A Comparative Study of Parallel Computing on Stock Price Prediction Models

Course: BIA 678 Big Data Technology
Professor: David Belanger
Team 3 members:
Tianrui Wang
Xing Fang
Ziyu Ma
Yuning Yang

1. Abstract

Our project is using a stock price prediction dataset to run on various designed AWS clusters and local machines. The purpose of this study is to testify the scalability and time-efficiency on parallized cloud servers and local servers among different stock price prediction models. The value of this study is to give insights to the stock market movement and five different algorithms' implementations through Amazon Elastic MapReduce (EMR) with two nodes, four nodes and six nodes, and local computers. The work is presented in the following manner: we started Section 3 with data description; a brief exploration of the data is presented in Section 4; in section 5 we talked about the data processing pipeline used in the project; in section 6 we adopted five different kinds of stock prediction classification models to examine the effect of scale on model performance and time complexity using parallel computing; model comparison is presented in section 7. In the end, a comprehensive conclusion and recommendation are given in section 8.

2. Introduction

Stock price prediction has been an ongoing topic both within the industry and academia. At each given point in time, the market is affected by various factors with different levels of significance. Financial data from different sources has been used in predicting the stock market: using either the historical data or data from related news alone would be insufficient for a market with such volume and velocity. Given our access to the enormous amount of data on the daily stock market and the news articles, we explored the power of parallel computing and AWS services by conducting a regression analysis using five models: Linear Regression, Generalized Linear Regression, Random Forest, Gradient Boosting Tree, and XGBoost. The main target is to derive the effect of scale on model performance and execution time, as well as the effect of parallelism on execution time.

3. Data Description

The data we used is retrieved from a Kaggle Competition held by Two Sigma. It can be broken down into two parts. The first part is the stock market data from Intrinio. This dataset includes the target variable, which is the 10-day stock returns calculated on open prices and adjusted for market-residuals. Others include stock price information, such as trading volume, open and closing prices, 1-day stock returns and 10-day stock returns. The second is the News data from Thomson Reuters, including the headline of the news; the relevancy of the news piece; sentiment scores; information on the article, e.g. word count; and the time effect of news, counted by novelty trading volume over different time periods. We joined the 2 tables on date and assetCode, to make sure that the news data we are using is specifically related to that stock on that day.

4. Exploratory Data Analysis

The time frame of the data is from 2007-01-01 to 2016-12-31. We used a total of 3562 assets to form our dataset. Figure X shows the closing price of 10 randomly chosen assets over the ten-year-period. It could be observed that some assets just stopped trading after 2007, like Palomar Medical Technologies, Inc. Some assets started trading later than 2007, like Power Integrations, Inc.

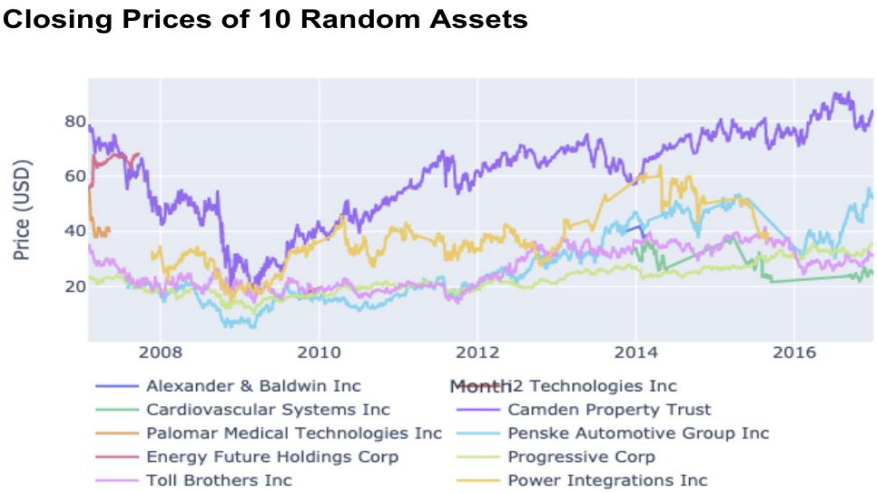

Figure 1: Closing Prices of 10 Random Assets

We also checked the closing price by quantiles, to see how the market moved over time. Figure 2 shows that the stock prices at a 0.95 quantile have increased over the 10 years, and the stock prices at a 0.05 quantile have decreased.

