Railway

——the report of Introduction to business computing

Team project : Real-time bond pricing-railway

Team members : 吉迎 吴琼琳 王蕊 盛竞逸

Report of team project

➢ Description

Our program named “Real-time bond pricing-Railway” is aimed to predict the next price a bond might be traded at, given information on the bond including current coupon, time to maturity and a reference price.

➢ Background

As far as price transparency is concerned, there is historically a huge gap between the amount of reference information available to those trading equities versus those trading corporate bonds.

Stock exchanges report trades, bids and offers all the time on the Internet. Free access is available online with a 15 minute delay. And traders who demand more information can pay for real time data and information about size of current bids and offers.

By contrast, bond trades are required to be reported within 15 minutes and only those who pay for the TRACE feed can access this information. No quotes are publicly available. Alternatively there are data companies provide end of day prices, published after the market has closed and with no guarantee that the specific information sought will be included. Accurate bond pricing is also hindered by lack of liquidity.

“Real-time bond pricing-Railway” is just the provider of real-time corporate bond prices. We can provide accurate prices of bonds by incorporating interest rate data, trades or quotes of bond in question and other bonds as well as other input sources. Pricing bonds accurately requires an exacting knowledge of payment schedules, trading calendars and reference data for each bond. This requirement of real-time information of so many bonds is what we feel challenging throughout our project. In spite of this, we provide you with a reference price which is an intermediate result of our calculations and is labeled 'trade_price' in the dataset.

➢ The process of programming and the result

Before we started to program, we happened to find that the there are countless missing values in the data, which may cause the failure in running the program. Our data had about 750,000 lines, so it was impossible to eliminate the missing values by ourselves. To solve this problem, we rewrote several codes to remove the NaN (missing value), and saved the new data in another file called “whole_price_data.csv”.

And also, as we have mentioned above, the revised data file was still very huge (more than

250MB) to read by the computer, and it caused us long time to train the model and get the value of MSE. In this case, we added codes for randomly choosing 150,000 lines of data and saved them in a new file called “150000_price_data.csv”. Since our data was deduced from 750,000 to 150,000, someone may doubt that our data is not that precise and thus increase the errors in our prediction. But before we started to cut our data, we carefully checked our data and tried to ensure that the deduction of data won’t have too much impact on the result.

After we improved the data, we were going to find the best algorithm in training the machine and predicting the price of bonds, based on US corporate bond trade data computed by Benchmark Solutions. Here are the algorithms we want to try:
1.KNeighborsRegressor
2.LinearRegressior
3.DecisionTreeRegressor
4.RandomForestRegressor
5.AdaBoostRegressor
And we used the metric MSE (Mean Square Error) for measuring the performance of a regression model.

The following are the detailed procedures by which the python program works:

First, the python program reads the new price data using [pandas]. As we can see in the chart, the data is divided into two parts. One is thirteen features of different bonds, including bond id, weight, time to maturity, trade size, trade type, and etc. Another is the price of a corresponding bond. Then, we prepared feature matrix X and label vector y, and use holdout validation to split X and y into two parts: training (80%) and testing (20%).

As the value X and Y are prepared, it is time to train a model .We used the algorithms (KNeighborsRegressor, LinearRegressior, DecisionTreeRegressor, RandomForestRegressor, AdaBoostRegressor) in turn (Not in a loop because it will take the computer a long time to run the code).

After we trained the model, we were going to evaluate the regression model on testing part. We ran the program to compute the MSE of each algorithm for further comparing. It seemed work quite well. The following chart shows what the program has got:

	KNeighbors Regressor	Linear Regressor	DecisionTree Regressor	RandomForest Regressor	AdaBoost Regressor
MSE	80.762049	0.930923	1.372639	0.655586	10.127853

At first, we set the original n_neighbors = 30, but the MSE of it was extremely high (about 86), so we decided to use a loop to find the best n_neighbors for the KNeighborsRegressor. However, what confused us is that the MSE of KNeighborsRegressor was always higher than 80, and when n_neighbors = 200, the MSE was the smallest but also about too high (about 80). So we can draw

a conclusion that the algorithm of KNeighborsRegressor doesn't fit our case, and the RandomForestRegressor performs the best.

Since we have found the best algorithm to predict the price of the bond, now it's time to use the program and predict the bond price which we don't have in the market. However, the bond to predict data has the same problem as the training data: there are many missing values in it, which may cause the failure to run the program. So we also need to write several codes to remove the NaN (missing value), and saved the new data in another file called "bond_to_predict.csv".

