Capstone Project Report Yaoru Peng

AWS Machine Learning Engineer Nanodegree July 17, 2022

Prediction of Stock Returns Using AutoML and Neural Network (The Winston Stock Market Challenge by Kaggle)

Definition

Project Overview

Predicting future stock returns has always been a favorite topic for market investors. Market data is noisy, and some scholars even claim that short-term stock returns follow stochastic processes; in other words, returns are random and unpredictable. However, the existence of momentum in every stock market, no matter in the NYSE or Europe stock market, demonstrates the delicate relationship between stock past prices(returns) and future performance.

In this project, I will create a stock return predictor using the AutoGluon framework and deep learning models. The predictor will take several stock features (25), daily returns of previous days (2), and intra-day one-minute return for two hours (120) as the input, and it will predict the intra-day one-minute return for the next hour and also two future daily returns.

Problem Statement

The goal is to create a stock return predictor on both future intra-day one-minute returns and daily returns for the next few days. The tasks involved are listed below:

1. Fetch data from the Kaggle website using Kaggle API and process the stock data
2. Use train_test_split to generate training sets and validation sets
3. Train the return predictor within the AutoGluon framework, which will automatically choose the machine learning model with the best performance.
4. Train another predictor using a neural network, and compare the two predictors with chosen metrics.
5. After getting the better predictor, generate test outputs and submit results to Kaggle.

Evaluation Metrics

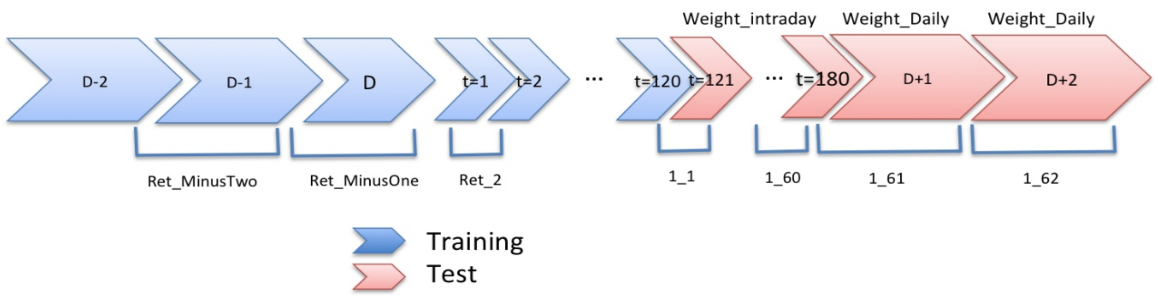
Root Mean Square Error (RMSE):

RMSE is a loss term widely used in machine learning, and it will penalize significant errors more than MAE (Mean Absolute Error). Since we do not want any significant deviation between expected return and real return, RMSE would be a better metric for training our predictor to avoid large errors.

Analysis

Data Exploration

The datasets used in this project are downloaded from Kaggle, containing a training set and a test set. Training sets include provided 25 features (Feature_1 - Feature_25) of stocks, the daily returns in days D-2, D-1 (Ret_MinusTwo, Ret_MinusOne), and intra-day returns (Ret_2 - Ret_120) in day D. It will also contain our target variables, the intra-day returns in the rest of day D (Ret_121 - Ret_180) and the daily returns in day D+1, D+2. Then, the test sets contain the same columns except for our target variables.



More accurately, our training data consists of 40,000 rows of stock data, and there are 120,000 rows of data consisting of Feature_1 - Feature_25, Ret_MinusTwo, Ret_MinusOne, and Ret_2 - Ret_120 for the test dataset.