Figure 2: Trends of Closing Prices by Quantiles

5. Data Processing Pipeline

For data processing, we built a pipeline using Pyspark. We first loaded the data and then took all the numerical features, to a vectorAssembler. And then we scaled the vector through a

standardScaler provided by the Pyspark Framework.

There is also one categorical feature, the AssetCode label, used to distinguish different stocks. we took it through a stringIndexer and did the dense vector one-hot-encoding to solve both the word embedding and sparse matrix problem. Then we assembled all the features and got the transformed data.

We fit the pipeline on training and transform the training and the test data.

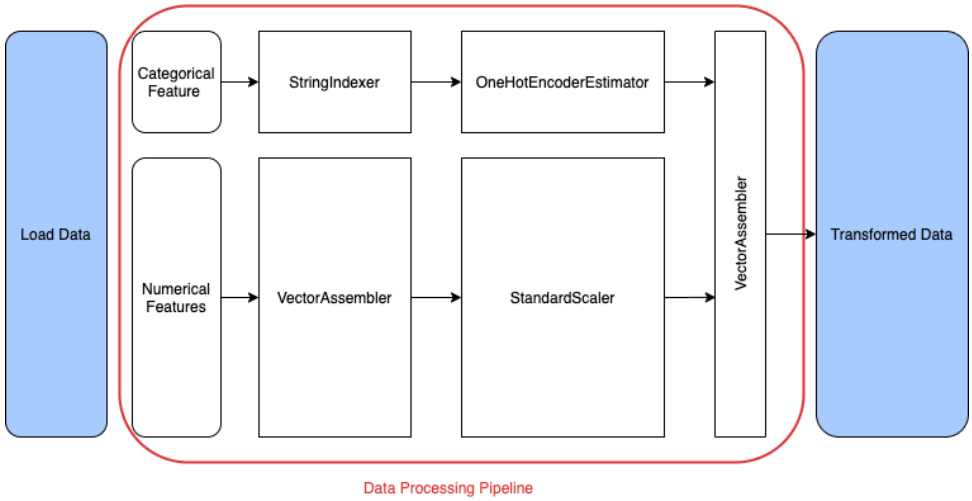

Figure 3: Data Processing Pipeline

6. Applied Models and Results

We used five models to train the data at 20%, 40%, 60%, 80%, and 100% scaling. All the experiments are done using Pyspark Framework. We held 80% for training and the rest for testing for all of the models. Training is run on local machines, as well as AWS EMR clusters, with 2, 4, 6 nodes. We used Root Mean-Squared Error (RMSE) to calculate the error of our models. Time is counted from the beginning of the data processing pipeline, until the end of the calculation of test errors.

Linear Regression (LR)

After processing the data and based on the data structure, we tried to run linear regression firstly. As we all know linear regression is a linear approach to modeling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). In this step, we used the whole dataset with 2.6 Gigabytes. And we applied paramgridbuilder to test different parameters. After that we utilized cross validation to tune the parameters.

The root mean-squared-error (RMSE) results comparison between AWS EMR clusters with different nodes and local shows that with 20% or 40% data used, the model achieves lower RMSE. And models run on 2 nodes and 4 nodes almost have the same result.

After comparing scale versus time with different levels of parallelism, the graph shows that with the increasing of nodes, time needed for model fitting decreases at the same time; percentage of data used does not have a significant influence on time using; At last, because the data size is too large, we can just run at most 40% data on local. Even though, from the graph we can also see that the local take longer time than AWS which agrees with our common sense.

