Future Sales Prediction

Jeongwon Jo
Guillermo Romera
Checkpoint 2
Problem & Dataset Description

Predict total sales for every product and store in the next month, November 2015;

- **sales_train.csv** - the training set; historical sales data from January 2013 to October 2015;
  - date, date_block_num, shop_id, item_id, item_price, item_cnt_day
    - date_block_num: a consecutive month number; (January 2013 is 0, February 2013 is 1, ..., October 2015 is 33)
    - item_cnt_day: number of products sold daily
- **test.csv** - the test set for which we need to predict the target value
  - ID, shop_id, item_id
- **items.csv** - supplemental information about the items/products.
  - item_name, item_id, item_category_id
- **item_categories.csv** - supplemental information about the items categories.
  - item_category_id, item_category_name
- **shops.csv** - supplemental information about the shops.
  - shop_name, shop_id
Pre-Processing

- Identified outliers for price and sales features and removed samples that are out of range
- Removed duplicates of three shops
- Retrieve city name from shop_name (ex. of original shop_name: “Philadelphia Wegmans” >> “Philadelphia” “Wegmans”)
- Retrieve type and subtype of an item from category value
- Retrieve revenue for each item by multiplying item_price by item_cnt_day
- Sum up the number of items sold in one month for each item and clip it from 0 to 20 so training dataset can be similar to the submission dataset
- Included day and month values as features
Feature Engineering

- Needed to further improve the results and prediction power of any algorithm used
- Hard to figure out what needs to be engineered. Needed to do some reading and help from Kaggle
- Because this is a time-series problem, lag and encoded means had to be used to get more insight
- Lag allows to create features that might be more relevant towards the final dataset
- It helps the prediction model by “knowing” what happened in the previous time period
Feature Engineering_Encoded Means

- Mean encoded variables can be created by using the target variable and another categorical variable present in the data set to further increase its suitability

Example:
- Average number of items sold in each time value
  (ex. Avg of items sold in November 2014)
- Average number of particular item sold in each time value
  (ex. Avg of Item 1 sold in November 2014)
- Average of items sold in each shop per each time value
  (ex. Avg of items sold in Shop 1 in November 2014)
- Average of items sold by category in each time value
  (ex. Avg of Category 1 sold in November 2014)
- Average of items sold by category per shop in each time value
  (ex. Avg of Category 1 sold in Shop 1 in November 2014)
- Average of items sold by type per shop in each time value
  (ex. Avg of Type 1 sold in Shop 1 in November 2014)
- Average of items sold by subtype per shop in each time value
  (ex. Avg of Subtype 1 sold in Shop 1 in November 2014)

Lag Features

We used 1, 2, 3, 6, 12 as windowing for mean encoded features (t-1 month, t-2 month, t-3 month...)

