THE SALES PREDICTION MODEL USING SPARK

Haitao Liu, Anujh Sawant

Introduction

In this project, our objective is to predict the total sales for every product in each store that is present in the data for each month, followed by the next one. This was accompanied with some useful insights and visualizations that provide an accessible way to see and understand trends and patterns in the data. Due to the large data size, we decided to build the model on the spark cluster on AWS environment. We compared the performance of different kinds of models in order to find the relation between data scale and accuracy or execution time. The capability of parallel calculation was measured by add the instances on AWS.

Data

1. Exploratory Data Analysis

The data was provided to us by one of the largest software firms in Russia, named 1C Company. The dataset consisted of time series data of daily sales, and it came in 5 different files. The 5 dataset files consisted of a sales training set, a testing set, a dataset with the list of shops, a dataset with the list of items and a dataset with the list of the categories of those items. Since the data provided to us was in Russian language, we first had to translate it in English before beginning with any visualizations.

First, the top 20 stores were found. This would give us an idea of how well distributed market it is. Figure 1 shows the distribution

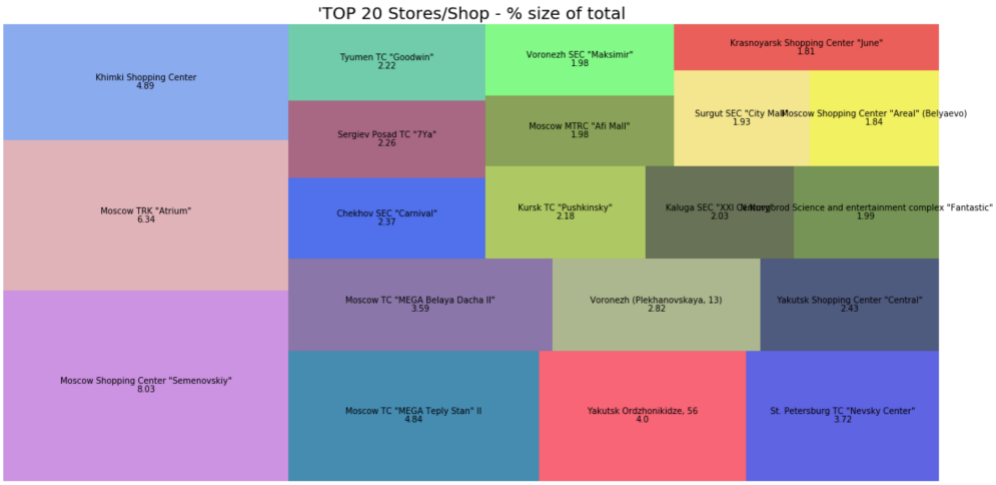


Figure 1

We can see that the market is well distributed and there is no sign of any monopoly in it. Now moving on to the total value of the items sold by the stores. In order to calculate the total amount, the item price was multiplied with the quantity of the items sold. Figure 2 shows the plot of the top 25 stores. From figure 2 we can see that the Moscow Shopping Center has the most amount sol. Next, the top 15 products that were sold were found out. Cinema DVD were the most sold products of all. Now let's move to the most expensive trades in figure 3.