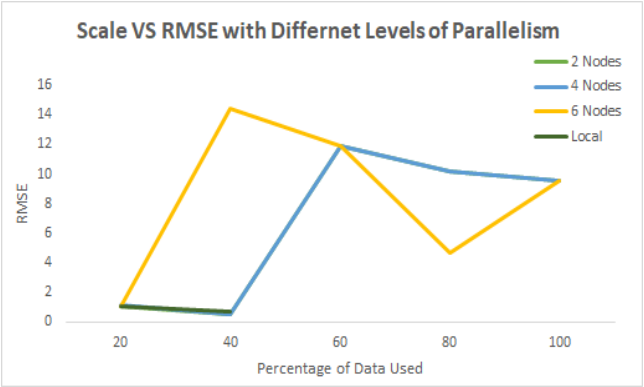

Figure 4: Scale VS RMSE with Different Levels of Parallelism (LR)

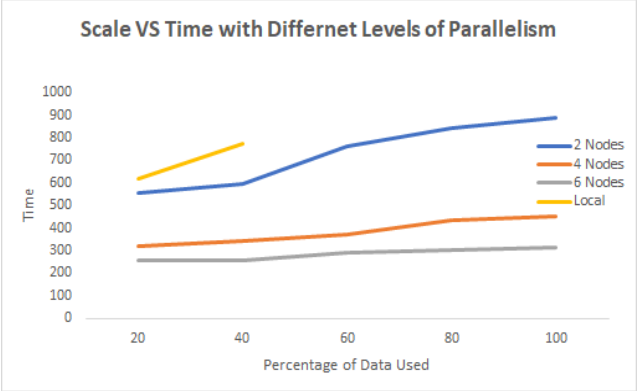

Figure 5: Scale VS Time with Different Levels of Parallelism (LR)

Generalized Linear Regression (GLR)

Given the result of LR, we decided to move on with only 50% of the data, which is 1.3 Gigabytes. The second model we used is a GLR with gaussian family and other parameters setted in Table 1.

link	"identity"
maxIter	10
regParam	0.3

Table 1: Parameter Settings for GLR

GLR requires the assumption of the output to follow a Gaussian distribution. While this might not be exactly the case, we think that most stock returns tend to be log-normally distributed. For example, the density plot for Apple, Inc. (code: AAPL) uses data from our dataset, and it follows a bell-shape. The result turns out to be promising, achieved an root-mean squared error (RMSE) of 0.06.

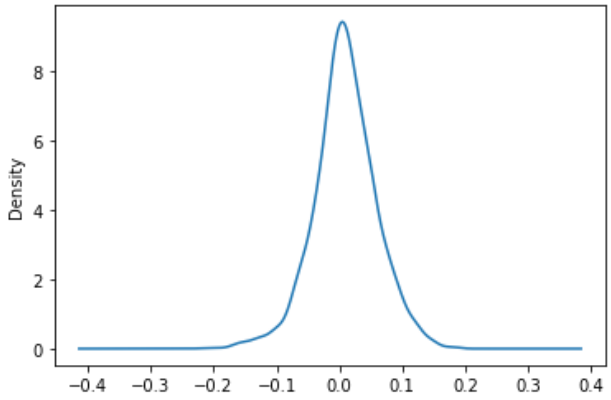

Figure 6: Density Plot for Apple, Inc (AAPL)

We conducted experiments on both local, and AWS EMR clusters, with 2, 4, 6 nodes. Figure 7 shows that the local line is collapsed with the 2-nodes line. This is similar to the situation when applying LR. As one could observe from Figure 7, with the increase of the Scale, the performance of the model is not very stable. The best performance is reached using 20% of data. This result is quite surprising, but it is consistent with the results of LR. After 20%, it is safe to say that the performance is increasing with more data used for training. This aligns with our assumption.

