71 tickers, and each ticker has 3,684 observations.

5 Empirical Analysis

In this section, we compare the prediction performance of ESRNN and Telescope on both asset level and portfolio level to gain an empirical insight. As shown in Figure 5, the data is split into set A, B, and C. Set A is a training set, which contains 3,000 observations and is used to train the models; set B is a testing set, which contains 90 data points and is used to test the prediction performance on asset returns. Finally, set C also contains 90 data points and is used to evaluate portfolio performance.

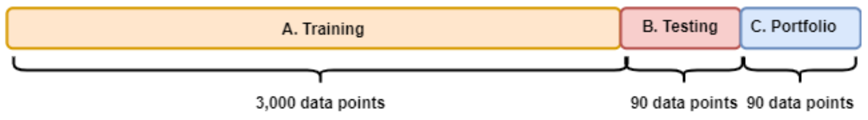


Figure 5: Data split for empirical analysis

In addition, we convert asset prices into returns in our Preprocessing module before feeding them into models. To accommodate the input data with various models, one extra step of standardization is performed as follows:

$$\tilde{r}_{i,t} = r_{i,t} - \min_{t' \in \mathcal{T}} r_{i,t'} + 1 \tag{7}$$

where $\tilde{r}_{i,t}$ is the input data to forecasting models, the standardized return for ticker i at time t. $r_{i,t}$ is the original return of asset i at time t.

After the predicted returns are generated by the models, they are converted back to normal returns.

5.1 Performance Metrics

5.1.1 Metrics for Models

The metrics that are used to compare model performances and to calculate loss are overall weighted average (OWA) and root mean squared error (RMSE).

To compare ESRNN and Telescope with the benchmark model, we use OWA, the performance measurement created and used in the M4 Competition. It is the weighted average of symmetric mean absolute percentage error (sMAPE) and mean absolute scaled error (MASE).

$$sMAPE = \frac{100\%}{n} \sum_{t=1}^{n} \frac{|\hat{Y}_t - Y_t|}{|\hat{Y}_t| + |Y_t|} \tag{8}$$

where $\hat{y}_t$ is the forecast value and y_t is the actual value.

$$MASE = \frac{\frac{1}{h} \sum_{t=n+1}^{n+h} |\hat{Y}_t - Y_t|}{\frac{1}{n-m} \sum_{t=m+1}^{n} |Y_t - Y_{t-m}|} \tag{9}$$

where $\hat{y}_t$ is the forecast value, y_t is the actual value, h is the forecasting horizon, n is the number of the data points available in-sample, and m is the time frequency of consecutive observations. The idea of MASE is to use the mean absolute forecast error from a naive forecasting strategy, which assumes that the actual value at time t equals to the actual value at time $t-m$, to scale the mean absolute prediction error from our model. In this project, we set m be the frequency (daily, i.e. $m = 1$) of our input time series.

$$OWA = \frac{1}{2}\left(\frac{Model\ sMAPE}{Benchmark\ sMAPE} + \frac{Model\ MASE}{Benchmark\ MASE}\right) \tag{10}$$

There are many model performance metrics available in literature. Each has its own strength and weakness. For example, sMAPE uses percetage errors that are scale-independent and intuitive to understand, but it is always equal to two for a zero actual value, regardless of the forecast that is used (Syntetos 2001) [12]. While, MASE is able to fix potential problems of sMAPE and to provide an alternative with better mathematical properties such as a defined mean and a finite variance, as stated in Franses (2016) [6]. Given such a lack of agreement in terms of metrics, we believe that the M4 Competition metric OWA is objective and decide to adhere to it. Besides, since the main purpose of the models is prediction, RMSE is another desired metric that we want to add into our project, for it is a popular absolute measure of fit. With OWA and RMSE, we should be able to identify the model that gives us the more accurate predictions.