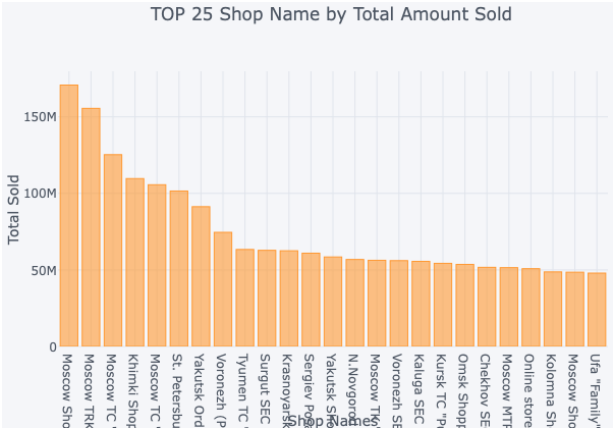


Figure 2

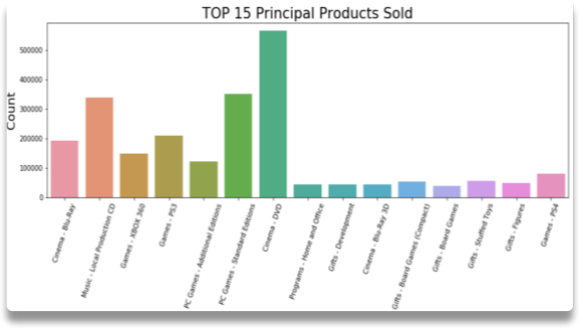


Figure 3

2. Feature Engineering

We grouped daily sales by each item in each shop in each month in order to get our label, which is total monthly sale for each item in each shop in a certain month. We added the price of item and the category of item as the item features. This is a time series model, so we created time lag features in our model including the monthly sales of one month, two month, three month, six month and twelve month ago for each item in each shop. The example of input data for random forest and gradient boosting is shown as Figure 4 .

date_block_num	shop_id	item_id	item_cnt_month	item_price	item_category_id	item_cnt_month_lag_1	item_cnt_month_lag_2	item_cnt_month_lag_3	item_cnt_month_lag_6	item_cnt_month_lag_12
33	2	31	1	399	37	0	0	0	0	0
33	2	486	3	300	73	1	0	0	1	1
33	2	787	1	420	49	0	0	0	0	0

Figure 4

The followings are the data fields of input data:

- Date_block_num: time series of month
- shop_id: unique identifier of a shop
- item_id: unique identifier of an item or product
- item_cnt_month: the number of products or items sold per month.
- item_price: the current price of an item
- item_category_id: name of the item/product
- item: name of the shop
- item_cnt_month_lag_1 to item_cnt_month_lag_12: the number of products or items sold one month ago, 2 month ago, 3 month ago, 6 month ago, 12 month ago.

In LSTM model, the structure of the data is different, which is a three dimensional vector. The first dimension is samples, the second dimension is time series and the third dimension is features. So it is unnecessary to create time series and time lag features. The structure of the data contains those information. So the features we put in only monthly sales of items and the price of items and the categories of items.

Model

In this project, three models have been built to predict the future sales, which are Random Forest, Gradient Boosting and Neural Network. In order to compare the performance by different perspective, there are two versions of code. The one was built by Python packages, the other one was built by Pyspark Dataframe and Pyspark ml packages. Random forest and gradient boosting are ensemble methods, so It's unnecessary to apply lower lever algorithms such as linear regression and decision tree to this data. LSTM is a wild known deep learning algorithm that has been wildly used in predicting time series data. Using this method dealing with sales data could be a good fit.

1. Random Forest

   We use regression decision tree as an estimator in this model to predict the sales of the next month. Random forest is one of the bagging ensemble methods which aim at selecting the random subset of the training data set. The parameters for each decision tree are crucial for the performance of the model. So I introduced the "GridSearchCV" in sklearn package to find the best parameters. We set the range of the number of decision tree from 50 to 100 and the range of max depth for each tree from 5 to 11. According to the result, we can get the lowest RMSE with the default value of other parameters when the max_depth = 8 and n_estimators= 80. The result shows the RMSE is 2.34 with 17867 time series were predicted (the number of shop in this data is 18). The RMSE for the baseline we created is 5.03. So this method can reduce 53.5% root mean square error comparing to the baseline value.

