Winning Model

Name: José Luis Ricón
Location: London, UK
Email: joseluisricon@gmail.com
Competition: Final project. Predict future sales.

Score (Coursera grader): 0.947989 (Private test set)

Background

- Academic/professional background: Data Scientist in a Strategic Consultancy firm. I hold two Msc in Aerospace and Mechanical engineering respectively.
- Prior experience: I haven't done kaggling "seriously" until know, but I knew how to do an EDA and set up models.
- Motivation: Learning about advanced techniques for data analysis competitions
- Time spent: Around 60-70 hours.
- Team: I didn't compete as part of a team, as I wanted to get to try everything on my own for learning purposes.

Summary

The final model is a composed of four sub-models: A neural network, a linear regressor, a GBDT (LightGBM), and a RF, using linear regression as a meta-model. The most important features were undoubtedly the previous months' values for the target variable and their differences. The model, once tuned, requires no more than 16 GB of RAM to run, and it can do so in under 10 min.

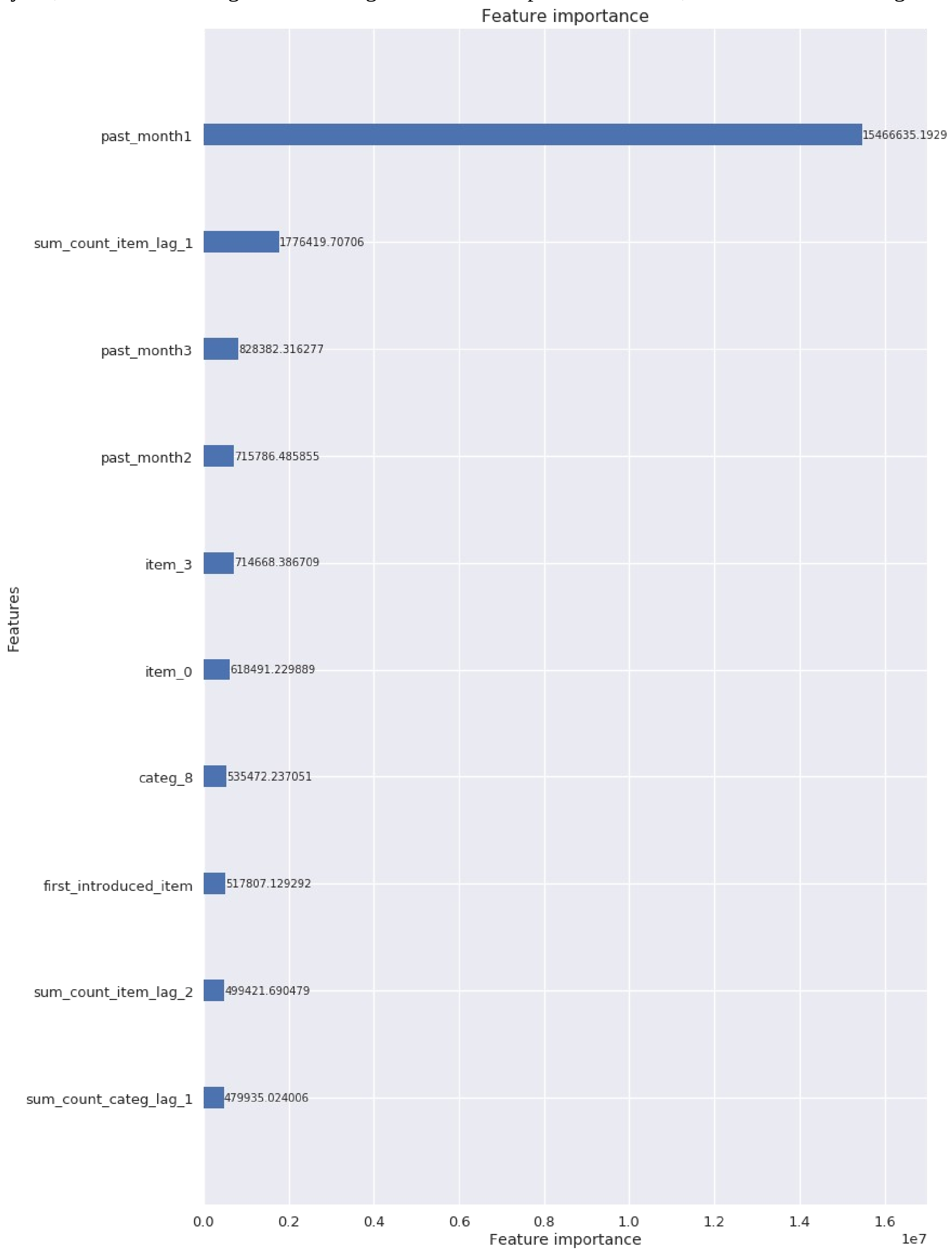
Feature Selection / Engineering

Initially, I split the source data into three: prices, quantities, and text.