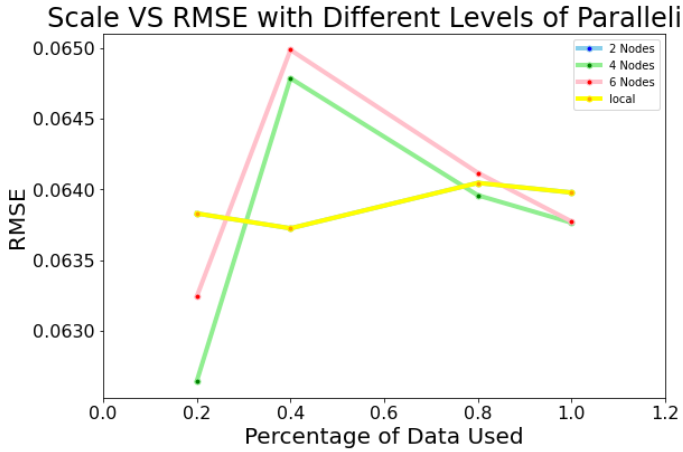

Figure 7: Scale VS RMSE with Different Levels of Parallelism (GLR)

Talking about the effect of scale on Time, we could see that the 6-nodes experiment is achieving the shortest time. The advantage of parallelism did not reveal until going up to 6 nodes on EMR clusters. It is also worth mentioning that for local training, we used Pyspark. Thus, the local

machine can utilize the advantage of parallel programming with a 4-core CPU. This might cause the local result to be similar to a 4-node training on the EMR cluster.

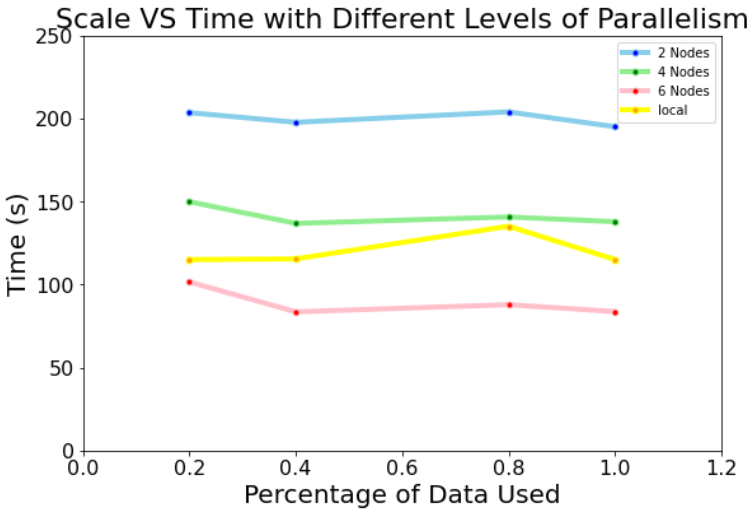

Figure 8: Scale VS Time with Different Levels of Parallelism (GLR)

Ensemble Models: Bagging and Boosting

Ensemble models are designed to provide the classifier with lower variance and higher stability. For the bagging model, each iteration in the training process includes bootstrap sampling on the data, and each batch of data will be used to train a classifier. For the boosting model, all training data is used while several weak classifiers are trained. Weight on each data point will be updated according to the result of the previous classifier.

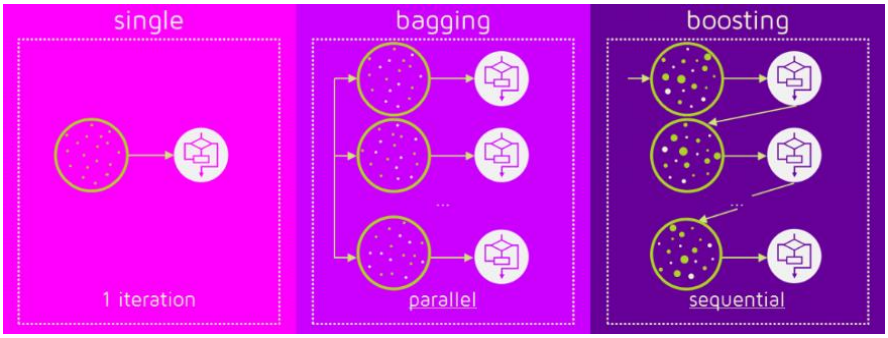

Figure 9: First Stage of Single / Bagging / Boosting

After classifiers are trained, the second process, which shows below, does the work to concatenate all pre-trained classifiers together using weighted average or major vote. From the principles and structures these two kinds of models have, we can see how parallelism fits for the training process