2. Gradient Boosting Regression

   We use regression decision tree as an estimator in this model to predict the sales of the next month. Gradient boosting is one of the boosting ensemble methods aims at minimizing loss functions using gradient descend with regularization. Each round will fit the pseudo residual of the last around. We performed the same experience with random forest model part finding the best parameters for this model. The parameters have the huge impact on the performance of the model, so having the right parameters can make this model more accurate. We set the range of the number of decision tree from 30 to 60, the range of max depth for each tree from 4 to 10, learning rate from 0.01 to 0.1. According to the result, we can get the lowest RMSE with the default value of other parameters when the max_depth = 6, n_estimators= 40, learning rate=0.05 and loss function= lad. The RMSE with 17867 time series predicted is 2.04. The RMSE for the baseline we created is 5.03. So this method can reduce 59.5% root mean square error comparing to the baseline value. The importance of variable in this model is shown as Figure 5. It can be told from this graph that the sales of last month and the category of item are the most important variables in this model. Twelve month time lag has less impact on our model.

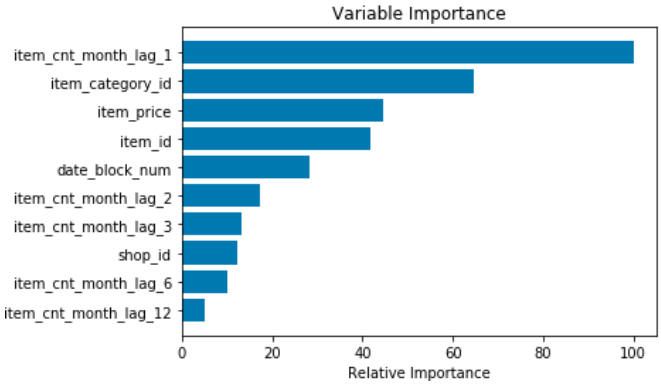


Figure 5

3. Long Short Term Memory Neural Network

   We created a recurrent neural network with Long Short Term Memory. RNN performs the same task for every elements of a sequence with the output being depended on previous computations. LSTM is designed to avoid the long term dependency, which enable network to remember the information for a long period of time. Spark doesn't have LSTM related library, so we decided use TensorFlow to solve this problem. TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. TensorFlow on spark enables developers to quickly and easily get started with deep learning in the cloud. The structure of the neural network is shown as below. The parameters for this neural network: dropout rate: 0.4, units=64, batch size=128, epochs=10, optimizer= "adam". The other parameters remain default value.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_8 (LSTM)                (None, 64)                17152
_________________________________________________________________
dropout_8 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 65
=================================================================
Total params: 17,217
Trainable params: 17,217
Non-trainable params: 0
```

.

Figure 6

   The input value should be a 3 dimensional vector: (samples, time steps, features). The input data has 33 time steps, 2 features (item price and monthly item sales), 6670 time series. The format of the data is pretty different from the data we use for random forest and gradient boosting. The multidimensional vector contains the time serious information without specifying them. The RMSE with 17554 time series predicted is 3.2. The RMSE for the baseline we created is 5.03. So this method can reduce 32% root mean square error comparing to the baseline value. We set the early stop to prevent overfitting.

Performance Evaluation

1. Environment

The execution environment is AWS EMR. We stored our data in S3 bucket and connected the clusters to S3 bucket to retrieve the data. We selected 1 master and 2 cores to execute the algorithms for all the results I present in this paper except parallel evaluation part. The Figure 7 shows the hardware information:

