Final Project Report

Yuan Chen

https://github.com/yuanchenGH/ML_MarketPrediction

Introduction

Equities, or stocks, are the most popular asset class that attracts numerous individual and wholesale investors. An enduring topic is how to accurately predict the movement of the stock market. Although the market fluctuation seems random most of the time, it is highly correlated to macroeconomic conditions. Therefore, most macro trading strategies are based on the various combination of some indicators.

In this project, I use three macroeconomic indicators base on interest rates, key factors that have a significant impact on the economy - default spread, term structure spread, and the risk-free rate, to predict the stock market return.[1]

Default spread is defined as the yield gap between AAA and BBB grade bonds. AAA bonds are issued by the companies with a great outlook, so it has the lowest risk thus lowest yield. And BBB bonds are those with a higher risk of default, yet still, have investment value. When the economy is at stake, investors seek a higher and higher return to pay off the increasing risk.

The term structure spread is the yield curve of bonds, which demonstrates the interest rates of bonds with the same quality at different maturities. The slope of the curve is a crucial pointer of the current state of the economy. An upward sloping means the long-term yields are higher than the short-term ones point to an expanding economy and vice versa. A flat cure means the economy is stalling or the market is unsure of the future.

A risk-free rate is considered as the baseline of investment since they will only invest if the return is greater than the risk-free rate. The three-month U.S. Treasury bill is usually used for the calculation for the fact that it has the lowest default risk.

In this project, three training methods are deployed to investigate the correlation, namely, LASSO, Ridge Regression, and Neural Network.

Data collection

Raw data were downloaded from databases listed at the end of this report.[2] including yield history of AAA (labeled as DAAA) and BBB (labeled as DBAA) grade bonds, yield history of treasury 10 years (labeled as DGS10), 1 year (labeled as DGS1), and 3 months (labeled as DGS3MO) bonds, return history of Dow Jones, SP500 and Nasdaq index.

I choose 20 years' worth of data between Jan. 4, 1999, to Dec. 31, 2019, and obtain 5244 data points. Training data with Risk-Free Rate (rfr), default spread (ds), and term spread (ts) is calculated as below:

rfr = DGS3MO

ds = DBAA – DAAA

ts = DGS10 – DGS1

Each data point is consisted of [rfr, ds, ts, 1], resulting in a matrix of 5244*4 dimension, marked as A. The labels are the direction change of the market index, so if the market return of a given day is positive, the label is 1, and -1 otherwise. Three sets of 5244*1 labels are stored in the B matrix of 5244*3 dimensions. The last data in A and the first label in B were removed to reflect the fact that we are using the previous day's data to predict the second day's market movement. So the final data is 5243*4 and 5243*3 respectively.

Several data points are missing. Since it's unlikely that the bond rate or stock market will change dramatically in a short period, they are filled with the average value of the previous and next data. The columns of data were merged using the Pandas by date.

LASSO and Ridge Regression Training

The preliminary training using LASSO and Ridge Regression is shown below. A 10 fold cross-validation is used to optimized λ. The data is divided into 10 groups, in each case, 8 groups are used to train the classifier, two groups are left as hold out, one is used to test λand choose the best value by the error rate. The resulting classifier is evaluated by the second hold out data. Squared error and error rate was calculated at last using the evaluation (Table 1, see Final Project Iteration 1.ipynb for more detail). The prediction is not much better than just guessing the direction of market movement. Each classifier gives an approximate squared error of 0.06, and an error rate ranging from 0.463 to 0.479. The Ridge Regression prediction of the Nasdaq index provides the best result with an average error rate of 0.463 and the LASSO prediction of SP500 is the worst with an average error rate of 0.479.

	LASSO		Ridge Regression	
	Squared Error	Error Rate	Squared Error	Error Rate
DJIA	0.0601	0.474	0.0601	0.474
SP500	0.0604	0.479	0.0599	0.470
Nasdaq	0.0600	0.472	0.0594	0.463

Table 1. Square error and error rate corresponding to each algorithm and stock market index

Due to the disappointing result from the last iteration, I tried different approaches by preprocessing the data. First, I compared the gradient descent(GS) and stochastic gradient descent(SGD), as expected the error rate has almost no change from the last iteration (see Final Project Stochastic Gradient Descent.ipynb for more detail).

