Gradient Boosting with LightGBM, LSTM + CNN Network Model in M5 Forecasting Competition

Yan Fang

University of California, Irvine

Email: yfang9@uci.edu

***Abstract* - The M5 forecasting competition held by the University of Nicosia requires using hierarchical sales data proffered by Walmart to forecast the next 28 days' daily sales. Our target is to advance the theory and practical use of forecasting by determining the method that provides the upmost accurate point prediction for each of the 42840-time series proffered by the dataset. This report will demonstrate how we decided the process by Exploratory Data Analysis and how we ensemble the advantages of two ways to approach the accurate result.**

I. Introduction

This report includes five sections: Exploratory Data Analysis, Feature Engineering, Methods, Ensemble models Implementation, Results, and Discussion.

The given dataset is enormous and intricate, including unwanted information and data that needs to be regularized. Time Series exploratory data analysis is used to approach this problem; it allows me to extract useful variables and maximize insight into the dataset. Time series is a sequence of observations recorded at regular time intervals. Depending on the frequency of observations, a time series may typically be hourly, daily, weekly, monthly, quarterly, and annual. Sometimes might have seconds and minute-wise time series, like, the number of clicks and user visits every minute, etc.

After analyzing the dataset, choosing, and fitting models is essential. The best model depends on historical data's availability, the strength of relationships between the forecast variable and any explanatory variables, and how the forecasts are used. It is common to compare two or three potential models. Each model is itself an artificial construct based on a set of assumptions (explicit and implicit) and usually involves one or more parameters that must be estimated using the known historical data. In this competition, we used two models: **Gradient Boosting Model (LightGBM)** and **Artificial Neural Network** model based on **Long Short Term Memory Network (LSTM)** and **Convolutional Neural Network (CNN)**.

II. Exploratory Data Analysis

a. Trend, Seasonal, business cycles

Time series is a sequence of observations recorded at regular time intervals. Depending on the frequency of observations, a time series may typically be hourly, daily, weekly, monthly, quarterly, and annual. Sometimes, you might have seconds and minute-wise time series like the number of clicks and user visits every minute, etc.

As the figure shown below, most time series have low daily count statistics, alongside the large percentage of zero numbers we already noticed. On the one hand, this suggests that spikes are not going to be overly pronounced. But it also indicates that accurate forecasts will have to deal with quite a lot of noise.

Some of our sample time series starts in the middle of the time range, and some have long gaps in between, which is an additional challenge.