Node type & name	Instance type
MASTER Master Instance Group	**m5.xlarge** 4 vCore, 16 GiB memory, EBS only storage EBS Storage: 64 GiB
CORE Core Instance Group	**m5.xlarge** 4 vCore, 16 GiB memory, EBS only storage EBS Storage: 64 GiB

Figure 7

There are wild range of packages that have been used in this project. The basic sklearn packages for python version of random forest and gradient boosting. The tensorflowonspark and keras packages for deep learning model. The pyspark ml and datarframe library have been used for building machine learning model on cluster.

2. Baseline

In order to measure the RMSE we created baseline for comparison. We calculated the mean value of training label, then calculated the RMSE of testing data by using this mean value as the predicted value. The RMSE turns out different by different data scale.

Data Scale	3	6	9	12	15	18
Baseline RMSE	2.16	3	3.28	6.2	5.44	5.03

Figure 8

It can be seen from the chart that the RMSE increased a little with the increasement of the data scale. The reason is that the volume of testing data also increased, which means we will predict more time series. In this project, the accuracy is measured by how much percent of RMSE reduced from baseline.

3. Evaluation of Execution Time with Different Data Scale

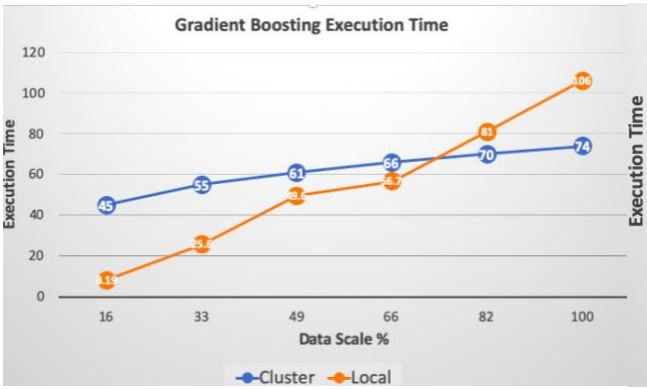


Figure 9

Figure 10

As is shown above, we compare the execution time with different data scale and different algorithms. With the increase of data size, the execution time gradually increased. We found out that the increasement of execution time on cluster is slower than the increasement on local machine for both gradient boosting algorithm and random forest algorithm when the data size is getting larger. In random forest algorithm, execution time on cluster increase about 4 seconds with the increase of 16% of the data size in compare that the increase execution time on local machine is about 5 to 15 seconds. The execution time on local machine surpass the execution time on cluster when the data scale is greater than 66 present. The reason is that the cluster has multiple notes that can pass different tasks to each other. In other words, these notes work parallelly in order to reduce the execution time. The execution time on local machine is less than on cluster when the data size is small. It is unnecessary to execute model when the data size is relatively small because local machine may run this faster. But if the data size is relatively large, running the model on cluster would be a good option. We cannot simply compare the execution time of those two algorithms because the number of selected feature and the format of the tree such as max depth of tree are different. Also, different numbers of iteration will lead to the variate of execution time. In general, the execution time of gradient boosting is normally larger than random forest when they have the same iteration time and parameters. Each decision tree in random forest is independent with each other. So multiple decision trees can be created at the same time with multiple core notes. However, the decision trees in gradient boosting is dependent. The new decision tree is created based on the previous results in gradient boosting.