After reviewing the data and definition of the features, I conclude that the trend of the macroeconomy indicators changes may be more important than the absolute value themselves, so I defined three new features and added them to the dataset:

rfr2, ds2, and ts2 which are defined as the day-to-day change of the rfr, ds, and ts value.

The inclusion of these features only slightly improves the error rate of the LASSO but does improve the error rate of the Ridge Regression by a non-neglectable amount, to a range of 0.454 to 0.468. (Table 2, see Final Project Added Trend.ipynb for more detail)

	LASSO		Ridge Regression	
	Squared Error	Error Rate	Squared Error	Error Rate
DJIA	0.0600	0.472	0.0591	0.459
SP500	0.0604	0.479	0.0596	0.468
Nasdaq	0.0593	0.461	0.0588	0.454

Table 2. Square error and error rate with new features added

On top of the new features, I tried to normalize the data and it showed the opposite effect on the error rate. The LASSO results improve significantly to a range of 0.431 to 0.443 but negate the improvement brought by the new features to Ridge Regression. (table 3, see Final Project Normalized.ipynb for more detail)

	LASSO		Ridge Regression	
	Squared Error	Error Rate	Squared Error	Error Rate
DJIA	0.0572	0.431	0.0600	0.472
SP500	0.0580	0.443	0.0600	0.472
Nasdaq	0.0576	0.437	0.0594	0.463

Table 3. Square error and error rate with normalized data

In the sparse solution obtained from the last two trails, it appears some features have a weight of 0, which means they may have little influence on the label. To test the impact of these features, I test different combinations of the six features. The combination of rfr2, ds2 win the contest and provides more than 1% improvement. The LASSO algorithm end with an error rate of 0.420 to 0.428. And the Ridge Regression with an error rate of 0.454 to 0.459. (table 4, see Final Project Reduced.ipynb for more details).

	LASSO		Ridge Regression	
	Squared Error	Error Rate	Squared Error	Error Rate
DJIA	0.0565	0.420	0.0587	0.454
SP500	0.0570	0.428	0.0591	0.459
Nasdaq	0.0567	0.422	0.0590	0.456

Table 4. Square error and error rate with reduced features

Further investigation of the feature interaction (Final Project Data Interaction.ipynb) offered no significant improvement over the current algorithm.

Neural Network Analysis

I choose Pytorch since it's one of the most mature neural network training libraries, and more importantly, it has the ability to utilize a graphic card to significantly reduce the training time (Final Project Neural Network2GPU.ipynb). The user can use a local graphic card or a commercial cloud card to speed up the computation. Without it, each part of the compution may take hours to finish.

The data set used in this chapter is the expanded one with A = [rfr, ds, ts, rfr2, ds2, ts2, 1] as a 5243*7 matrix. This time I try something different for the cross-validation, instead of choosing a fixed 10 set of train/evaluation data, I randomly hold out 10% of the whole data, repeat the training and evaluation process 10 times and calculate the average prediction accuracy to judge the result.

The first step to start a Neural Network training is the determination of the number of hidden layers and nodes. As a general rule, I start with the same amount of nodes as the number of features(7) and test out how increasing and decreasing the number of nodes affect accuracy. I start the training with 3 hidden layers due to the project requirement. Each hidden layer uses a ReLu activation function and a Mean Square Error loss function (MSELoss from Pytorch) for backpropagation. The result is shown in table 4 below, which also shows how many epochs are needed to complete the training.

Nodes	Market	Epochs					
		1000	2000	4000	6000	8000	10000
4	DJIA	0.459	0.444	0.426	0.419	0.412	0.433
	SP500	0.466	0.447	0.425	0.419	0.421	0.448
	NASDAQ	0.449	0.437	0.427	0.412	0.402	0.448
7	DJIA	0.455	0.434	0.412	0.414	0.413	0.422
	SP500	0.46	0.41	0.42	0.414	0.425	0.436
	NASDAQ	0.446	0.406	0.419	0.408	0.402	0.446
10	DJIA	0.455	0.435	0.417	0.408	0.424	0.422
	SP500	0.454	0.44	0.421	0.41	0.43	0.439
	NASDAQ	0.448	0.431	0.52	0.417	0.412	0.444
20	DJIA	0.449	0.429	0.427	0.417	0.413	0.422
	SP500	0.454	0.436	0.431	0.412	0.426	0.443
	NASDAQ	0.448	0.433	0.428	0.422	0.408	0.445

Table 4. Accuracy result of different number of nodes and epochs

Table 4 shows that the number of nodes does not have a meaningful impact on the accuracy rating, the second set of results exhibits a slightly higher overall accuracy so the general rule holds well and the 7 nodes will be used for the rest of the training. On the other hand, the accuracy reaches a peak at 6000 epochs, which will be used for the rest of the training. The jump after 8000 epochs is probably due to the non-convex nature of this training problem. Table 5 shows the impact of hidden layers. In this case, it seems to have no effect, which means one hidden layer is sufficient for the problem.