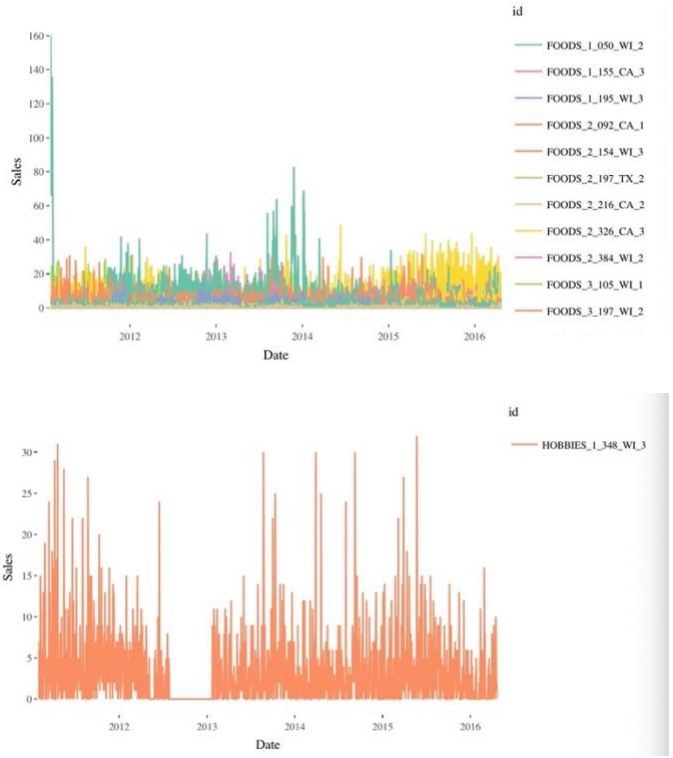


Figure 1. Sample Time Series

b. Feature Analyzing

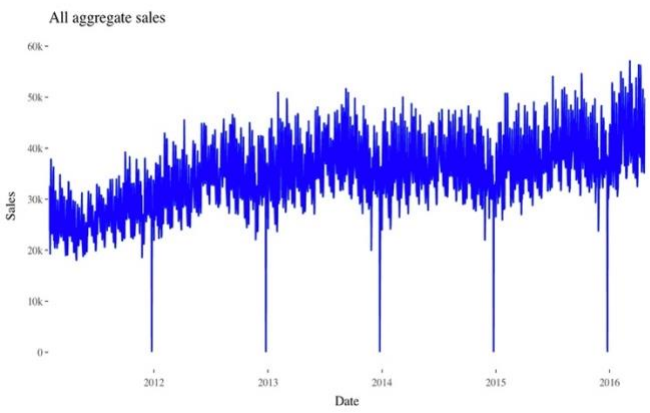


Figure 3. Aggregate Sales

As the figure is shown above, the sales are generally going up. We can make out some yearly seasonality and a dip at Christmas, which is the only day of the year when the stores are closed. Zooming in, we can see strong weekly seasonality plus possibly some additional overlaying patterns with shorter periods than yearly. The most recent 2016 sales numbers appear to grow a bit faster than in previous years.

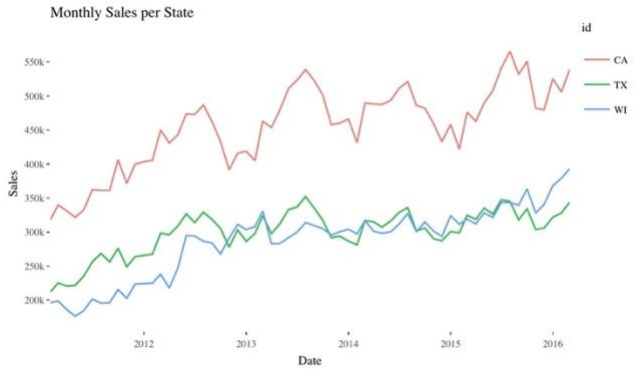


Figure 4. Monthly Sales per State

From Figure 4, we observed that California (CA) sells more items in general, while Wisconsin (WI) was slowly catching up to Texas (TX) and eventually surpassed it in the last months of our training data. CA has pronounced dips in 2013 and 2015 that appear to be present in the other states, just less severe. These dips and peaks do not seem always to occur, but they might primarily reflect the yearly seasonality we noticed already.

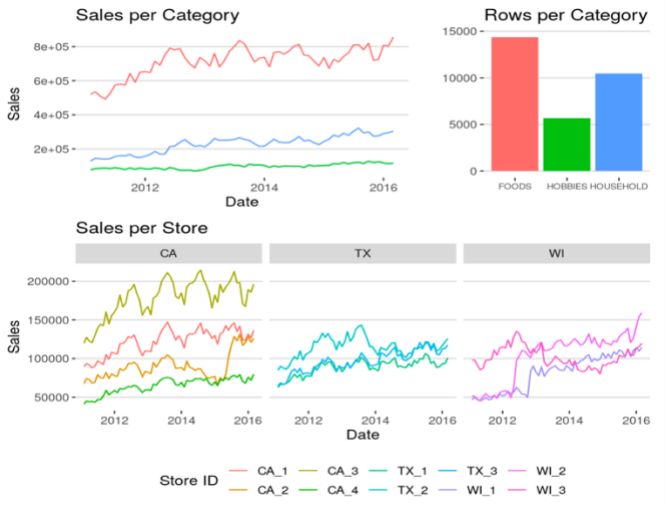


Figure 5. Sales by Multiple Features

"Foods" are the most common category, followed by "Household", which is still above "Hobbies". The number of "Household" rows is closer to the number of "Foods" rows than the corresponding sales figures, indicating that more "Foods" units are sold than "Household" ones.

In terms of stores, we see that the TX stores are quite close together in sales with "TX_3" rising from the levels of "TX_1" to the level of "TX_2" over the time of our training data. The WI stores "WI_1" and "WI_2" show a curious jump in sales in 2012, while "WI_3" shows a long dip over several years.

