Model Summary

1 Background

Competition: Predict Future Sales

Team: none., just me

Name: Maria del Mar Martínez Sánchez

Location: Madrid

Email: mar_martinez@yahoo.com

Public Leaderboard Score: 0.939900

Private Leaderboard Score: 0.949492

Coursera grade: 10/10

I am an IT worker with high experience (>20 years) in mobile networks and protocols area, and with passion for ML and DL. My academic background is engineering degree with telecommunications specialization.

I am enrolled in this competition because of the Coursera course “How to win a data science competition”.

I spent about 16 hours in the competition (not considering models training lapses).

2 Model Summary

The features generated are lags from the monthly sold item counts and revenues (price * items), and an expanding mean encoding of item category.

The initial solution was a staking of models with two levels, but at the end, a LGB model alone performs better!

To see (and reproduce) the evolution of the project work, just follow the python notebooks in the order of their names.

The main used tools are:

```
numpy 1.17.0
pandas 0.25.0
sklearn 0.21.3
scipy 1.3.0
torch 1.1.0
lightgbm 2.0.6
```

3 Features selection/engineering

3.1 Initial preparation

I firstly added the revenue (item_price * item_cnt_day) to the transactions dataset, because I thought that the price evolution of the item will impact in their sales.

Additionally, I added the item_category_id to the transactions, to analyze the impact of the type of item in the sales as well.

The first data preparation was grouping the transactions values (items sold and revenue) in months, because the goal of the competition is to predict the next month sales.

The grouping of items count (*target*) and revenues are done by month (data_block_num) combined with four different keys:

- shop_id + item_id
- shop_id
- item_id
- item_category_id

I also added a combination of revenue_per_item_id/elements_sold_of_that_item_id in a month to reflect the item_id price evolution.

3.2 Exploratory data analysis

The available data consist mainly on a table of sales transactions with items in shops and the date and the price the item was sold at, during consecutive time sequence of 34 months (id from 0 to 33). Besides this, there are some extra information about the name of the shops and the items. There is no external data to work with (or I don't think so).

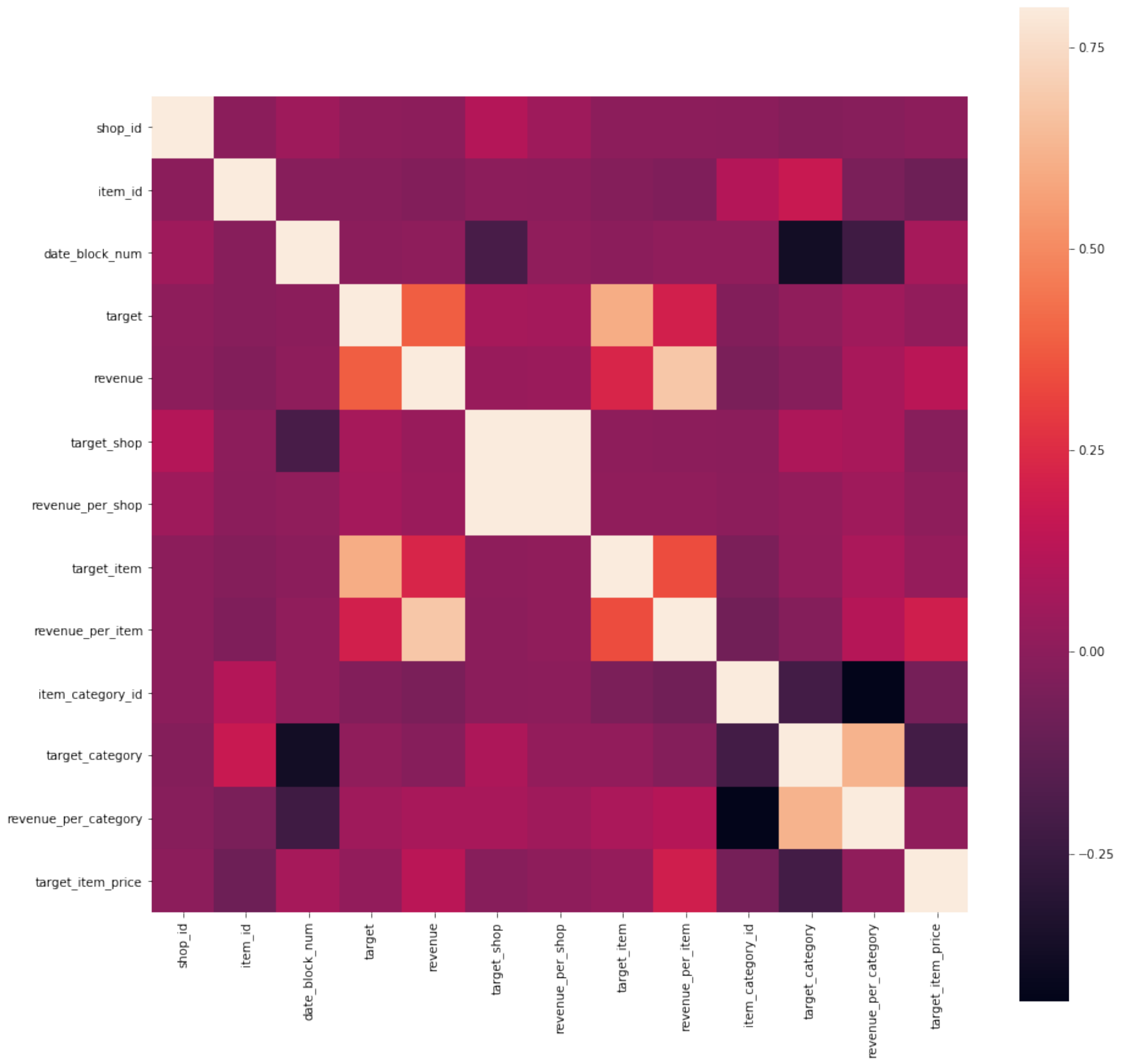
The goal is to predict the item sales in number of elements (not price) for the next month (the 35th) with id=34.