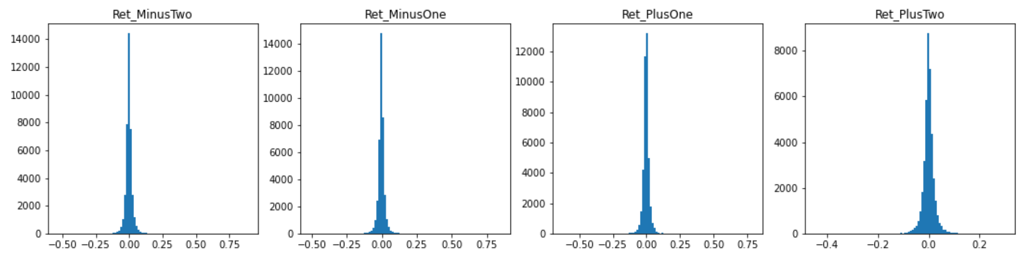
Exploratory Visualization

The provided datasets are quite noisy, consisting of many NaN values. The plot on the right shows how many percents of real data (not NaN, containing values) each feature has. We could see only 16% data in the Feature_1 column and <50% data in the Feature_10 column containing values, so we decide to drop those two features when we build the model.

```
Feature_1         0.167175
Feature_10        0.513225
Feature_2         0.771350
Feature_20        0.804350
Feature_4         0.806975
                    ...
Ret_135           1.000000
Ret_134           1.000000
Ret_133           1.000000
Ret_139           1.000000
Weight_Daily      1.000000
```

After that, we will fill the NaN values with pandas' built-in fillna method. The aim of this step is to build legit deep learning models for prediction (see the Data Processing Section)

To test the model assumption, we will test the distribution of our features and test variables. Most linear and regression models require the data to have balanced (normal) distributions; more accurately, the data distribution should be following a multi-normal one.



The figure above presents the distribution of our two most important features (Ret_MinusTwo, and Ret_MinusOne) and two of our target variables (Ret_PlusTwo, and Ret_PlusOne). Either of them shows an approximately normal distribution centered at 0 and spread with a small variance. Therefore, we could assume the data distribution is multi-normal, and the regression assumption-multinormal distribution is satisfied.

Algorithms and Techniques

In this project, we will test out different machine learning and deep learning techniques to build our predictor. The first technique we apply for our dataset is the process to split the training data into train data and validation data with a proper proportion. Since there is no opportunity for us to evaluate the performance of different predictors from the test data, we need to build our own validation data to help us choose the best predictor model. Then, to predict daily returns (Ret_PlusTwo, and Ret_PlusOne), we will apply both the AutoML framework and a neural network with an appropriate structure. AutoGluon is an intelligent framework that could automatically choose the best machine learning models for a specified problem (our project will be a regression-type problem). After we set up the right metric, “eval_metric="root_mean_squared_error",” the only hyperparameters for the AutoML model we want the tune would be the “time_limit” and “presets.” The larger the time limit is, the better performance the final resulting model will have. In this project, we will set “time_limit = 600” due to space constraints.

The following parameters can be tuned to optimize the AutoML model:

- Eval_metric (root_mean_squared_error)
- Time_limit (600)
- Presets (best_quality)

The neural network, however, has a much more flexible structure. Since most research on deep learning applications for stock prediction focuses on LSTM, and CNN Time Series techniques, which target only one stock specifically, we could not apply those well-trained neural network structures for transfer learning. Instead, we have to build our own neural networks in this project.

The following parameters can be tuned to optimize the neural network:

- Network Structure:

- ➢ Number of Layers (the models which will be built in this project will only contain <5 layers for simplicity)
- ➢ Activation function (we will use Tanh() for the activation since it returns values from -1 to 1, which could perfectly model the stock return value)
- ➢ Layers parameters

❖ Training parameters

- ➢ Optimizer
- ➢ Learning rate (how fast the parameter will learn from the gradient for each loop)
- ➢ Batch size (how many rows of data to train once during a single training step)

Benchmark Model

$$Q_{j,t} = \sum_{i=1}^{k} P_{i,t} * F_{j,i,t-1} + u_{j,t} \qquad (1)$$

K = the number of independent variables
P = regression coefficient of independent variable I in month t
$F_{j,\,i,\,t-1}$ = independent variable I for stock j at the end of previous period (month t-1).
$Uj,_{t}$ = error terms for each regression
Qj, t = price of (dependent variable) stock j in month t

There exist multiple approaches to predicting stock returns. One of the most well-known models is CAPM, which assumes a linear model between the stock premium returns and market returns. In this project, we will adopt the idea of linear modeling and build a simple regression model between features and target variables, and **we will consider the linear regression model as our benchmark model**. In fact, linear regression has the best interpretability, and it's widely used in financial institutions and research. We will build separate linear regression models for daily return predictor and intra-day return predictor, and compare both the machine learning model and neural networks with our linear regression model.