The CA stores are relatively well separated in store volume. Note "CA_2", which declines to the "CA_4" level in 2015, only to recover and jump up to "CA_1" sales later in the year.

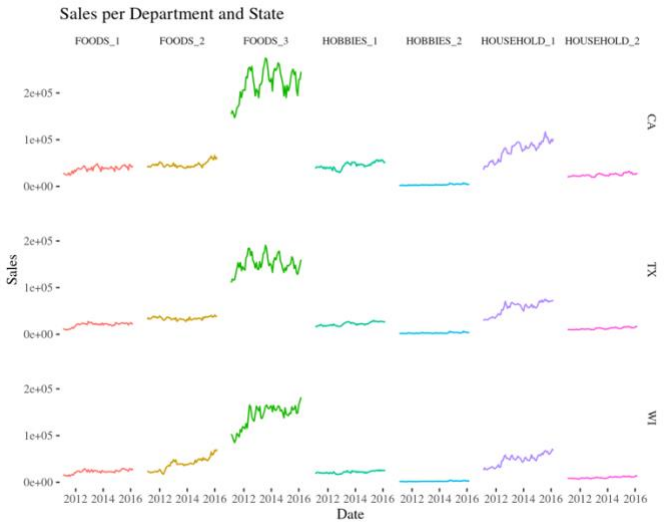


Figure 6. Sales per Department and Sate

"FOODS_3" is clearly driving the majority of "FOODS" category sales in all states. "FOODS_2" is picking up a bit towards the end of the time range, especially in "WI".

Similarly, "HOUSEHOLD_1" is clearly outselling "HOUSEHOLD_2". "HOBBIES_1" is on a higher average sales level than "HOBBIES_2", but both are not showing much development over time.

C. Pattern Analyzing

Figure 7. Weekly Pattern#1

As the figure shown above, the weekly pattern is strong. With Saturday and Sunday standing out prominently. Also Monday seems to benefit a bit from the weekend effect. The months of Nov and Dec show clear dips, while the summer months May, Jun, and Jul suggest a milder secondary dip. Certain holidays, like the 4th of July, might somewhat influence these patterns; but over 5 years they should

average out reasonably well. Though we tweaked the smoothing fit parameters, they could probably get more optimized.

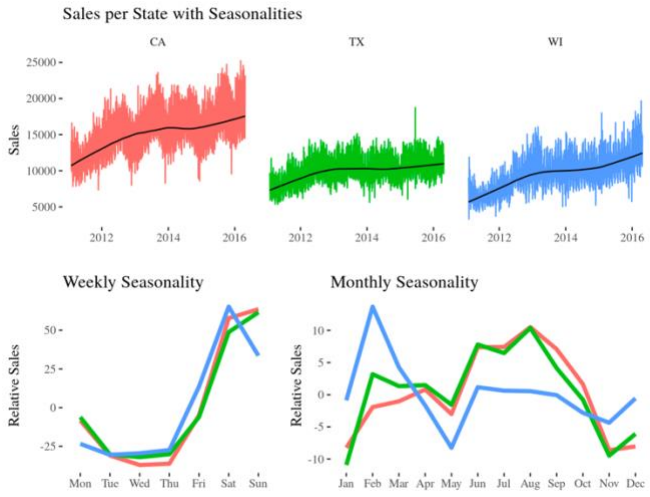


Figure 8. Weekly Pattern#2

After scaling, the weekday vs. weekend pattern is very similar for all 3 states, except for an interesting downturn in Sunday sales in WI.

The monthly seasonality is indeed complex. There is a dip in the winter months and a second, generally shallower dip around May. WI is again the odd state out: it sells notably less in the summer compared to TX and especially CA, so much so that the Feb/Aug ratio is inverted for WI vs. CA/TX.