The data is strongly time dependent, and the lag features are mandatory.

There are around 85% more combinatios of shop_id-item_id in the train data than in the test data, so the distribution is uneven and must be considered. There is also a potential leakage here, but I didn't find out how to exploit it.

A cross correlation matrix gives the intuition that number of sold items is highly correlated with the type of item and the revenue (price) but not much with the shop where the items are sold. This will be later confirmed.

The analisys of the sales per shop and item reflects that there are some outliers item_id with too much revenue or count, removing them could improve the solution.



3.3 Addition of features

The item_category_id has been encoded with expanding mean technique.

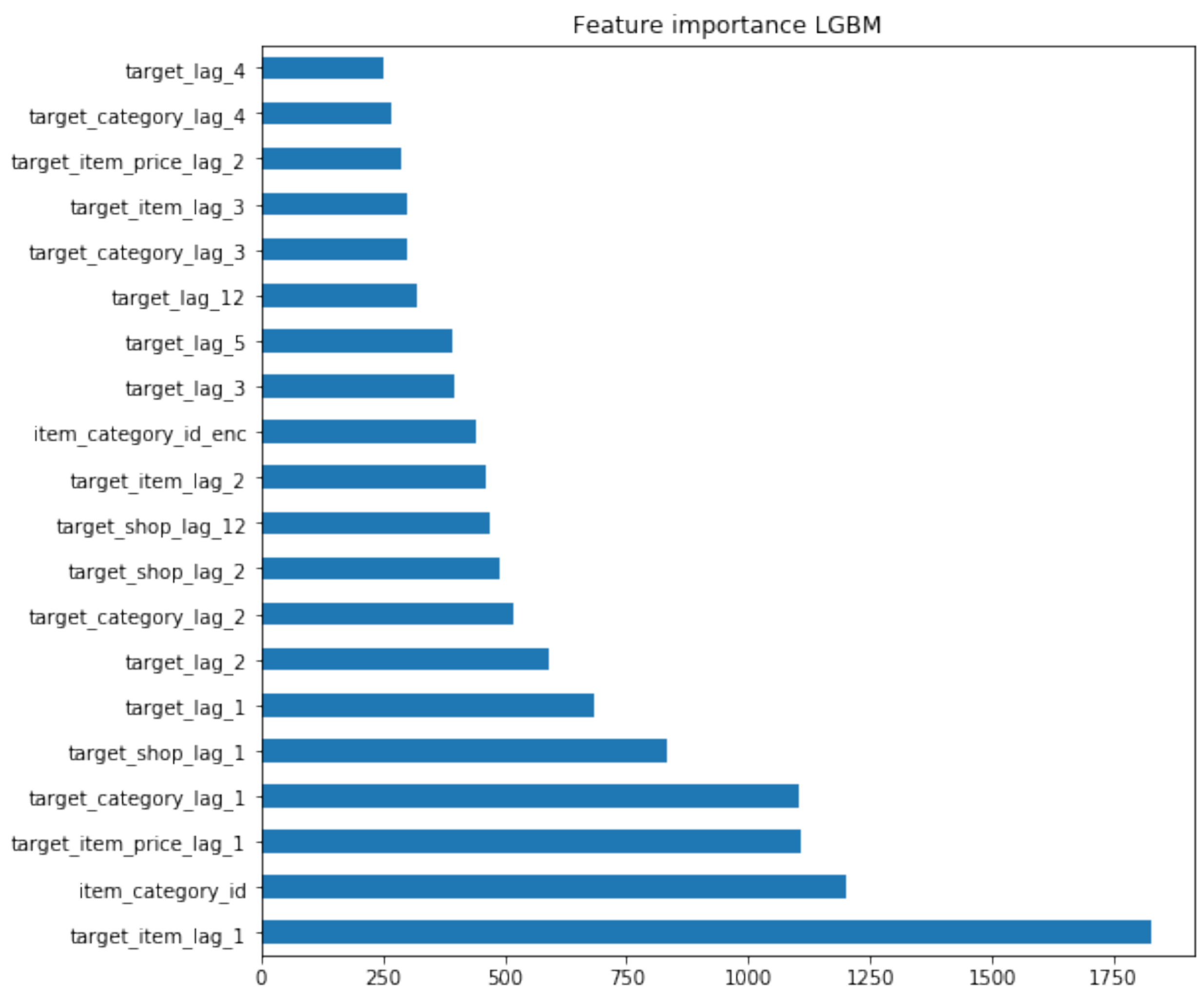
The data features then are the count targets (per different keys asides the month) and the category and the encoded category.