5.1.2 Metrics for Portfolios

The metrics that are used to compare portfolio performance are annualized returns, annualized volatilities, Sharpe ratio, and maximum drawdown (MDD). The Sharpe ratio is defined as the annual return of an investment above the risk-free rate, also called the excess return, divided by the volatility of the return. In short, it is a measure of "reward per unit of risk." In a risk-neutral framework, a higher Sharpe ratio is what an investor often seeks for. It helps in getting the correct analysis of the funds and enhancing the returns on investment. However, as a single ratio, little is learned about the consistency of returns and the magnitude of potential loss. To remedy this weakness, MDD, defined as the single largest loss experienced over a defined time period, is considered because it focuses on capital preservation, which is a key concern of investments.

5.2 Model Performance

The asset return prediction performances of ESRNN and Telescope are compared with a baseline model. The baseline model is an average one-step Naive S forecast (i.e. forecasts are equal to the last known observation of the same period). In our case, one period consists of five observations. We believe that if we only train one ESRNN and one Telescope model and use the 90-day testing set to evaluate the performance, the result will not be accurate. Therefore, we apply the forward chaining method and train 7 models for each examined framework. For each model, the training set contains 3,000 observations for each ticker, and the testing set contains 90 observations. The average performances of the 7 ESRNN models and the 7 Telescope models are obtained.

As shown in Table 1, the OWA is 0.946 for ESRNN and 0.689 for Telescope. This means that on average, both models beat the benchmark. However, outperforming the baseline model is not an indication of accurate predictions. To visualize their accuracy, we pick one ticker from each asset class and show one of their forecasting graphs (See Figure 6 in Appendix).

Table 1: Model Metrics

	ESRNN	Telescope
RMSE	1.545%	1.157%
OWA	0.9456	0.6888

The graphs show that Telescope's forecast is less volatile than ESRNN's forecast. This is because Telescope only takes significant Fourier terms and seasonality terms as features into the XGBoost while ESRNN's underlying recurrent neural network captures more frequency terms. When there is no significant future trend predicted in Telescope's forecasting step, the forecasted time series tends to repeat itself after some seasons. Inherently, time series methods have a mean-reverting behavior. ESRNN also presents similar behavior, but, due to a more complex underlying structure, it can capture more volatile behaviors.

Both methods present underestimation on the volatility of asset returns. Compared to ESRNN, Telescope only recognizes seasonality that matches natural seasonality (hour, day, week, month), which is insignificant in asset returns. Therefore, Telescope's forecasts are all close to the mean over the forecast period, presenting a lack of volatility behavior. Although ESRNN captures more price volatilities than Telescope, it also presents underestimation on the forecast data. This is due to the fact that the level at the forecasting stage is averaging on all previous training data. A possible remediation is to estimate the forecast volatility based on a relative short period of data right before the forecast period, and then scale the forecast back to that volatility.

5.3 Portfolio Performance

Next, we compare ESRNN and Telescope at a portfolio-level. Even when a model obtains a good prediction at individual asset level, it may fail to accurately produce portfolio-level forecast since portfolio returns distribution is sensitive to the dependence among asset returns. Thus, assessing the forecast performance at portfolio-level will provide an additional indirect evaluation of the model and its robustness, allowing us to evaluate the models' ability and to transport their forecasting ability to a broader investment context.

In our study, two types of portfolios, minimum variance portfolios and efficient risk portfolios, are formed. A minimum variance portfolio is a collection of assets that combine to minimize the price volatility of the overall portfolio, while an efficient risk portfolio is collection of assets combined in a way that produces the highest possible returns at the given level of risk. These two types of portfolios heavily rely on the accurate estimation of the covariance matrix of asset returns. However, the historical covariance matrix often contains much noise. We are interested in assessing whether a more accurate covariance matrix can be obtained from the predicted returns, and thus, in constructing enhanced portfolios that could outperform the benchmark portfolios constructed using historical data in terms of annualized returns and Sharpe ratio while maintaining a small volatility.

The portfolios contain all the tickers that we use for our analysis. Since 7 models are trained, for each model, we construct the two enhanced portfolios as mentioned above using the predicted returns. The performances of these enhanced portfolios and of the benchmark portfolios are compared on the evaluation datasets.