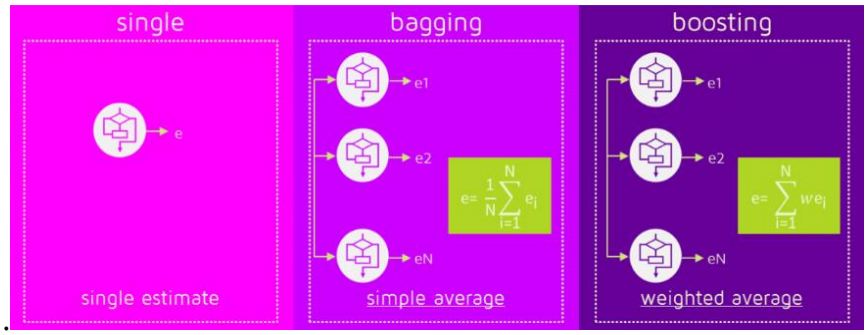

Figure 10: Second Stage of Single / Bagging / Boosting

Random Forest (RF)

We selected random forest as an example of bagging models. As same as the GLR model, we used the 1.3G data which is nominally half of the original dataset. In order to avoid the overfitting problem, we set the number of trees to be 5 and maxdepth of each tree to be 4.

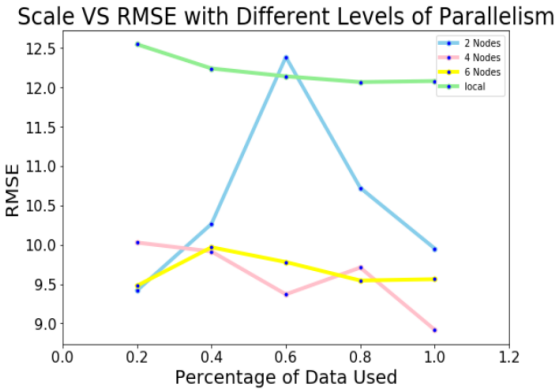

Figure 11: Scale VS RMSE with Different Levels of Parallelism (RF)

From the results, the local machine performed the worst, and the cluster of 4 nodes has the best performance on Rooted Mean Square Error. And in general, using 20% of the data to train the

model is a good choice. Besides, there is a positive relationship between numbers of nodes and the performance. Meanwhile, a fluctuation was observed when using 60% of the dataset to train the model, this may because our dataset includes AssetCode as one of the attributes, the unbalance problem is introduced to the dataset, unlike regression models, tree models may derive certain branches especially for AssetCode, this uncertainty results in the fluctuation problem.

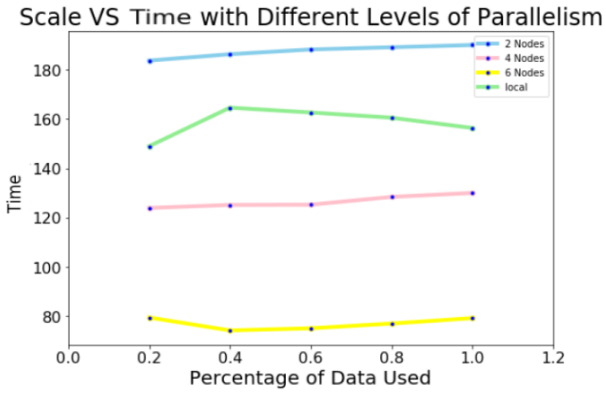

Figure 12: Scale VS Time with Different Levels of Parallelism (RF)

For the running time, it's quite similar to the previous two models, the running time does not increase too much when using more data to train the model, and it makes sense that running time decrease with number of nodes increase, but local machine performed better than cluster with 2 nodes, and we can see the running time decrease exponentially when scaling number of nodes from 4 to 6

Gradient Boosting Tree (GBT)

For Boosting models, we tried Gradient Boosting Tree first.