After we improved the predicting data, we were about to use the program to predict the price of each bond. Our goal is to have a relationship between bond id and its trading price. So we used a loop and put all the result into a new file: "result.csv". The contents of the file only have bond id and trading price because we want the file to be brief and concise so that anyone can easily find what he want in a short period.

That's the whole process of our programming, and we have also shown our thinking process in it.

➢ Problems in programming

We have met several problems during the process of program and here I'd like to describe some important ones and how we solved them.

1. The missing value

Missing value is a huge obstacle to run the program successfully because the computer can't train a model with NaN. And since we have never met it before, we were at a loss about how to deal with it. After we ask the teacher for help, we know the new concept of NaN, and the way to solve it is very simple: we could just remove the lines with NaN and save the new data in a file. The only concept we need to use is the how to build a loop and how to open a new file.

2. Too huge file to operate

Our data is about 250MB, but the normal size of a data is about 50M. Every time we run the program, the computer will take nearly 20 minutes to figure out the result. People who use this program will easily get impatient when waiting so long. So we raise a new question: Can we deduce our data to cut the time in running it but still achieve the same result as the original one? At first, we want to deduce either the feather of bond or the number of bond. But we are afraid that if we deduce the feature of bond, the result that we come across won't be as accurate as it is originally. On the contrary, if we deduce the number of bond, almost every bond trade in the same market has the same feature and fluctuate in the same way, so the number of bond won't influence the result too much. At last, we randomly choose one-fifth of the data and make it into a new file. In this problem, we should think before we do, and the concept we need to use is about the random shuffle.

3. Application of the trained model

Actually, we learned a lot about how to evaluate the model in class, but we paid less attention to how to use the trained model to predict things. After we picked out the best algorithm for our data, we wanted to use it to predict the bonds which were stored in another file. The problem is how to put the contents in different files together into one file. We opened a new file called “result.csv”, and wanted to store the bond id and its predicted trading price into this file. We had trouble in picking out one feature from the whole metric X and then put the bond id and its relating price together. After we turned to teacher for help, we made it clear that the type of feature class and the prediction result were lists. So what we need to use is a loop and the list index.

These are not all the problems we meet in programming, but we think that they are very important during the programming process so we list it out to highlight it. What we can learn from this team project is not the successful process, but the problems and failures we met instead.

➢ Advantages of our project

Since the bond market doesn’t fluctuate so much as the stock market, it’s possible for us to predict the trade price of each bond.

As in the bond market of America, free access to the price information is available online with a 15 minute delay. If one wants to get the exact real-time price as well as size of current bids and offers of bonds, he has to pay for them.

The biggest advantage of the project is letting investors know the approximate real-time price fifteen minutes earlier. At the same time, they can spare the money for buying the information. With our model, bond buyers can enlarge the profit they may get in the secondary market.

Also, the bond market doesn’t change as quickly as the stock market. So it’s possible for us to predict the bond price and the predicted trade price will not be far from the real price.

➢ Disadvantages of our project

Firstly, since we downloaded the data information from a competition on ‘Kaggle’, the data we used for training the model was created about four years ago, which means that they could be out of date and does not accord with present situation of the bond market.

Secondly, sometimes the capital market is too changeable to be controlled and predicted, so our prediction won’t be always accurate. For example, if the company that a creditor buys bonds from occurs an accident like a credit scandal, causing its bond price to fall fiercely. The sudden crisis is not taken into concern in our model, apparently. Bond price is influenced by stock price,

too. When the stock market slumps, people are more likely to invest in the bond market and the increasing demand of bond will increase its trading price, which we can't show in the predicted result.

Although some factors happened in real world will influence the accuracy of our model, the cost of applying the model is really low. The information it offers us can be of great benefit with such a low cost, so it is not a bad choice to use it!

➢ Conclusion

After we successfully ran the program for the first time, we found an unavoidable deficiency is that it takes quite a long time to finish the running process and compute the final result. Apart from that, it's also difficult to get access to the detail information of so many bonds in real time. However, it's reasonable for us students to encounter such problems in our first project and we think it's worth a try to look into the bond market and work out such useful program. Obviously, it's just a primitive version of the benchmark solution and we'll improve it according to the actual usage and suggestions from users in the future.