The execution time of LSTM is the longest. If we increase the data scale by adding more time series, the execution time will have a large increase. The length of time series will decide how many times the information going through a single neural with sequence. So the processing time cannot be reduced in this case. If we increase the data scale by adding more shops or items, the increase rate will lower than previous case because the increased data can be processed in parallel notes.

Figure 11

4. Evaluation of Accuracy with Different Data Scale

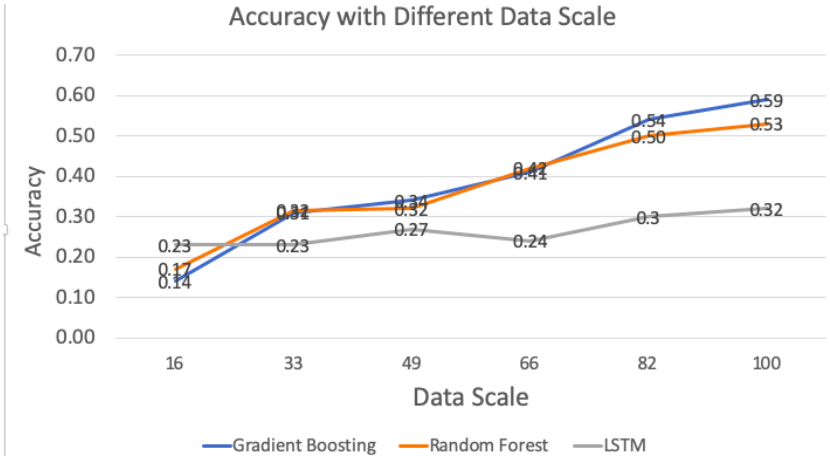
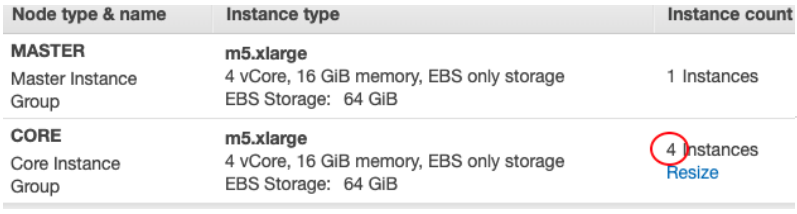


Figure 12

We define the accuracy is the percentage of reducing RMSE from baseline. The baseline represents the RMSE of mean value of training label. For example: The baseline is 5.03 when the data scale is 100%. The RMSE for gradient boosting is 2.04. So the accuracy should be (5.03-2.04)/5.03=0.59, which means we reduce 59% of RMSE using gradient boosting model. The line chart above shows the accuracy with the increasement of data scale. We can see from the chart that nearly all three algorithms shows rising trend. Gradient boosting model shows that highest accuracy and the LSTM shows the lowest. Because the feature we put in those two models are different and the ways we split data were different. So far the LSTM model can still improve so much more by putting more features and narrowing the range of data. The more data means more characters can be calculated, but the accuracy might not always increase. The accuracy might be improved with the increase of the time series length. In this sale data, each store has the different situation. So the prediction of the sale in each shop each item could be hardly depend on other shops. The way we split data here is limiting the number of shop in the data instead of limiting the length of time series, so it is normal that the accuracy might not be improved with the data scale increased.

5. Evaluation of Execution Time with Different Number of Parallel Nodes



Node type & name	Instance type	Instance count
MASTER Master Instance Group	**m5.xlarge** 4 vCore, 16 GiB memory, EBS only storage EBS Storage: 64 GiB	1 Instances
CORE Core Instance Group	**m5.xlarge** 4 vCore, 16 GiB memory, EBS only storage EBS Storage: 64 GiB	4 Instances Resize

Figure 13

The number of core nodes can be changed by simply setting the instance number on hardware page on AWS as Figure 13 shows. We set the range of nodes from 0 to 6 and recorded the execution time of each situation.

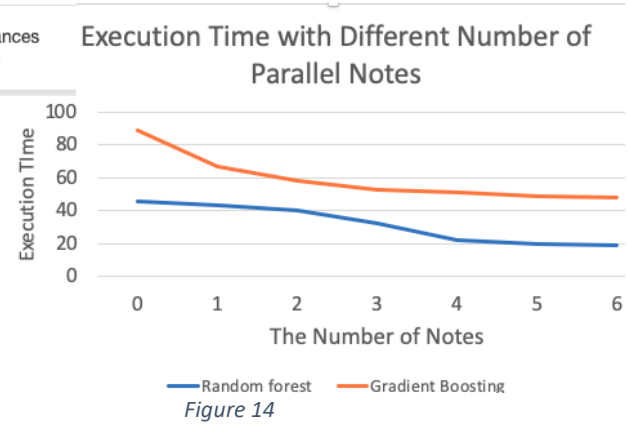


Figure 14

In random forest model, multiple decision trees can be calculated at the same time using multiple parallel notes. So the more parallel notes, the higher parallel processing capability. It can be seen from the line chart that the execution time decreased for random forest model as the number of notes getting bigger. The gradient boosting model also shows a drop when the number of nodes increase. But each decision tree in gradient boosting model is based on the previous one, so execution time that can be saved will be less than random forest model.

Conclusion

All three machine learning models show the good capability of predicting future sale. Executing model on spark cluster can reduce the amount of execution time when the data size is relatively large. The execution time increased as the data scale increases in all three machine learning models. Running the model multiple times can find the best parameters for the model. Gradient boosting performs best in all perspectives so far, but other algorithms such as LSTM can be improved by perfecting feature engineering and neural structure.