The data features are then set in lag values of 1, 2, 3, 4, 5, and 12 months' shift, because the sales evolution for the test month will have only the past information available.

At the end, all the features are downcasted to float32, to save memory.

3.4 Importance of features

Training a preliminary LGB model shows up the more important features:



As the initial intuition, the item price and the item type are very important, as well as the more recent sales (lags 1, 2, 3...). The same month last year (lag 12) is relevant as well.

4 Models and training

I have been trying different models but in a very superficial way (as I have no much time to spend).

The models are: ligthGBM, linear models (ElasticNet) and Neural Network Regressor with a simple two layers' approach. The choice was done attending to the training time, because the intended data split is a time series progressive KFold and all models should be trained seven times each.

The tuning of the hyperparameters of each model was done in a different way:

- ElasticNet with ElasticNetCV, although the default parameters seem to be the best?
- ligthGBM with early stopping.
- Neural Network with early stopping and some trial-error, because a grid search is unaffordable.

The initial solution was to ensemble three models (LGB, EN and NN) or maybe stack them with another model (Linear Regressor or a Decission Tree). The 2nd model should be quite linear attending to the predictions correlation scatterplot.

The meta features of the 1-st level models were generated with a progressive KFold on time series, using the last 6 months as next level training data.

Surprisingly, the best 2nd level model was not the Linear Regressor or the Decission Tree but a simple linear convex mix tuning the participation of each model (weight) with a grid search.

When combining the three models (LGB, EN and NN) the best mix weight for the NN model was zero! suggesting that the other two models performed better alone.

When combining the other two models (LGB and EN), the best mix weight for the EN was very small, that leads me to intuit that the LGB may performed better alone.

Finally, the LGB model alone got the best grade! 10/10.

I also tried the NN model alone, and gets a fine grade 8/10, but underperforms LGB. The EN model alone has a poor score.

Maybe this behavior is derived of some bug in my code, or the data split scheme, or maybe of a bad tuning of NN... I didn't find out.

5 Interesting findings

I am amazed than the LGB model performs better alone.

I should invest more time exploring this, but I don't have it now.

6 Simple features and methods

I compared the LBG model training with all the features vs the more interesting ones, and the results are:

- time 189s vs 127s (faster with less features)
- RMSE in validation set 0.932 vs 0.978 (more accurate with more features)

7 Model execution time

I have a NVIDIA GPU GTX1060 connected by means of a thunderbolt 3 port with a Intel Nuc 7i (2.6 GHz up to 3.5 GHz Turbo, Quad-Core 6MB cache) and DDR4 SODIMM 16 Gb RAM.

The ensemble model takes about 6 hours to train and execute from features generation to submission.

The LGB alone model (submission) takes about 5 minutes to train and execute end to end.

8 References

Coursera https://www.coursera.org/learn/competitive-data-science/home/welcome

Kaggle Past Solutions http://ndres.me/kaggle-past-solutions/

Torch Finetuning https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html

9 Future work

Obviously, the solution may be improved a lot, but I have not much more time to spend, so these are the improvement areas where to work in the future:

- Explore train-test distribution leakage
- Remove outliers
- Add features based on names of item, category and shop, with td-idf of words or n-grams or with more advanced techniques (i.e. pretrained word2vec in Russian language).
- Use more advanced featuring techniques, as KNN encoding or even t-SNE.
- Explore different folding schemes for the ensemble-stacking of models, or find the code bug if any.
- Train the LGBM with a different folding scheme than hold-out.
- Do an appropriate hyperparameters tuning of NN or use more advanced elements for time series (LSTM, GRU).
- Try different decision trees ensembles as XGBoost or CatBoost.