Methodology

Data Preprocessing

There are several steps to prepare the data before we load our data into the model, and it consists of the following steps:

- Drop Feature_1 and Feature_10 since <50% of data in these two columns are NaN values
- Use df.sample() method to split the dataset into training data and validation data
- Fill out NaN values with df.fillna() method: the main reason to fill Nan values is our implementation of deep learning models since in the process of calculating the gradient of each loop the NaN values will lead to null or infinite gradient, and then lead to failure of learning.

Implementation

The implementation of two predictors could be divided into two main stages:

- ❖ Build daily return predictor (target variables: Ret_PlusTwo, and Ret_PlusOne)
    - ➢ Take the linear regression model as the **Benchmark model**
    - ➢ Train the data within the AutoML framework, and get the RMSE error with the "best" machine learning model
    - ➢ Design a neural network to predict the daily return
    - ➢ Tune hyperparameters and change layers of the neural network to get a better performance
    - ➢ Compare different models and choose the best predictor (lowest RMSE) as the daily return predictor
- ❖ Build Intra-day return (minute scale) predictor (target variables: Res_121 - Res_108)
    - ➢ Take the linear regression model as the **Benchmark model**
    - ➢ Design a neural network, tune hyperparameters, and change structures
    - ➢ Compare deep learning solutions with the benchmark model

The features we use to train daily return models and to train intra-day returns are different. In this project, we will not use Ret_2 - Ret_120 in terms of minutely returns for two hours on Day D to predict the daily return on D+1 or D+2. Therefore, we will use Feature_2…Feature 9, Feature 11…Feature 25, Ret_MinusTwo, and Ret_MinusOne to build a daily return predictor. Then, we will use Feature_2 …Feature 9, Feature 11…Feature 25, Ret_MinusTwo, Ret_MinusOne, and Ret_2…Ret_120 as input features to predictor minutely return.

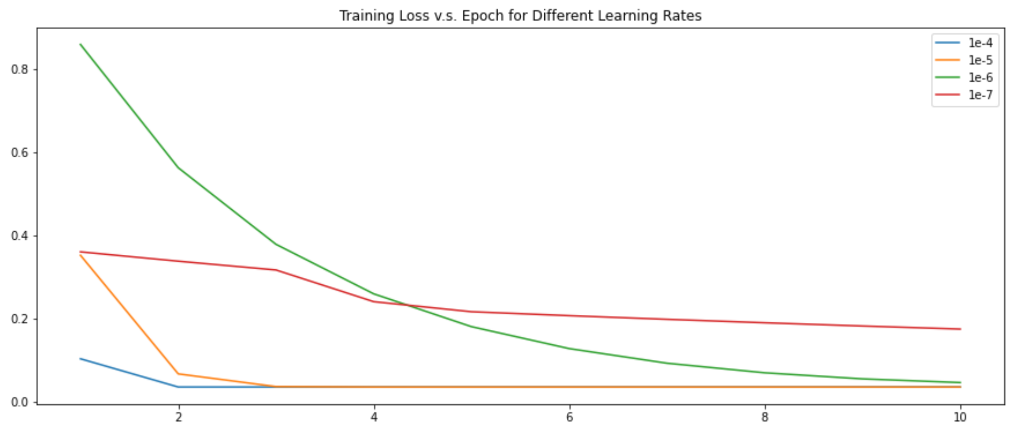
After training the AutoGluon model, we could see it chooses "WeightedEnsemble_L3" as the best machine learning model for both the Ret_PlusOne predictor and the Ret_PlusTwo predictor.

Our implementation of deep learning solutions is built within the PyTorch framework, and the process of tuning hyperparameters is mainly through changing the learning rate, batch size, the number of layers, and layer parameters.

Refinement

There is not much space for improvement on general linear regressions. The performance of the AutoGluon model could be improved by setting the time_limit larger, and the default value for this parameter is 120s. In this project, we have tried both 600s and 1200s for time_limit to predict D+1 daily return; however, the expected superior predictive performance of the 1200s-model is not seen (0.01897502 v.s. 0.01890028). Since the later model will take approximately 20 minutes to complete training, and it would occupy a great amount of memory space, we will continue to use 600s as the input parameters for the AutoGluon model to predict Ret_PlusTwo.

To tune the hyperparameters of our deep learning model, besides changing the network structure, we basically will only change the learning rate, and the plot of training RMSE errors with respect to different learning rates is presented below.