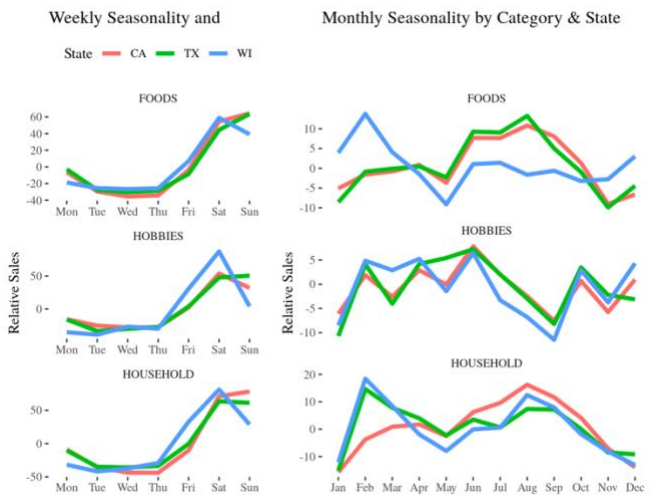


Figure 9. Weekly Pattern#3

The weekly patterns for the "foods" category are very close for all 3 states, and WI shows some deviations for "hobbies" and "household". We also see the Wisconsin's characteristic Sunday dip for all 3 categories.

The monthly patterns show some interesting signatures: For "foods", CA and TX are close but WI shows that inverted summer/winter ratio. In contrast, the 3 states are much more like each other in the "hobbies" category. And when it comes to "household" items, CA doesn't seem to sell as much of them during the first 3 months of the year but slightly more in the summer; compared to WI and TX.

d. Price Features Analyzing

Figure 10. Price Distribution

Firstly, the distributions are almost identical between the three states. There are some minute differences in the "FOODS" category, but this might be due to the smoothing bandwidth size. For all practical purposes, I think that we can treat the price distributions as equal.

There are notable differences between the categories: FOODs are on average cheaper than HOUSEHOLD items. And HOBBIES items span a wider range of prices than the other two, even suggesting a second peak at lower prices.

Within the categories, we find significant differences. First, Among the three food categories, department 3 (i.e., "FOODS_3") does not contain a high-price tail. Second, the HOBBIES category is the most diverse one, with both departments having quite broad distributions but "HOBBIES_1" accounting for almost all the items above $10. "HOBBIES_2" has a bimodal structure. Thirdly, the HOUSEHOLD price distributions are quite similar, but "HOUSEHOLD_2" peaks at clearly higher prices than "HOUSEHOLD_1".

Overall, the price distributions are stable over the years, with only slight increases that are likely due to inflation. This is best visible in HOBBIES_1. An interesting evolution is visible in HOBBIES_2, which over time becomes much more bimodal: the second peak at $1 is increasing in importance until it almost reaches the level of the main peak just above 2 dollars. At the same time, the small secondary peak at half a dollar in HOBBIES_1 becomes flatter after 2012. The HOUSEHOLD departments are stable. FOODS shows small changes like the relative growth of the $1 peak of FOODS_1.

III. Feature Engineering

Through the above exploratory data analysis, we can have a clearer framework about the sales data and generate an assumption about what features influence the sales. Except for the features provided in the competition, we found that the lag time features of 7 days and 28 days impact sales since week and month period patterns remarkable in the figures. Hence, we create custom features by following codes.

```
def create_fea(dt):
    lags = [7, 28]
    lag_cols = [f"lag_{lag}" for lag in lags]
    for lag, lag_col in zip(lags, lag_cols):
```

```
            dt[lag_col] =
dt[["id","sales"]].groupby("id")["sales"].shift(lag)

        wins = [7, 28]
        for win in wins:
            for lag, lag_col in zip(lags, lag_cols):
                dt[f"rmean_{lag}_{win}"] =
dt[["id",lag_col]].groupby("id")[lag_col].transform(
                    lambda x: x.rolling(win).mean())

    date_features = {

        "wday": "weekday",
        "week": "weekofyear",
        "month": "month",
        "quarter": "quarter",
        "year": "year",
        "mday": "day",
        #       "ime": "is_month_end",
        #       "ims": "is_month_start",
    }
```

When we have extracted and customized useful features and reconstructed the dataset, we set up the models we need to train and make predictions.

IV. Methods

Gradient boosting is a type of machine learning boosting methods. It generates prediction models in an ensemble of weak prediction models (usually decision trees). In each iteration, a new tree is added and trained on the labels caused by the existing tree's errors.