For minimum variance portfolios evaluation, Table 2 shows that the enhanced portfolios significantly increase the Sharpe ratio relative to the benchmark portfolios on average. However, these portfolios' volatilities are also higher. For the portfolios constructed using predicted returns from Telescope, the average volatility is 10 times larger than that of the benchmark portfolios.

For efficient risk portfolios evaluation, we consider two risk levels: 0.05 and 0.2. From Table 2, we can see that when the target volatility is set to 0.05, the enhanced portfolios achieve higher annualized returns than the benchmark portfolios; the returns increase significantly with the Telescope enhanced portfolios. On the other hand, the enhanced portfolios' volatilities are much larger than the target volatility 0.05 when they are tested on the evaluation sets, which results in lower Sharpe ratios than the benchmark portfolios. When the target volatility is set to 0.2, the enhanced portfolios' returns are more than 5 times higher than the benchmark portfolios'. Although their Sharpe ratios are still lower than the benchmark, their volatilities are within the given target.

The results of portfolio performance indicate that both ESRNN and Telescope lack the ability to make accurate portfolio forecasts consistently. While the enhanced portfolios do obtain higher annualized returns, they are overly volatile. One possible explanation is that, as we mention above, both models underestimate the volatilities of asset returns. Therefore, the correlations of each security pairs will not be accurate. When forming the portfolios, if the return volatilities are largely underestimated, it is likely that the volatilities explode when they are tested on the evaluation sets.

Table 2: Portfolio Metrics

	Minimum Variance Portfolio		
	Benchmark	ESRNN	Telescope
Annu. Return	0.002	0.007	0.007
Annu. Volatility	0.010	0.018	0.102
Sharpe Ratio	-0.023	0.620	0.850
Max DrawDown	-0.005	-0.012	-0.050

	Efficient Risk Portfolio (Vol=0.05)		
	Benchmark	ESRNN	Telescope
Annu. Return	0.059	0.064	0.172
Annu. Volatility	0.038	0.167	0.238
Sharpe Ratio	1.615	0.414	0.764
Max DrawDown	-0.019	-0.103	-0.108

	Efficient Risk Portfolio (Vol=0.2)		
	Benchmark	ESRNN	Telescope
Annu. Return	0.020	0.138	0.113
Annu. Volatility	0.016	0.132	0.200
Sharpe Ratio	1.176	1.113	0.700
Max DrawDown	-0.007	-0.065	-0.090

6 Conclusion

In this project, we research on hybrid methods that use statistical methods and machine learning methods which yield better performance on time series forecasting. The statistical methods are used to identify and extract the time series characteristics of asset prices while the machine learning methods recognize more complex structure in time series forecasting. We conduct an empirical study on two methods: ESRNN and Telescope. We test the two methods by forecasting asset returns and using them to construct portfolios. While the result of our empirical study shows some validity on using these two methods in empirical asset pricing, there are drawbacks that need to be addressed. The primary problem is the underestimation of volatilities on asset return forecasts from both methods. Such underestimation leads to the volatile portfolios. The second issue is that Telescope is unable to recognize enough frequency terms to construct a tree-based model.

Going forward, in order to target the first issue, we propose to add an extra step: estimate the volatility on the forecasting period and re-scale the forecast. As for the second issue, we would explore the possibility of adding more features in Telescope. Furthermore, based on the prediction performance of ESRNN and Telescope, we consider that the price movements are too volatile to make accurate predictions. Meanwhile, macroeconomic or financial factors are often more stable. Hence, for future improvement, if we first use penalized regression, such as Lasso, to select significant factors, then predict their time series, and finally use factor models to forecast asset returns, better predictability should be achievable.

References

[1] Andr Bauer, Marwin Zfle, Nikolas Herbst, Samuel Kounev, and Valentin Curtef. Telescope: An automatic feature extraction and transformation approach for time series forecasting on a level-playing field. 04 2020.

[2] Alexei Botchkarev. A new typology design of performance metrics to measure errors in machine learning regression algorithms. *Interdisciplinary Journal of Information, Knowledge, and Management*, 14:45–79, 01 2019.