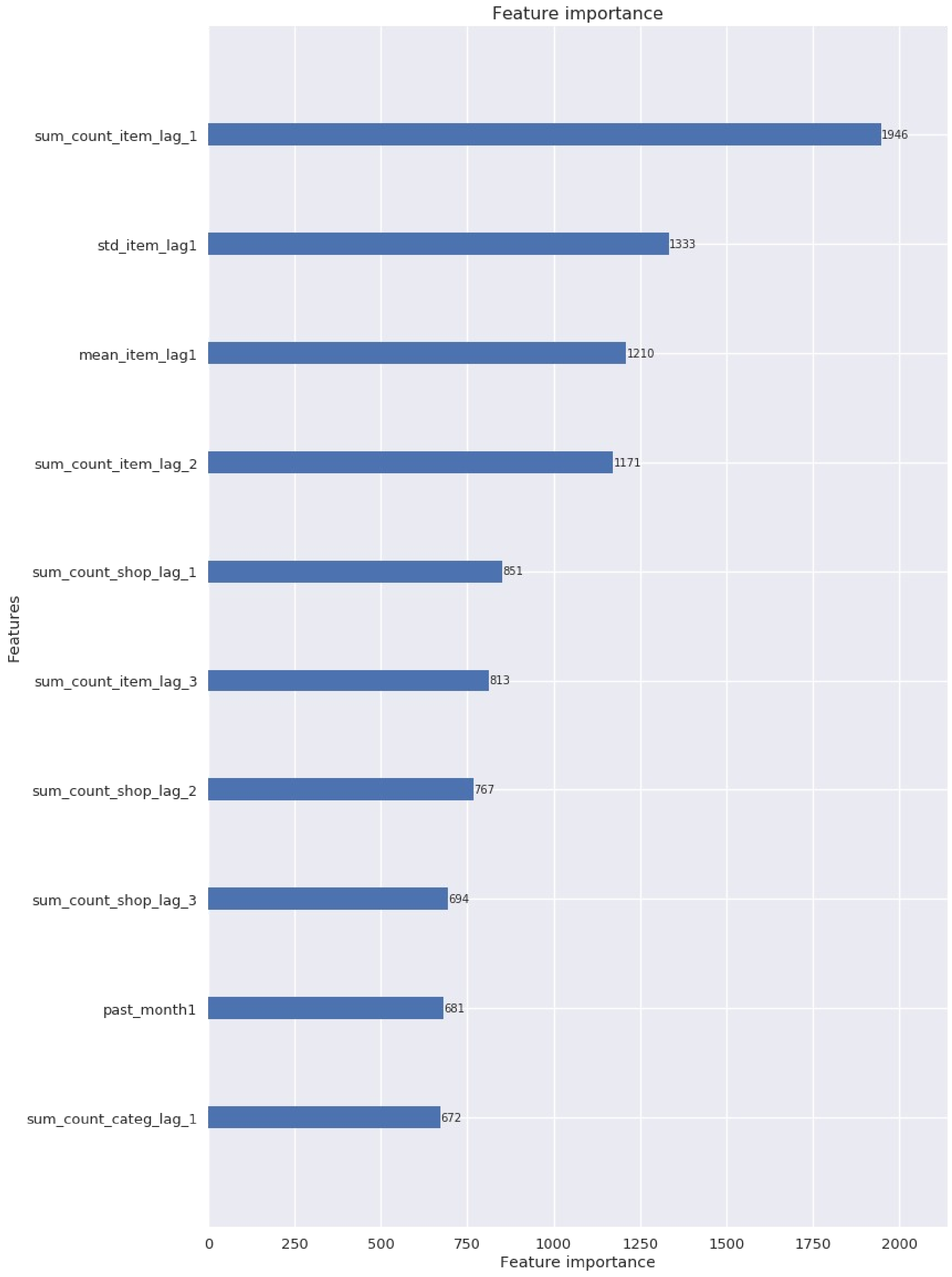
- For text, I one-hot encoded (In a sparse matrix where neeed) the texts for categories, items, and shops. I then ran NMF and PCA (I ended up keeping PCA only) to extract "pseudoembeddings" for the text. I extracted around 10 vectors from each (15 for items). This is probably the most interesting feature I extracted, but not the most important one. This, along with lagged sales, plays the role of "mean encoding" (Though it is not a mean encoding as such, but like mean encodings they are a way of reducing the dimensionality of the problem space)
- For prices, I used lagged mean and std for item prices, but this feature wasn't very important.
- For quantities, I aggregated by month, then clipped between 0 and 20. I calculated lagged sales by category, shop, and item.
- Also, I included a flag variable for the case where a new product had been introduced. This takes the value 1 the first month an item_id appears in the dataset and 0 otherwise. I didn't do this for shops because shops tended to be in most periods, while items were very volatile (some were not sold anymore past some period, and new products were introduced)
- For purposes of calculating features for the test set, I used the last price available instead of previous month price.
- No external data was used, and no interactions of note were included.