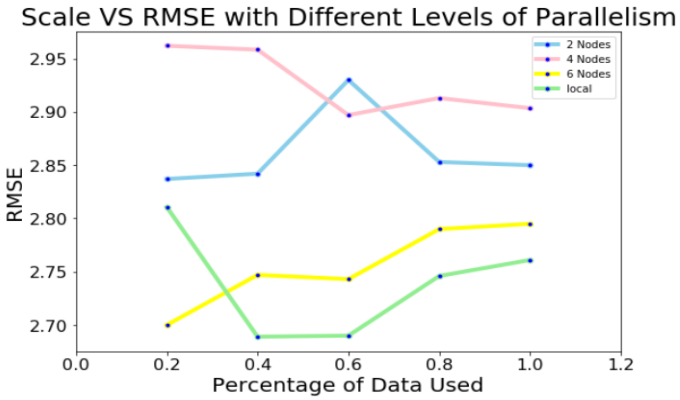

Figure 13:Scale VS RMSE with Different Levels of Parallelism (GBT)

With the batch size scaling up, the different clusters performed with huge differences, most of them reached the best result when using 20% data. Meanwhile, The models also showed unexpected fluctuation when using 60% of the data and abnormally we found our Local machine has the best performance when training size increases, the parallelism only shows better performance on smaller dataset. This problem may be due to the parameter setting. We think the performance issue here may be due to the parameters we set, because I only set my models as baseline models, the depth is shallow, the advantage of parallelism may not show off, but when comparing different clusters, it's true that with more clusters, the performance can be better.

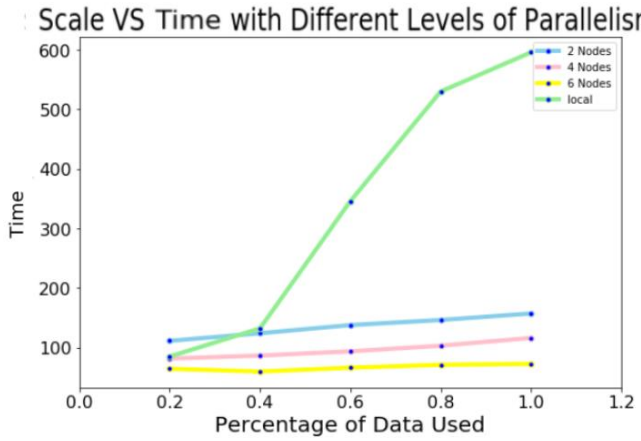

Figure 14:Scale VS Time with Different Levels of Parallelism (GBT)

Regarding the running time, the running time of the local machine increase exponentially when scaling up the batch size but EMR does not have this problem, comparing to the random forest, the boosting algorithm does not use bootstrap, it used different weight on different data, so here the parallelism shows great advantages.

XGBoost

In order to achieve better comparison, we also tried XGBoost as another boosting model.

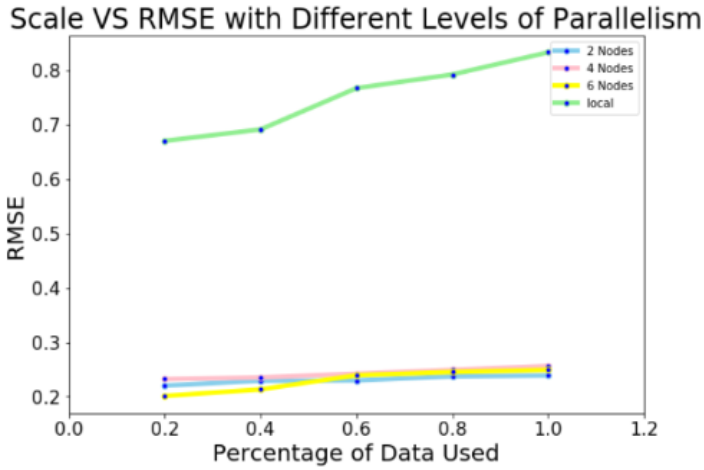

Figure 15:Scale VS RMSE with Different Levels of Parallelism (XGBoost)

The local machine shows relatively high RMSE. This may due to the linear regressor can perform better when using different computational cores. Most clusters performed the best when using 20% data to train. Cluster with 6 nodes has the best performance among clusters. And with the size of data increases, the cluster with 6 nodes performs the worst.