[3] Luyang Chen, Markus Pelger, and Jason Zhu. Deep learning in asset pricing, 2020.

[4] Guanhao Feng, Jingyu He, and Nicholas G. Polson. Deep learning for predicting asset returns, 2018.

[5] Guanhao Feng, Nicholas G. Polson, and Jianeng Xu. Deep learning in characteristics-sorted factor models, 2019.

[6] Philip Hans Franses. A note on the Mean Absolute Scaled Error. *International Journal of Forecasting*, 32(1):20–22, 2016.

[7] Shihao Gu, Bryan Kelly, and Dacheng Xiu. Empirical Asset Pricing via Machine Learning. *The Review of Financial Studies*, 33(5):2223–2273, 02 2020.

[8] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. The m4 competition: Results, findings, conclusion and way forward. *International Journal of Forecasting*, 34(4):802 – 808, 2018.

[9] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. The m4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 36(1):54 – 74, 2020. M4 Competition.

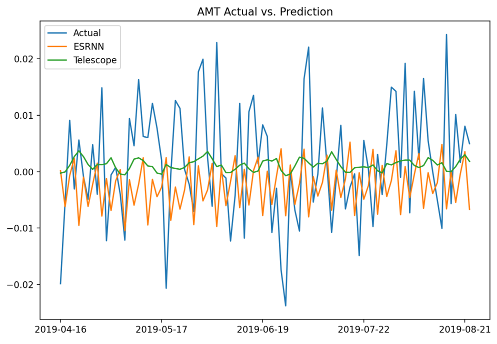
[10] Boriss Siliverstovs. Dissecting models' forecasting performance. *Economic Modelling*, 67:294 – 299, 2017.

[11] Slawek Smyl. A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *International Journal of Forecasting*, 36(1):75 – 85, 2020. M4 Competition.

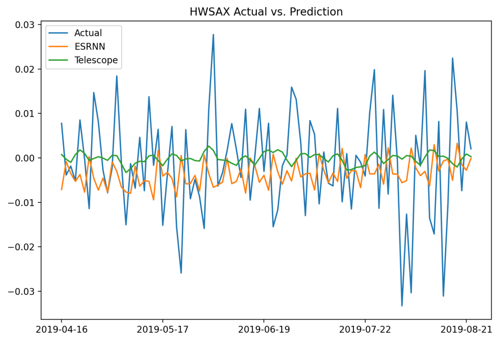
[12] A.A Syntetos and J.E Boylan. On the bias of intermittent demand estimates. *International Journal of Production Economics*, 71(1):457 – 466, 2001. Tenth International Symposium on Inventories.

