**Model** **Documentation**

**Competition: Predict future sales**

This documentation contains both general information about my approach and technical information about Kaggle’s Predict Future Sales competition <https://www.kaggle.com/c/competitive-data-science-predict-future-sales/leaderboard>

Public LB RMSE score is .90845, 7th position on LB as of 4/16/2018, and private LB score is 0.915106 (Private score is retrieved from Coursera ‘How to win data science competition’ MOOC and they have restricted submission rule to prevent cheating).

1. **Summary**

* Main methods I used for this competition that provides the desired Leaderboard score: LightGBM
* Methods I tried to implement but resulted in worse RMSE: XGBoost, Neural Network with categorical embedding, Stacking (both simple averaging and metal models such as Linear Regression and shallow random forest)
* The most important features are lag features of previous months, especially the ‘item\_cnt\_day’ lag features. Some of them, which can be found in my lag dataset, are
  + **target\_lag\_1 and target\_lag\_2**: item\_cnt\_day of each shop – item pair of previous month and previous two months
  + **item\_block\_target\_mean\_lag\_1, item\_block\_target\_sum\_lag\_1**: sum and mean of item\_cnt\_day per item of previous month

Important features are measured from LightGBM model

* Tools I used in this competition are: numpy, pandas, sklearn, XGBoost GPU, LightGBM, and fast.ai libraries (running Pytorch)
* All models are tuned on a linux server with Intel i5 processor, 16GB RAM, NVIDIA 1080 GPU. Tuning models took about 8 to 10 hours, and training on the whole dataset took <=5 minutes

1. **Exploratory Data Analysis**

More information can be found in notebook **EDA.ipynb**

Basic data analysis is done, including plotting sum and mean of item\_cnt\_day for each month to find some patterns, exploring missing values, inspecting test set …

Here are few things interesting I found from doing EDA:

* Number of sold items declines over the year
* There are peaks in November and similar item count zic-zac behaviors in June-July-August. This inspires me to look up Russia national holiday and create a Boolean holiday features. More information can be found in ‘Feature Engineering’ section
* Data has no missing values
* Some interesting information from test set analysis:
  + Not all shop\_id in training set are used in test set. Test set excludes following shops (but not vice versa): [0, 1, 8, 9, 11, 13, 17, 20, 23, 27, 29, 30, 32, 33, 40, 43, 51, 54]
  + Not all item in train set are in test set and vice versa
  + In test set, a fixed set of items (5100) are used for each shop\_id, and each item only appears one per each shop. This possibly means that items are picked from a generator, which will result in lots of 0 for item count. Therefore, generating all possible shop-item pairs for each month in train set and assigning missing item count with 0 makes sense.

1. **Feature Engineering**
2. **Generate all shop-item pairs and Mean Encoding**

Since the competition task is to make a monthly prediction, we need to aggregate the data to monthly level before doing any encodings

Item counts for each shop-item pairs per month (‘target’). I also generated sum and mean of item counts for each shop per month (‘shop\_block\_target\_sum’,’shop\_block\_target\_mean’), each item per month (‘item\_block\_target\_sum’,’item\_block\_target\_mean’, and each item category per month (‘item\_cat\_block\_target\_sum’,’item\_cat\_block\_target\_mean’)

This process can be found in **generate\_lag\_features.ipynb**, under ‘Generating new\_sales.csv’. Datasets generated from this steps will be saved under the name ‘new\_sales.csv’

1. **Generate lag features**

Lag features are values at prior time steps. I am generating lag features based on ‘item\_cnt’ and grouped by ‘shop\_id’ and ‘item\_id’ . Time steps are: 1,2,3,5 and 12 months.

All sale record before 2014 are dropped, since there would be no lag features before 2014 as we have a 12-month lag.

These lag features turn out to be the most important features in my dataset, based on gradient boosting’s importance features.

More information can be found in **generate\_lag\_features.ipynb**, under ‘Generate lag feature new\_sales\_lag\_after12.pickle’

1. **Holiday Boolean features**

As mentioned above, I look up few Russia national holidays and created few 5 more features: December (to mark December), Newyear\_Xmas (for January), Valentine\_Menday (February), Women\_Day (March), Easter\_Labor (April). This might help boosting my score a little since December feature seems to be helpful

After all this steps, you should have a pickle file name in ‘data‘ directory: 'new\_sales\_lag\_after12.pickle'. This is the main file I used for training models

1. **Cross validations**

Since this is time series so I have to pre-define which data can be used for train and test. I have a function called get\_cv\_idxs in utils.py that will return a list of tuples for cross validation. I decide to use 6 folds, from date\_block\_num 28 to 33, and luckily this CV score is consistent to leaderboard score. I can call:

*cv = get\_cv\_idxs(dataframe,28,33)*

*# dataframe must contain date\_block\_num features*

The results from this function can be passed to sklearn GridSearchCV.