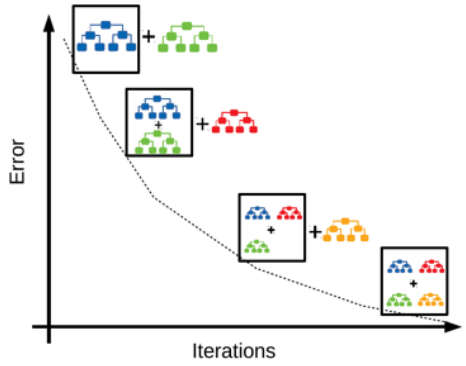


Figure 11. Gradient Boosted Trees

Before introducing LightGBM, we must know more about its counterpart – XGBoost, a gradient boosting framework programmer usually compare to LightGBM. XGBoost is a "variant" of GBDT. The biggest difference between them is that XGBoost uses Taylor expansion to approximate loss function and obtain the nodes' weight to be fitted in the next step. Therefore, the amount of reduction on each nodes' loss can be obtained according to the loss function. However, XGBoost's disadvantage is very evident. Its level-wise split leads to increasing complexity and overfitting.

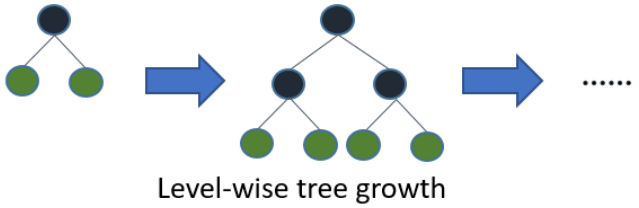


Figure 2. Level-Wise Tree Growth

Level-wise tree treats all nodes in the same layer equally. Nodes with meager returns from splitting and have no effect on the result are forced to split, significantly increasing the computational cost. Therefore, using XGBoost to train a larger dataset takes a very long time. This competition has an enormous amount of dataset, and we choose LightGBM instead of XGBoost.

LightGBM is a gradient Boosting framework developed by Microsoft. We choose LighGBM because compared to XGBoost, and it has faster training efficiency, much lower memory usage, higher accuracy, supports parallel learning, and, most importantly, can handle large-scale data. LightGBM uses a decision tree algorithm based on a histogram, which is different from the exact algorithm in XGBoost. The histogram algorithm has significant advantages in memory and computational cost.

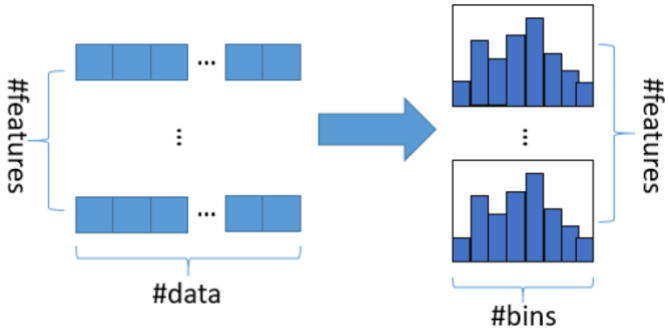


Figure 12. Histogram Algorithm

Different from XGBoost, LightGBM is using leaf-wise growth to construct a decision tree. Leaf-wise is a more efficient strategy, and it splits on the nodes with the largest information gain instead of every single node in the same level of leaves. Therefore, compare to level-wise, leaf-wise growth can reduce more errors and produce better accuracy when the number of splits is the same. The disadvantage of leaf-wise is that it may generate a decision tree with large depth, which will result in overfitting. Therefore, LightGBM adds a maximum depth limit to prevent overfitting while ensuring high efficiency.

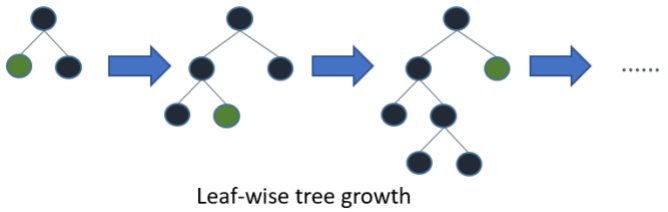


Figure 13. Leaf-Wise Tree Growth

This growth is defined in function as:

- $(p_m, f_m, v_m) = arg \min_{(p,f,v)} L(T_{m-1}(X).split(p,f,v), Y)$

- $T_m(X) = T_{m-1}(X).split(p_m, f_m, v_m)$

LightGBM can also handle categorical features very well. LightGBM optimizes the support for categorical features, which we can directly input category features, without additional one-hot encoding.