Appendices

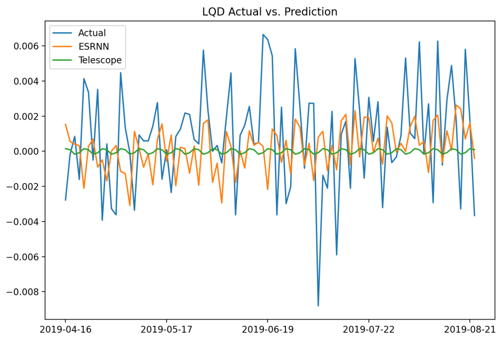
A Model Performance Graphs



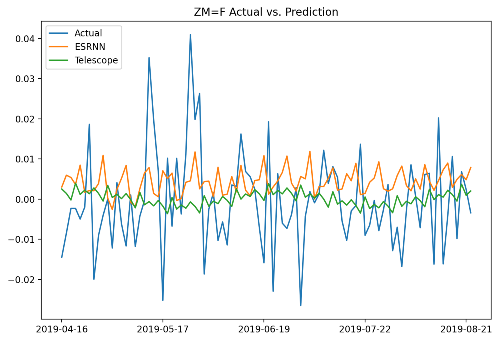
(a) AMT actual vs. predictions



(b) HWSAX actual vs. predictions



(c) LQD actual vs. predictions



(d) ZM=F actual vs. predictions

Figure 6: Model performance

B Portfolios Daily PnL Graphs

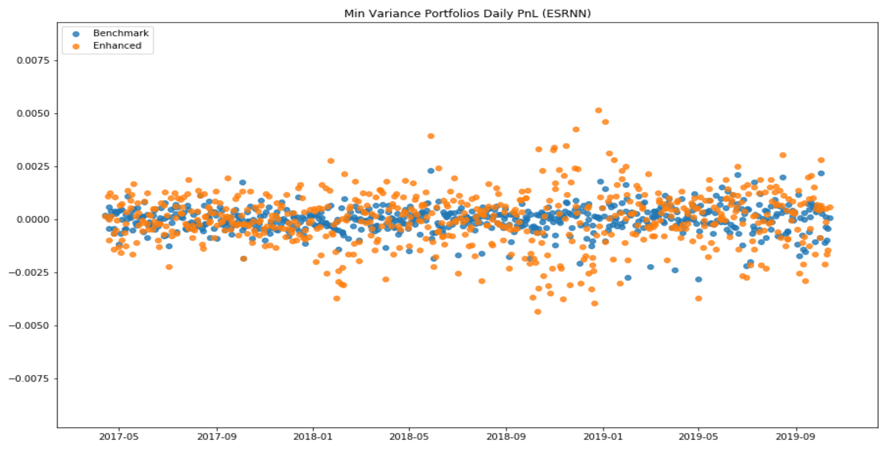


Figure 7: Minimum variance portfolios daily PnL (ESRNN)

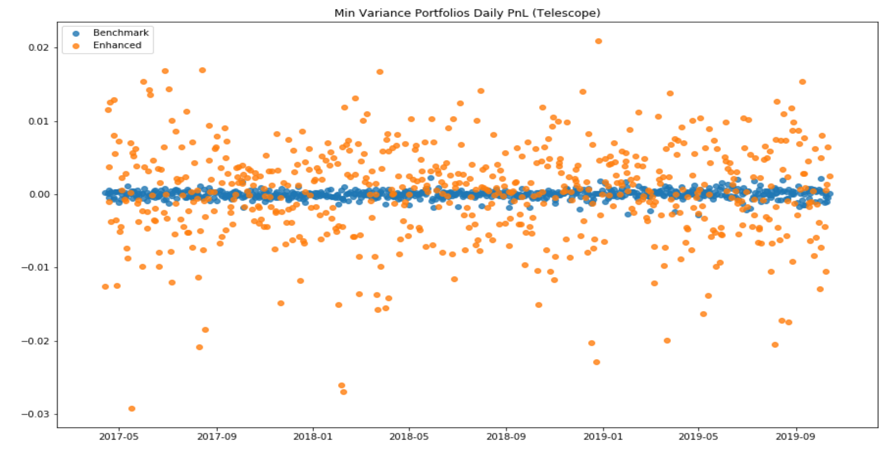


Figure 8: Mininum variance portfolios daily PnL (Telescope)

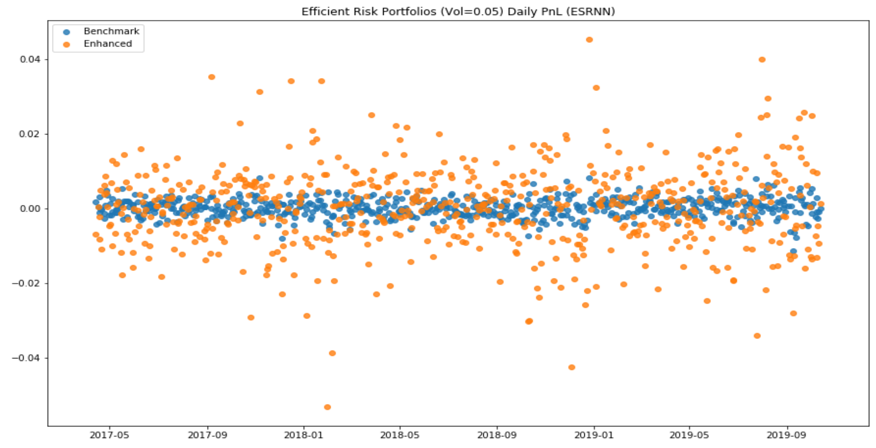


Figure 9: Efficient risk portfolios (vol=0.05) daily PnL (ESRNN)