	1 layer	2 layer	3 layer	4 layer
DJIA	0.41	0.413	0.414	0.414
SP500	0.41	0.41	0.414	0.409
NASDAQ	0.403	0.406	0.408	0.41

Table 4. Accuracy by hidden layers (7 nodes, 6000 epochs)

After the number of nodes, hidden layers and epochs are chosen. I dig further into other parameters, for instance, learning rate and loss function. Table 5 and 6 show the result of the result.

Learning Rate	0.0001	0.001	0.01	0.1	1
DJIA	0.407	0.41	0.411	0.416	0.451
SP500	0.412	0.41	0.415	0.413	0.438
NASDAQ	0.415	0.403	0.405	0.409	0.438

Table 5. Accuracy by learning rate (7 nodes, 6000 epochs, 1 hidden layer)

In table 5, the smaller learning rates up to 0.1 have a negligible impact on the training process and result in similar accuracy. When the learning rate becomes too big, it negatively impacts because it's harder to reach the global minimum of the weights. The choice of loss function does contribute to the accuracy greatly. Here I only used the loss function provided by Pytorch, the attempt to write my own loss function only partially successful, but computes very slow so that I don't have enough time to finish the training. For the loss functions I tried in this sector, the L1 loss function provides the best accuracy of ~60%, although it's quite close to the mean squared error loss.

	MSELoss	L1Loss	HingeEmbeddingLoss	MarginRankingLoss
DJIA	0.41	0.397	0.558	0.416
SP500	0.41	0.401	0.56	0.431
NASDAQ	0.403	0.406	0.562	0.427

Table 6. Accuracy by loss function (7 nodes, 6000 epochs, 1 hidden layer)

Conclusion

There is quite some take away from this project. The most important one is that data preprocessing is important. Normalization by far contributes the biggest improvement to the accuracy. It has two major effects, one is the increased stability of the data set. Irregular changes in the data lead to irregular jumps in the gradient descent, and cause oscillations and fail to converge. It may also speed up the training by avoiding a lot of overshoot.

It's likely that some of the macroeconomic factors have greater effects, so that the LASSO algorithm, which gives a sparse solution, provides better accuracy in the two linear classification methods. And after removing some of the factors, it reached an even better prediction rate.

Neural Network, as expected, gives the best overall result. From this sector of the project, I learned that choosing good parameters and functions are important to balance accuracy and computation cost. There is still a lot to learn about the underlying theories and principles to fully understand the reasoning of these choices.

Many believed the stock market is random by nature and could be impossible to predict. The near 60% accuracy obtained by this Neural Network is quite significant considering that the model is not very sophisticated and there are still many factors that can be investigated. However, for anyone who wants to use the model to profit from the market, keep in mind that this is trained on a limited amount of historic data, and there is no guarantee that it will correctly predict the future market.

References:

1. (a)Campbell, J. (1987). Stock returns and the term structure. Journal of Financial Economics, 18, 373–399.

(b)Cochrane, J., & Piazzesi, M. (2005). Bond Risk Premia. American Economic Review, 95, 138–160.

2.
10-Year Treasury Constant Maturity Rate (DGS10):
https://fred.stlouisfed.org/series/DGS10

1-Year Treasury Constant Maturity Rate (DGS1)
https://fred.stlouisfed.org/series/DGS1

3-Month Treasury Constant Maturity Rate (DGS3MO)
https://fred.stlouisfed.org/series/DGS3MO

Moody's Seasoned Aaa Corporate Bond Yield (DAAA)
https://fred.stlouisfed.org/series/DAAA

Moody's Seasoned Baa Corporate Bond Yield (DBAA)
https://fred.stlouisfed.org/series/DBAA

DJIA
https://finance.yahoo.com/quote/%5EDJI/history?p=%5EDJI

NASDAQ&SP500
Wharton Research Data Services (WRDS): The Center for Research in Security Prices (CRSP): Daily Stock File
https://wrds-www.wharton.upenn.edu/