The **loss function** we choose in our model is **Tweedie** to maximize the negative log-likelihood of the Tweedie distribution. “Tweedie” is a family of distributions to which the “Compound Poisson distribution” belongs. In some cases, a Compound Poisson distribution can contain many instances of an event taking place 0 times (no purchases) and a long-tail of varying, exponentially decreasing values (zero-inflated regression).

This behavior can be suitable for our sale prediction objectives. In a large retails company like Walmart, most items entering the store will likely not make a purchase, and if they do, the sales range widely. Displayed in the dataset, it contains a lot of 0 sales. So, we choose Twiddie rather than Poisson. Tweedie is a loss function for large-scale data that suits our need for this competition, and the general function is shown below:

$$\mathcal{L} = -\sum_i x_i \cdot \frac{\tilde{x}_i^{1-p}}{1-p} + \frac{\tilde{x}_i^{2-p}}{2-p}$$

To judge the performance of our model, we need a **metric** to evaluate our model after the learning process. Here in our model, I chose root mean square error(rmse). To tuning the model for better accuracy, another parameter called **learning rate** is also important. Since each classification will update the weight, the learning rate will control the changing rate of weight updating.

Since the data-point we are dealing with is large, we set **force_row_wise** true to speed up boosting. Besides, to improve the result's accuracy, we enabled bagging by setting bagging_freq to a non-zero value. In this case, **bagging_freq** is 1, which means the model creates a new bag every iteration. Every tree will learn from a subsample of the dataset.

Finally, we have one more parameter called **lambda_l2**, which is a regularization parameter. The goal of this parameter is to avoid any weight surging up that may lead to overfitting.

Another attempt at this competition is **Long Short Term Memory Network**, a special kind of **Recurrent Neural Network** (RNN). Neural Networks, just like our brain, is composed of neurons and connected with corresponding layers. However, when human reading or learning something, we don't learn or think with blank brains. Our thoughts have persistence; on the other hand, we have memories. However, traditional neural networks do not have memory. For example, suppose you want to classify some events in a video on each video's specific point. Traditional neural networks can hardly deal with this problem since they cannot use previous events in the video to deduce subsequent events; however, RNN can solve this kind of problem. RNN is a network with loops, which gives it the ability to control the persistence of information.

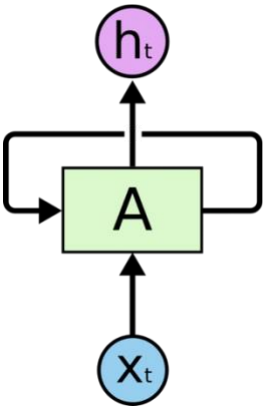


Figure 14. RNN with Loops

As the figure is shown above, layer A is reading an input "x" and outputting a value "ht". Looping allows information to pass from the current step to the next. Compare to RNN, LSTM was proposed by Hochreiter & Schmidhuber (1997) and was recently improved and promoted by Alex Graves. LSTM networks are well-suited to classifying, processing, and making predictions based on time series data since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the vanishing gradient problem encountered when training traditional RNNs (reference: Wiki). Generally, a repeating module in an LSTM contains four interacting layers:

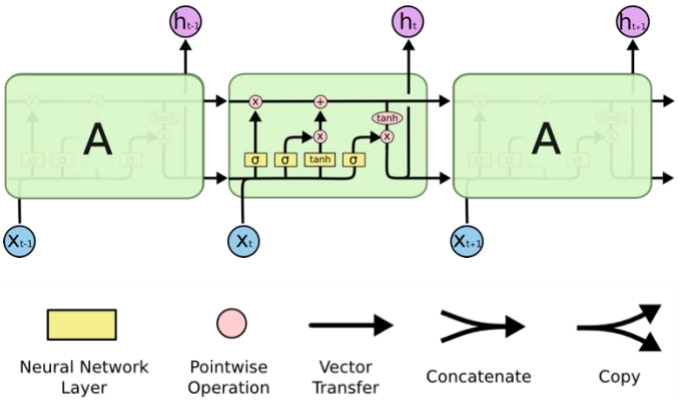
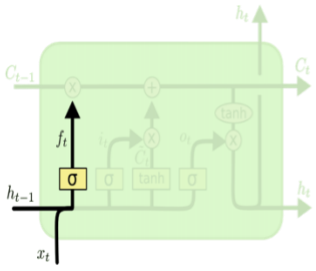
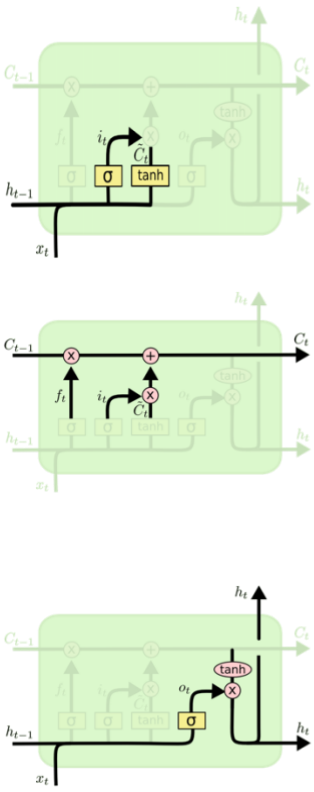


Figure 15. LSTM layers

If we walk through LSTM step-by-step:



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 16. Forget gate； Input gate; Cell state; Output gate

As the figure is shown above, LSTM consists of forget gate, input gate, cell state, and output gate. Sigmoid is the **activation function** to decide what memories we want to update, and the tanh function creates a candidate value since its output is in the range of -1 and 1. The cell remembers values over arbitrary time intervals, and the three gates regulate the flow of information into and out of the cell. As the regulated data flows forward to the next layer, we will use **backpropagation** to update our parameters.

The use of LSTM to preserve the old information to predict sales in the further future. To get a better result, I used **Convolution Neural Network** (CNN) to collect surrounding data and predict the current situation. In a convolutional neural network, the kernel is a filter which can extract the features from the input. In 1D CNN, the kernel is a filter array that moves over the input data, performs the dot product with the sub-region of input data, and gets the dot products' output. In 1D CNN, the kernel moves in one direction. Kernel Input and output data of 1D CNN is two dimensional, and they are mostly used in Time-Series data.

V. Ensemble Models Implementation

Python 3.7 is the optimal programming language for this project, due to its self-contained machine-learning libraries. Following libraries are used for this project:

- Pandas for dataframe manipulations
- NumPy for efficient arrays operations
- LightGBM for training tree-based boosting models
- Matplotlib for data visualization
- Pytorch for constructing Neural Network models
- Kaggle Datasets for connection with data source.

For the LightGBM, I created multiple features, including time lagging information. And pair each sale with its features to feed the model.

Hyperparameters set as follows:

Learning rate: 0.075, bagging freq:1 and lambda12 to 0.1.

For the ANN model, we feed the model as a fixed period for each item, but we feed all data in each store for NN's high computation capacity. The layers of Neural Network combined LSTM layer and 1D CNN layer. (Hyperparameters in ANN will be shown details in code on GitHub).

Though the single performance of ANN is not so good as LightBGM, the ensemble model combines the advantages of both models and gets a remarkable result on the final leaderboard.

As to the ensemble rate, LightBGM contributed 70 percent, and 30 percent is from ANN model

