Name: Gaurav Chavan

Email: gauravgchavan91@gmail.com

Event: Fractal Analytics Hiring Hackathon

Problem Statement:

Welcome to Antallagma - a digital exchange for trading goods. Antallagma started its operations 5 years back and has supported more than a million transactions till date. The Antallagma platform enables working of a traditional exchange on an online portal.

On one hand, buyers make a bid at the value they are willing to buy ("bid value") and the quantity they are willing to buy. Sellers on the other hand, ask for an ask price and the quantity they are willing to sell. The portal matches the buyers and sellers in real-time to create trades. All trades are settled at the end of the day at the median price of all agreed trades.

You are one of the traders on the exchange and can supply all the material being traded on the exchange. In order to improve your logistics, you want to predict the median trade prices and volumes for all the trades happening (at item level) on the exchange. You can then plan to then use these predictions to create an optimized inventory strategy.

You are expected to create trade forecasts for all items being traded on Antallagma along with the trade prices for a period of 6 months.

EvaluationCriteria:

Overall Error = Lambda1 x RMSE error of volumes + Lambda2 x RMSE error of prices Where Lambda1 and Lambda2 are normalising parameters

Data

Data Dictionary:

Variable	Definition
ID	Unique_transaction_ID
Item_ID	Unique ID of the product
Datetime	Date of Sale
Price	Median Price at Sale on that day(Target Variable_1)
Number_Of_Sales	Total Item Sold on that day(Target Variable_2)
Category_1	Unordered Masked feature
Category_2	Ordered Masked feature
Category_3	Binary Masked feature

Tools Used:

R Programming– xts, forecast, packages

R Studio.

Approach:

- **Data Pre-processing:**

The dataset contained two files *training data* and *test data*. As we needed to forecast *number of sales* and *price* for *Item_Id*, I thought of using univariate time-series arima model for **number of sales** and **Item_Id,** and then appending the forecast values of each of these models to a final csv file.

I dropped Category_1, Category_2, and Category_3 from the dataset. Now, I was considering to loop over unique value of training data, forecast using **auto.arima()** function, and then append the results to test data.

But while appending the results, I got error: **some Id's were not present in test data**, so I looked back at test file, and found how many unique Id's are present using **unique()** function, it came out to be *1447*. So I thought of using only those *train_id's (training samples*) which are present in test dataset.

I constructed for loop to traverse over unique values of **test data (Item_ID)**. The loop created a training and test sample for that particular id.

- **Algorithmic METHODOLOGY:**

- **Why auto.arima() method ?**

In my M.tech dissertation, I tried developing efficient weather forecasting model, *efficient* in terms of memory consumed and time taken. I used Linear regression models, time series ARIMA models and semi-supervised support vector regression (SVR) model. The observation's showed that **auto.arima()** method was more efficient, as it gave less prediction error's, less time and space consumption compared to SVR model and Linear regression model.

- **About auto.arima() method:**

auto.arima() uses the variation of the Hyndman and Khandakar algorithm, which combines unit root tests, minimization of the AICc (Akaike Information Criterion) and MLE(Maximum Likelihood Estimation) to obtain the best fit ARIMA model with lowest AIC or BIC or AICC.

Now, to use **auto.arima()**, the data should be in *time-series* format. For that purpose, **xts()** function was used from *xts* library(extensible time series). This function is used for creating an extensible time-series object. *POSIXct() parameter* was used to order data according to time.

Eg:

```
r_xts=xts(traindata, order.by=as.POSIXct(traindata$Datetime))
```

For univariate forecasting of **number_of _sales**, only **number_of _sales** attribute needs to be considered which varies according to time. Also, the attribute considered needs to be in (*time-series numeric matrix*) form. It was accomplished using *ts(as.matrix(as.numeric()))* function.

Now, ARIMA model works well over stationary data, so to make data stationary over variance, I used $\mathbf{log_{10}}$ transformation over data**.**

For, giving better results I used ***first differencing*** by setting **d=1 and Seasonal difference D=3.**

I set **approximation=TRUE,** so that the estimation is done via conditional sums of squares and the information criteria used for model selection are approximated.

I set **stepwise=TRUE,** so that it will do stepwise selection, which is faster as compared to Non-stepwise selection

I set **trace=FALSE,** so that list of ARIMA models are not reported.

I set **allowdrift=FALSE**, as I didn't want the models with drift term's to be considered.

I set **ic=aic (**Akaike's Information Criterion**)** which is the Information criterion to be used in model selection.

I used **predict()** method to forecast 6 months ahead data which is **184 days**.

The forecasted values needed to be collected in their original form. As the data was transformed in l**og$_{10}$** form, I used **10^(forecastedvalues),** to get the data back into its original form.

The output thus obtained was converted into dataframe and then appended to a test data.

But solution considered needed data in terms of ID, Number of sales, price. So, I dropped Item_ID and DateTime by setting them **NULL.**

Now, for forecasting **price,** same parameters were used. The result thus obtained was appended again to solution dataframe.

- **Creating a csv File**

Now, the final results obtained from a dataframe, needed to be appended to a csv file while running the loop. For that purpose, I used write.table() function. I set **col.names to false** as column_names generated by dataframe were different. I set **row.names to false** as **row.names()** attribute generated were not needed in output csv file.

- **General Observations:**

I tried a lot of possible approaches, like using **log2 instead of log10 for transformation**.

Then I tried by setting **seasonality=TRUE/FALSE**. I tried by changing **test parameter** to **kpss, pp**.

I also tried changing **seasonality test parameter** to **ocsb,ch** I also observed the results by changing **Information criteria to bic, aicc.**

I don't remember exactly, which parameters gave me best result but I guess that the mentioned methodology gave me better error score **0.4075** (It is a guess, it might be 0.44 or 0.45, but best was 0.4075).

- **Other Insight's**

My aim was to reduce the error score which I tried till the end. Lastly I thought of using **hybrid model** using ***forecastHybrid package*.**

I used **arima-ets** hybrid and arima-neural network hybrid, but the **arima-neural network** was taking a much more amount of time and I was unable to submit the result of it (time-up).

I felt like using **arima-neural network** hybrid model or some other hybrid model earlier itself would have given me less error score.

- **Add-on Observations**

It seems better approach, if we try to analyse the program's efficiency by computing **time taken** and **memory used**.

To compute time **proc.time()** can be used to start and stop the timer and compute the system elapsed time for a program.

To compute memory **mem_used()** from *pryr library* can be used.

I have added this functionality in my final program now.