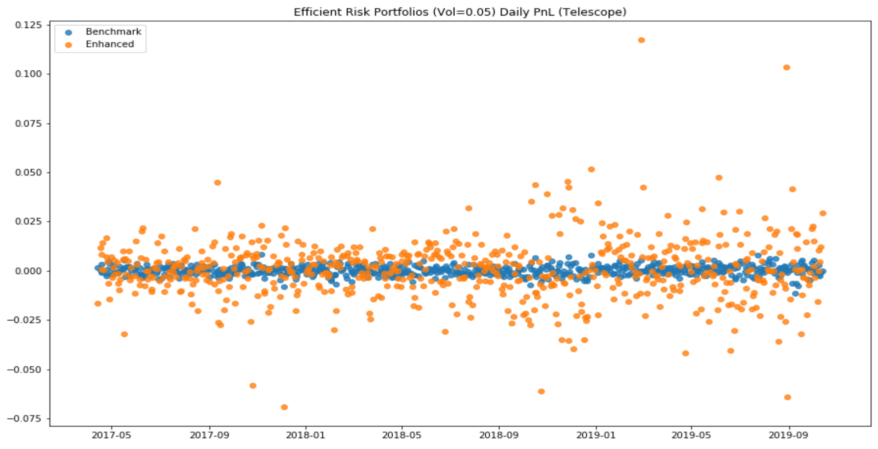


Figure 10: Efficient risk portfolios (vol=0.05) daily PnL (Telescope)

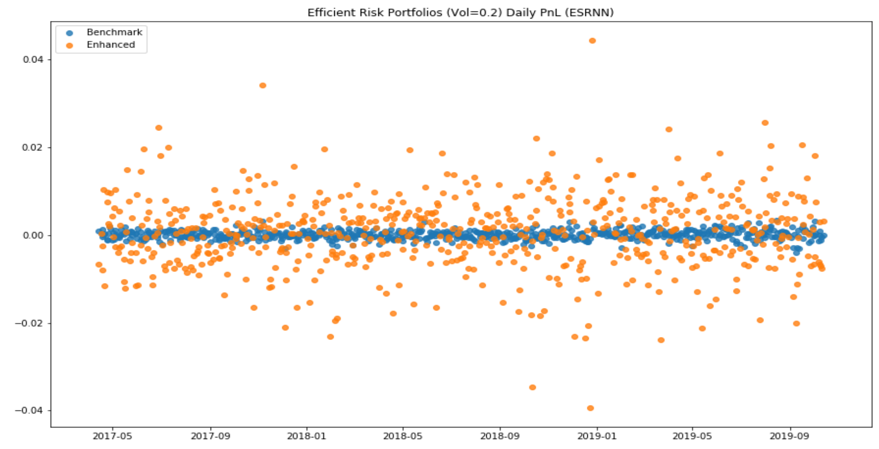


Figure 11: Efficient risk portfolios (vol=0.2) daily PnL (ESRNN)

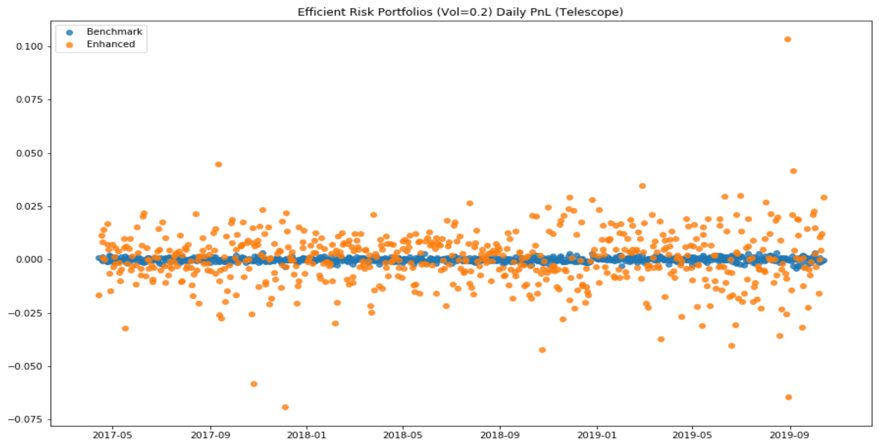


Figure 12: Efficient risk portfolios (vol=0.2) daily PnL (Telescope)