...
## Feature Engineering_Encoded Means

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>date_block_num</td>
<td>12.0000</td>
</tr>
<tr>
<td>shop_id</td>
<td>2.0000</td>
</tr>
<tr>
<td>item_id</td>
<td>36.0000</td>
</tr>
<tr>
<td>item_cnt_month</td>
<td>0.0000</td>
</tr>
<tr>
<td>city_code</td>
<td>0.0000</td>
</tr>
<tr>
<td>item_category_id</td>
<td>37.0000</td>
</tr>
<tr>
<td>type_code</td>
<td>11.0000</td>
</tr>
<tr>
<td>subtype_code</td>
<td>1.0000</td>
</tr>
<tr>
<td>item_cnt_month_lag_1</td>
<td>0.0000</td>
</tr>
<tr>
<td>item_cnt_month_lag_2</td>
<td>0.0000</td>
</tr>
<tr>
<td>item_cnt_month_lag_3</td>
<td>0.0000</td>
</tr>
<tr>
<td>item_cnt_month_lag_6</td>
<td>0.0000</td>
</tr>
<tr>
<td>item_cnt_month_lag_12</td>
<td>0.0000</td>
</tr>
<tr>
<td>date_avg_item_cnt_lag_1</td>
<td>0.41140</td>
</tr>
<tr>
<td>date_item_avg_item_cnt_lag_1</td>
<td>0.13050</td>
</tr>
<tr>
<td>date_item_avg_item_cnt_lag_2</td>
<td>0.08887</td>
</tr>
<tr>
<td>date_item_avg_item_cnt_lag_3</td>
<td>0.13050</td>
</tr>
<tr>
<td>date_item_avg_item_cnt_lag_6</td>
<td>0.10870</td>
</tr>
<tr>
<td>date_item_avg_item_cnt_lag_12</td>
<td>0.0000</td>
</tr>
<tr>
<td>date_shop_avg_item_cnt_lag_1</td>
<td>0.14810</td>
</tr>
<tr>
<td>date_shop_avg_item_cnt_lag_2</td>
<td>0.10065</td>
</tr>
<tr>
<td>date_shop_avg_item_cnt_lag_3</td>
<td>0.08905</td>
</tr>
<tr>
<td>date_shop_avg_item_cnt_lag_6</td>
<td>0.09605</td>
</tr>
<tr>
<td>date_shop_avg_item_cnt_lag_12</td>
<td>0.0000</td>
</tr>
<tr>
<td>date_cat_avg_item_cnt_lag_1</td>
<td>0.23280</td>
</tr>
<tr>
<td>date_shop_cat_avg_item_cnt_lag_1</td>
<td>0.05945</td>
</tr>
<tr>
<td>date_city_avg_item_cnt_lag_1</td>
<td>0.14810</td>
</tr>
<tr>
<td>date_item_city_avg_item_cnt_lag_1</td>
<td>0.00000</td>
</tr>
</tbody>
</table>

Total number of item_id 36 sold in shop_id 2 last 1, 2, 3, 6, 12 months

Average of items sold last month

Average of item_id 36 sold last 1, 2, 3, 6 months

Average of items sold in shop_id 2 last 1, 2, 3, 6 months

Average of category_id 37 sold last month

Average of item category_id 37 sold in shop_id 2

Average of item sold in city_code 0

Average of item_id 36 sold in city_code 0
Feature Engineering_Price Trend

The ratio of the average (item price / shop revenue) for past month to the average (item price / shop revenue) for the whole...to determine whether the (item price / shop revenue ) has increased/decreased during the entire observation period

Example:
1. Average Price of Item 1 last month = x1
   (If the item was not sold last month, see if it was sold in last 2/3/4 months)
2. Average Price of Item 1 in general = x
3. Price trend = (x1-x)/x
4. if (price trend > 0):
   positive trend
   elif (price trend < 0):
   negative trend
   else:
   no change in trend
Complications

- Dataset is big, so used Google Colab to avoid possible RAM issues (They allow GPU service!)
- Data types need to be taken into account, the difference between .float(64) and .float(16) can be the difference between 3GB or 800MB
Model Comparison

- We avoided some models since we understood they would not be optimal to use (Naive Bayes, Logistic Regression).
- We compared MLP Regression, KNeighbors Regression, Random Forest Regression, Gradient Boosting Regression, Long Short-Term Memory, and XGBoost Regression “baseline” parameters;
  - RMSE was used to test the ‘accuracy’ of different models
  - $R^2$ was used to see how close the data are to the fitted models to avoid overfitting.
  - $\gg$ XGBoost Regressor was selected
Parameter Tuning

- GridSearchCV was used to obtain the best parameters for XGBoost Regressor
  - Tested parameters:
    - max_depth = (3, 5, 7, 9)
    - min_child_weight = (1, 3, 5)
  - Relatively high learning_rate (0.3) and low n_estimators (100) to boost up the training time
- Once the best parameter was identified, we lowered learning_rate and increased n_estimators to get better performance (0.05, 200)
- We also identified feature importances in overall and removed three least important features and ran the model; turned out to downgrade performance
Our future plans include:

- Further try more feature engineering methods
- Further try more parameter tunings for XGBoost Regression
- Try more “exotic” algorithms
- Build multiple ensemble models that take the best models overall and see which one performs better (hard voting, soft voting, ensemble voting regressor)