1. **Training methods:**
2. **LightGBM**

LightGBM is tuned using hyperopt, then manually tune with GridSearchCV to get the optimal result. One interesting thing I found: when tuning the size of the tree, it’s better to tune min\_data\_in\_leaf instead of max\_depth. This means to let the tree grows freely until the condition for min\_data\_in\_leaf is met. I believe this will allow deeper logic to develop without overfitting too much. Colsample\_bytree and subsample are also used to control overfitting. And I keep the learning rate small (0.03) throughout tuning.

Mean RMSE of 6 folds CV is 0.8088, which is better than any other models I used.

You can find more information in **lightgbm\_tuning.ipynb**. From this file I also created out-of-fold features for block 29 to 33, which is used for ensembling later.

Also from this notebook, you can get the leaderboard submission under the file name: ‘coursera\_tuned\_lightgbm\_basic\_6folds.csv'

(Note: I do not include some of hyper parameter tuning results from hyperopt since I tuned it at work and I do not have access to that machine now)

1. **XGBoost**

I ran the XGBoost with GPU version, and I follow the same tuning procedures as mentioned in LightGBM. For some reason, I can’t seem to get a consistent result while running XGBoost, even with the same parameters. One example is I get .812 CV score from hyperopt, but I can’t seem to get that result again when getting out-of-fold features (it jumps to .817). This never happens while using LightGBM.

Therefore, I pick 2 models: one with max\_depth tuned, and one without max\_depth tuned, to get out-of-fold features and hoping they are different enough for ensembling.

For the record, the first models results .812 CV score (in hyperopt) and .926 LB score, and second models results in .813 CV score (hyperopt) and .927 LB score. Either way, both are worse than LGB model

You can find more information about this in **xgb\_tuning.ipynb**. Prediction for the model with max\_depth tuned are named ‘tuned\_xgb\_basicfeatures\_6folds\_8126.csv’ and the other one are ‘tuned\_xgb\_basicfeatures\_6folds\_8136’

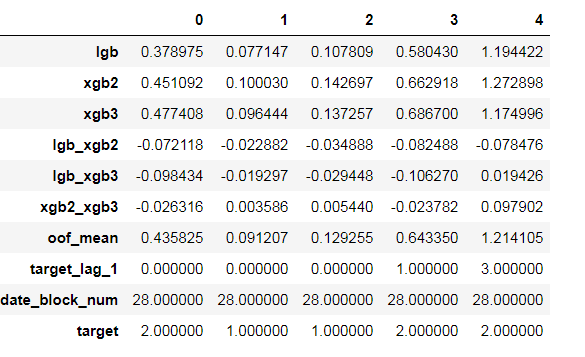
1. **Neural Network with categorical embedding**

This is by far the most time-consuming models, and I don’t have good results for this (1.001 on LB). I also made a fatal mistake while clipping the results due to late night coding, so I won’t discuss much about this. The idea is to separate all numerical features and categorical features, and apply embedding layers for each of those categorical features (i.e map each unique item\_id to a vectors of some length). These embedding layers and numerical features are fed into hidden layers of Neural network.

This method is used to get the 3rd place in Kaggle Rossman competition. With Fast.ai library, it is easy to implement this approach, but it doesn’t seem to work well with this dataset.

1. **Ensembling**

With LightGBM, XGB model 1 and XGB model 2’ out-of-fold features from previous methods, I calculated pairwise differences between them, get the mean of all 3 LGB, XGB1 and XGB2 out-of-fold features, and include the most important features from feature importance: ‘target\_lag\_1’. A meta dataset is created as follow



(This dataset is saved with name: 'ensembling\_nofast.pickle'). From here I try few ensembling methods

* Simple average and Weighted average
* SKlearn linear regression and Elasticnet
* Shallow Random Forest, tuned with 5 folds (from 29 to 33)

All of them results in score that is slightly more than the LightGBM best model, so LightGBM still outperforms them.

More information can be found in **ensembling.ipynb**

1. **Improvement:**

Few things that can be improved are:

* Use XGB CPU instead of GPU to check whether scores are persistent
* Re-implement fast.ai neural net with categorical embedding
* Implement neural net WITHOUT categorical embedding
* Generate more feature related to holiday, such as: differences between current month and holiday month.
* Translate item name to English and perform sentiment analysis on item name
* Use only subset of those meta features for ensembling