VI. Results and Discussion

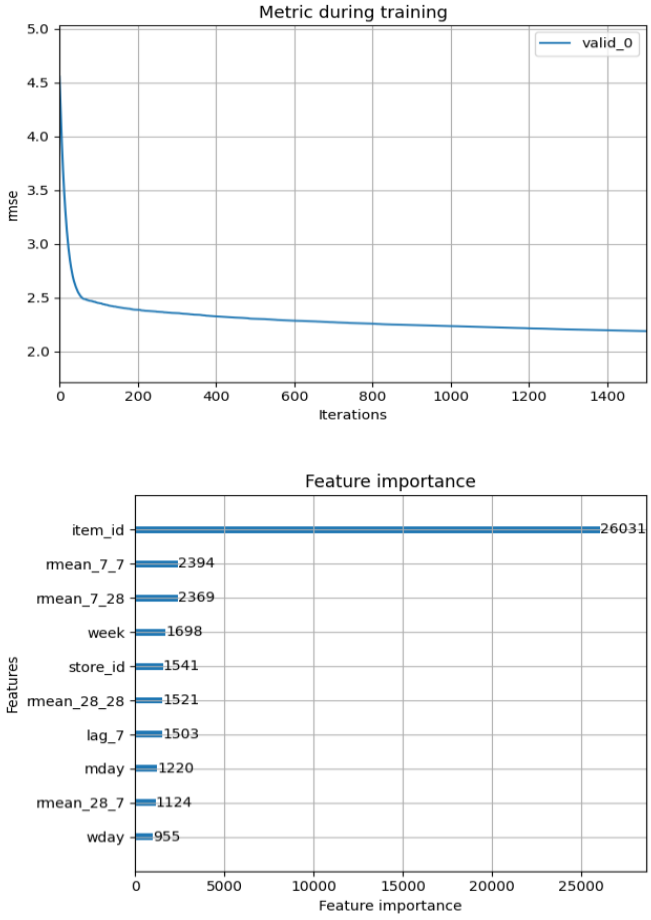
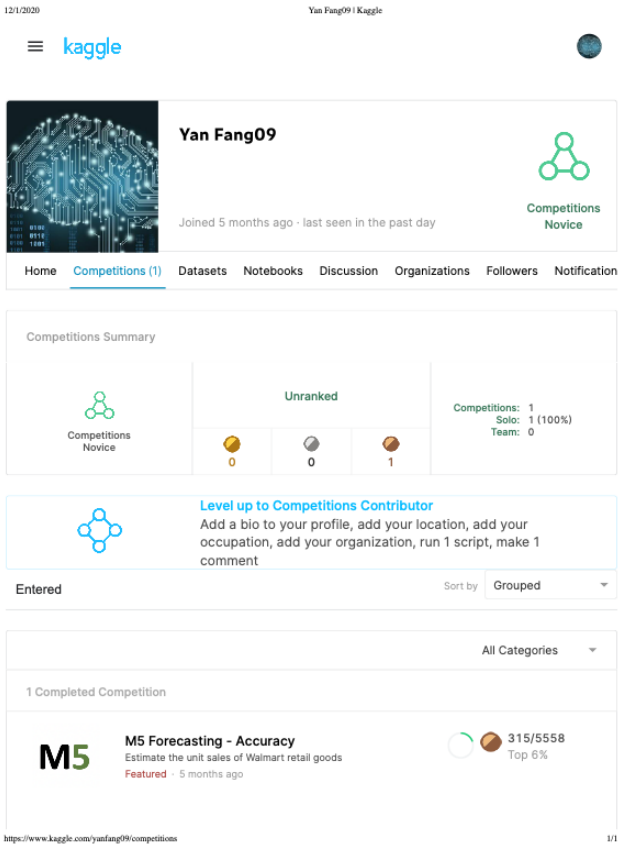


Figure 17. Training Result

Above figures are training results of LightBGM. The first shows training loss function decrease as iterations grows. The second figure illustrates our primary assumptions hold their stance. Through the

results, we can see that except of the item category information, mean lagging of 7- and 28-days cycle influence the sales most. In other word, our custom features act as a significant role in predicting the future sales.



Finally, the score reaches to 0.65099, ranked as 315/5558 in Kaggle's leaderboard.

VII. References

[1] Vasiloudis, T. (2019, August 26). Block-distributed Gradient Boosted Trees. Retrieved November 18, 2020, from http://tvas.me/articles/2019/08/26/Block-Distributed-Gradient-Boosted-Trees.html

[2] Mandot, P. (2017, August 17). What is LightGBM, How to implement it? How to fine tune the parameters? Retrieved November 18, 2020, from https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc

[3] Headsortails. (2020, October 04). Back to (predict) the future - Interactive M5 EDA. Retrieved January 17, 2021, from https://www.kaggle.com/headsortails/back-to-predict-the-future-interactive-m5-eda