As for the **most important features,** I looked at it in two ways. One is just to look at the correlations between different columns with item_cnt_day (After aggregation): Doing this we get that the most important feature is the 1 month lagged target variable. Every other important variable (With a correlation of over .2) were the different lags for the target variable (1,2,3,4,5,12 months)

and the lagged total item sales (This is, in a given month how many of an item was sold across all shops).
The second way of looking at it is using the plot_importance function of LightGBM looking at both gain (of including a feature) where you can clearly see that past month was the strongest predictor by far, but LGB did assign a lot of weight to other unexpected variables, like the item encodings.



We can also look at the default feature importance which measures how many times a feature was used to make splits, which reveals a different pictures.



Training
The final methods were as explained above. Given the huge amount of data, SKLearn's random forest and linear regression were not able to cope. Instead,

- For the first linear model I used keras, since it allows for online normalisation (batch normalisation) and processing (minibatches). A one layer, one neuron model in keras is effectively linear regression.
- The second model was LightGBM, as its speed vs XGBoost allowesd for faster iteration.
- The third model was a random forest, using LightGBM's booster_type="rf" option as it was faster than SKLearn's implementation.
- The fourth model was a four layer neural network implemented in keras with 32-16-8-1 neurons. I defined a custom loss function and activation function to optimise RMSE directly, and my custom activation ensured that the output was bounded between 0 and 20. Batch normalisation was used both here and for the linear model, as these models benefit enormously from normalised inputs.

Finally, SKLearn's linear regression was ran on top of these models.

To train the whole ensemble, I first tuned each model using months 32 and 33 as validation sets. I then predicted, using the four sub-models trained only on the training set, values for all the targets. I then trained them on the whole sample, and trained a linear model on top. It doesn't have any tunable parameter, so no meta-validation was needed. In previous iterations of my model (using RF as meta-model) I used meta-validation: I used the sub-models trained only on the training set to generate a prediction for all the targets, then finetuned the meta-model using the same training-validation split. I then trained all the models on all the data.

Interesting findings

- No doubt the most important trick used was "backfilling": making the training set resemble the test set by taking every month's items and shops, calculating the grid of all possible pairs, and inserting zeros if a value was not in the original dataset. This was key to go below RMSE<1
- Another consideration I took was to adjust the prices for inflation. Russia has had a very high rate of inflation in recent years. Since time was limited, I didn't use official numbers, but simply assumed a constant "real" price mean, and normalised prices by price mean in the last year available.
- A peculiar minor finding was that the only case of "weird" data was a single negative price of -1 that was removed. Otherwise no cleaning was needed.

Simple features and methods

- See feature importance section
- In terms of models, the model that on its own worked best was the random forest (RMSE=0.954894 and 0.950929 in the publoc/private leaderboards)

Appendix

A1. Model Execution time

- Software used: Jupyter notebooks
- Hardware: Amazon m5.2xlarge, with 8 CPUS (Intel®Xeon® Platinum 8175M) at 2.5GHz, 32 GB of RAM.
- How long does it take to train the model: Less than 10 min (Accounting for feature generation)
- How long does it take to generate predictions: Less than 5 min
- How long does it take to train the simplified model : Less than 5 min (Accounting for feature generation)
- How long does it take to train the simplified model : Less than 3 min (Accounting for feature generation)

A2. Dependencies

All the modeling and predictive work was done on Jupyter on Ubuntu 16.04. Versions for the python packages used are as follows:

- Keras 2.1.2
- LightGBM 2.0.11
- Jupyter Notebook 5.2.2
- Numpy 1.13.3
- Pandas 0.21.2
- scikit-learn 0.19.1
- scipy 1.0.0
- tensorflow 1.4.1

A3. How to generate the solution

Work through the FEAT_ notebooks to generate the required features. Note that the cells that generate the .csv.gz are commented out. Uncomment them if you do want the files.

Then work through MODEL_final to train the models. Do not blindly run the notebooks! Throughout the MODEL notebook there are a series of checkpoints that will save the progress so far (To h5 or pickle). This is intended to be used if you don't have a lot of RAM. You can just work up to that point, restart the notebook, run Cell 1 to import packages, then scroll down, and reload what you just saved, to wipeout unwanted memory.

In the model notebook, you will first have to train the models once on the training set (This generates the ALT_*_TRAIN) files, and then the ALT_MODEL files, which are the final models. These final models are included. The stacked model tuning is done using the TRAIN models, but at the end you will train the meta-model on the full models.

Finally, to predict run the Predict notebook (again, not blindly). Optionally you can try to zero out some predictions uncommenting one of the final lines, but my best score was achieved with the results "as they are".