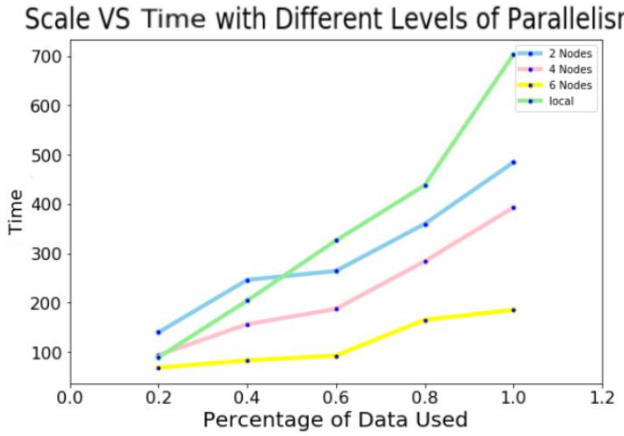

Figure 16:Scale VS Time with Different Levels of Parallelism (XGBoost)

For running time, the local machine also increased exponentially when scaling up the batch size, this is similar to Gradient Boosting Tree model, the parallelism here shows great advantages

7. The Comparison between Models

We compared different models based on 20% data, because based on previous research, we get a conclusion that with 20% data used, most models have the best performance.

From the figure 17 which shows the relationship between number of nodes and RMSE based on different models, we can see that using AWS does not have a significant influence on model performance. And the most efficient factor is model selection. So in our data analysis process, selecting a model is a very crucial step. As we saw from the graph, generalized linear regression contributed to the smallest RMSE, meaning the best model-fitting. However, random forest performed poorly from the perspective of RMSE, model-fitting feature.

From the figure 18 which shows the relationship between number of nodes and time based on different models, we can see that the number of nodes has a positive relationship with model fitting time; linear regression takes most time among all models; for different models, local takes almost the same time as 2 nodes.

Based on model comparison, we can come to the conclusions that GLR has the best performance and GBT takes the shortest time in model fitting. Because different models have different calculation steps, their performances vary in time and accuracy aspects. Based on the comparison results, we recommend that GLR, GBT, XGBoost are excellent models from the standpoint of model efficiency in stock prediction.

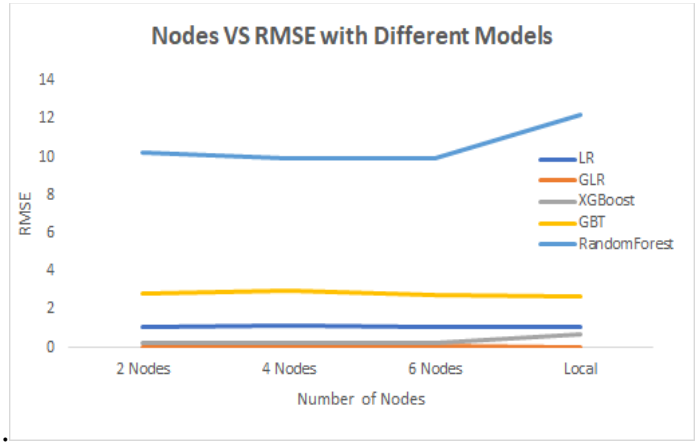

Figure 17: Nodes VS RMSE with Different Models (20% Data)

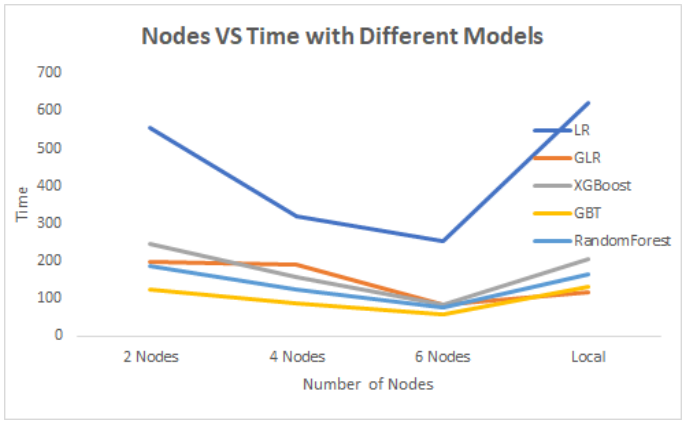

Figure 18: Nodes VS Time with Different Models (20% Data)