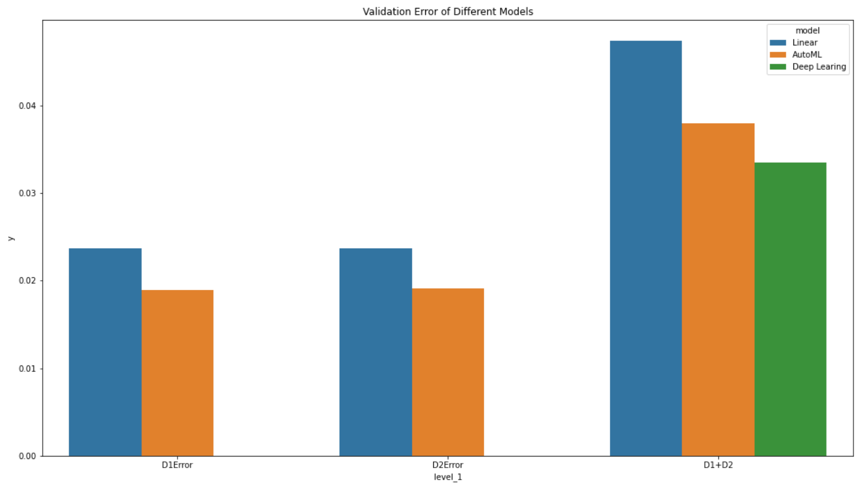


This figure helps us to tune the learning rates of our network to predict the daily return. We could see clearly through the figure above the training losses at learning rates 1e-4, 1e-5, and 1e-6 approximately converge to the same final results. When the learning rate equals 1e-4, it converges most quickly among these four choices, and it also starts with the smallest training loss at Epoch 1. Therefore, 1e-4 will be the optimal value as the input of the learning rate to model the daily return predictor.

Results

Model Evaluation and Validation

After we finish training our models (Linear Regression, Machine Learning, Deep Learning), we could compare these models with specified metrics (RMSE) on the validation set. The plot which presents the performance of each model is shown below.



The figure above shows validation errors across different models. For the neural network model, we train the model to output the target variables Ret_PlusOne and Ret_PlusTwo together; therefore, only the "D1+D2" error of the deep learning model is presented in this figure. Obviously, both the machine learning model and neural network outperform compared with our benchmark model, and the neural network has the lowest validation error across these three models. Therefore, we would use our designed neural network as our daily return predictor.

For the intra-day predictor, we implemented the benchmark model-linear regression and a neural network. After comparing the validation error (0.0090242 v.s. 0.0672745), we will apply deep learning models to predict the minute returns. Also, the small errors of both daily return and minute return predictors on validation sets show that our model is robust enough.

Therefore, the final predictor models we will apply are

Daily Return Predictor	Intraday Minute Return Predictor

❖ Structure:
torch.Size([16, 25])
torch.Size([16])
torch.Size([4, 16])
torch.Size([4])
torch.Size([2, 4])
torch.Size([2])
❖ Input: Feature_2…Feature 9, Feature 11…Feature 25, Ret_MinusTwo, and Ret_MinusOne (25 features)
❖ Batch size = 128
❖ Learning rate = 0.0001
❖ Optimizer = torch.optim.SGD
❖ Criterion = RMSE
❖ Output: Ret_PlusTwo, and Ret_PlusOne
	❖ Structure:
	torch.Size([256, 144])
	torch.Size([256])
	torch.Size([128, 256])
	torch.Size([128])
	torch.Size([64, 128])
	torch.Size([64])
	torch.Size([60, 64])
	torch.Size([60])
	❖ Input: Feature_2…Feature 9, Feature 11…Feature 25, Ret_MinusTwo, Ret_MinusOne, and Ret_2…Ret_120 (144 features)
	❖ Batch size = 128
	❖ Learning rate = 0.00001
	❖ Optimizer = torch.optim.SGD
	❖ Criterion = RMSE
	❖ Output: Ret_121 - Ret_180

Justification

Since we could see both neural networks and machine learning solutions are much better than the linear regression-benchmark models, we could claim our models are significant enough to solve the stock predictor problem. Also, I submitted my final solution to Kaggle, and get a score of "1756.44774," which definitely falls within the normal range among all the submissions.

References

1. Mansouri, Ali, et al. “A Comparison of Artificial Neural Network Model and Logistics Regression in Prediction of Companies Bankruptcy (a Case Study of Tehran Stock Exchange).” *SSRN Electronic Journal*, 2016, https://doi.org/10.2139/ssrn.2787308.