C ESRNN Hyperparameters Tuning Result

Table 3: Details of Hyperparameters

Epoch	Batch	Learning Rate	Dilations	Seasonality	Input Size	Output Size	Time (s)	Score
30	32	0.001	[[1,3],[5,10]]	[]	120	120	712	0.0691
15	64	0.001	[[1,3],[5,10]]	[]	30	120	56	0.0714
30	64	0.001	[[1,3],[5,10]]	[5]	30	30	155	0.0934
15	64	0.001	[[1,5,10]	[5]	60	60	161	0.0925
15	64	0.001	[[1,5,20]]	[5,20]	5	120	27	0.0748
15	64	0.001	[[1,3,5,10]]	[5]	5	60	27	0.0964
15	64	0.001	[[1,3],[5,10]]	[5]	5	60	23	0.0964
15	64	0.001	[[1,3,5],[10]]	[5]	5	60	22	0.0964
15	64	0.001	[[1,3,5,10]]	[5]	5	120	33	0.0746
15	64	0.001	[[1,3],[5,10]]	[5]	5	120	32	0.0730
15	64	0.001	[[1,3,5],[10]]	[5]	5	120	31	0.0732
15	64	0.001	[[1]]	[5]	5	120	29	0.0745
15	64	0.001	[[1],[5]]	[5]	5	120	33	0.0726
15	64	0.001	[[1],[5,20]]	[5]	5	120	34	0.0770
15	64	0.001	[[1,3],[5,20]]	[5]	5	120	33	0.0741
15	64	0.001	[[1,5],[10,20]]	[5]	5	120	35	0.0745
15	64	0.001	[[1,5]]	[]	5	120	11	0.0711
15	64	0.001	[[1,5]]	[]	30	60	52	0.0934
15	64	0.001	[[1,5]]	[5]	30	66	79	0.0900
15	64	0.001	[[1,5]]	[5]	60	120	175	0.0720
15	64	0.001	[[1,5]]	[5]	30	120	83	0.0726
15	64	0.001	[[1,5]]	[5]	120	120	233	0.0718
15	64	0.001	[[1,5]]	[5]	5	120	19	0.0704
15	64	0.001	[[1,5]]	[5]	5	132	19	0.0803
15	64	0.001	[[1,5]]	[5]	1	132	9	0.0894
15	64	0.001	[[1,5]]	[5]	0	132	7	0.1076
15	64	0.001	[[1,5]]	[5]	5	90	19	0.0621
15	64	0.001	[[1]]	[]	10	60	14	0.0952
15	64	0.001	[[1]]	[]	5	120	11	0.0723
15	64	0.0003	[[1]]	[]	5	120	11	0.0944
15	64	0.003	[[1]]	[]	5	120	11	0.0712
15	64	0.01	[[1]]	[]	5	120	11	0.0724
15	64	0.001	[[1]]	[]	30	252	419	0.1151
15	64	0.001	[[1]]	[]	30	120	53	0.0748
15	64	0.001	[[1]]	[]	10	120	19	0.0754
15	64	0.001	[[1]]	[5]	1	90	3	0.0682
15	64	0.001	[[1]]	[5]	2	90	3	0.0645
15	64	0.001	[[1]]	[5]	3	90	4	0.0641
15	64	0.001	[[1]]	[5]	5	90	8	0.0627
15	10	0.001	[[1]]	[5]	5	90	22	0.0636
15	64	0.001	[[1]]	[5]	10	90	20	0.0644
15	64	0.001	[[1]]	[1,5]	5	90	20	0.0649
15	64	0.001	[[1]]	[5,10]	5	90	12	0.0638