8. Conclusion and Future Study

In this project, we manipulated five algorithms, consisting of Linear Regression, Generalized Linear Regression, Random Forest, Gradient Boosting Tree, XGBoost. With the five models we conducted, we discovered three major outcomes of our entire project.

Among the five classification models running, the best performance achieved mostly when we used only 20% of Data. Our dataset contains 3562 assets with an average of 1619 prices per asset. However, some major assets with comprehensive information could have up to 61,786 data points, spending over the 10-years period. One may question the possibility of having over 3652 data points within the 10-year period. This is possible because, for such assets, thousands of news may come up during a "busy" day. The less data we use, the less asset classes we may include in our training, and the more focus the dataset will be for several particular assets. Reversely, more data meaning a more diversified "portfolio", and that could largely increase data complexity for our regression task.

Secondly, we found out that the plots of "Scale VS Time" tend to be flattened when using linear regression models. However, it is not sufficient to conclude that the use of clusters could not improve processing time. Our assumption is that the size of our dataset is too small to reveal the scaling effect on processing time.

Last but not least, our project implemented five different methodologies for stock price prediction modeling. According to the pragmatic application and result, we realized that Generalized Linear Regression contributed to the best model fitting with the smallest RMSE comparing other methods. Random Forest ended up with the worst performance from the perspective of model fitting. In Figure 18, time versus parallel computing graph, we could conclude that the Gradient Boosting Tree method outperforms all other methods with the shortest execution time. Second comes the Random Forest method. Generalized Linear Regression and XGBoosting are fairly efficient. The most inefficient method is actually Linear Regression. From the standpoint of parallel computing, all methods perform better and better as the number of nodes increases. Overall, AWS does provide a better and more efficient performance than local computers.

To our surprise, experiments run on local machines could compete with some of the experiments done on clusters, such as the 2-core clusters and even the 4-core clusters. It is believed that the process of large datasets can be optimized through the distributed framework of Apache Spark using Pyspark API on local machines. Running Spark locally with K worker threads, where K equals the number of cores on the local machine, could fully utilize the computational power of a multi-core local machine.

REFERENCES

[1] Hagenau, M., Liebmann, M., & Neumann, D. (2013). Automated news reading: Stock price prediction based on financial news using context-capturing features. *Decision Support Systems*, 55(3), 685-697.

[2] Jiang, W. (2020). Applications of deep learning in stock market prediction: recent progress. arXiv preprint arXiv:2003.01859.

[3] Kaya, M. & Karsligil, M.. (2010). Stock Price Prediction Using Financial News Articles. 478 - 482. 10.1109/ICIFE.2010.5609404.

[4] Mohan, S., Mullapudi, S., Sammeta, S., Vijayvergia, P., & Anastasiu, D. C. (2019, April). Stock Price Prediction Using News Sentiment Analysis. In *2019 IEEE Fifth International Conference on Big Data Computing Service and Applications (BigDataService)* (pp. 205-208). IEEE.

[5] Oncharoen, P., & Vateekul, P. (2018). Deep learning for stock market prediction using event embedding and technical indicators. In 2018 5th Interna-tional Conference on Advanced Informatics: Concept Theory and Applications (ICAICTA) (pp. 19–24). doi:10.1109/ICAICTA.2018.8541310.

[6] Rundo, F., Trenta, F., di Stallo, A. L., & Battiato, S. (2019). Machine learning for quantitative finance applications: A survey. Applied Sciences, 9 , 5574.

[7] R. P. Schumaker and H. Chen, “Textual analysis of stock market prediction using breaking financial news: The azfin text system,” ACM Trans. Inf. Syst., vol. 27, no. 2, pp. 12:1–12:19, Mar. 2009. [Online].