Predicting Walmart Sales Price

Project Description

Code

Qingchuan Lyu

June, 2020

This Kaggle project uses hierarchical sales data from Walmart to forecast daily sales for the next 28 days. The data covers stores in California, Texas, and Wisconsin and includes item, department, product categories, and store details between 01/2011-06/2018. In addition, it has explanatory variables such as price, promotions, day of the week, and special events. To reduce the large memory storage, I wrote a helper function to downcast numeric variables to the smallest float or integer dtypes, which decreased memory usage by 46%.

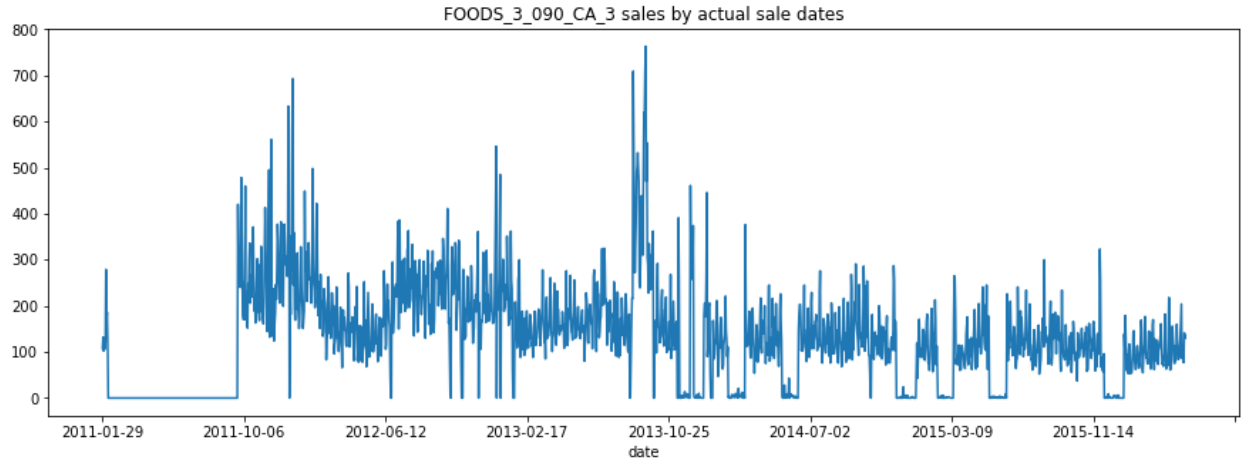

Figure 1. Unit Sales of Food from Department labeled "3" at a store labeled "3" in California 2011-2015

I first used Label-Encoder to transform categorical features to numeric arrays, such as even names. Then, I applied One-Hot-Encoder in order to omit the underlying orders created by Label-Encoder. Then, I added event/holiday information to a sales dataset by merging it with a calendar dataset with a sales dataset on the counting variable of days. To omit the noise from old data, I restricted the merged data to the most recent two years.

As zero demands were very popular (>50%) in the sales dataset, I created new features with demands and prices by generating summary statistics (such as mean, std and max) at the level of store id and item id combined, according to rolling windows of sizes 28, 28*2 and 28*3. Note that 28 is the target window size in this project. Surely, I could explore data more by changing summary statistics or window sizes (yearly, monthly), if I had more time.

To facilitate modeling scripts, I extracted date and time attributes, such as year and month. I also identified weekends by using the variable "day of week," as weekend demands are expected to be higher in general. As the dataset does not contain all the days (only sales of 1941 days were recorded in total), I re-counted day numbers with variables "month" and "day," assuming a month has 30 days.

Then, I defined a time series splitter by hand, as the built-in TimeSeriesSplit from the sklearn library only has two parameters: the number of splits and the maximum size of a single training set--I tailored it to specify the number of days to predict (28 days), control the number of days to train each folder (365 days in each of five folders), and extract indices of rows falling in reasonable date ranges (Real data was messy. Adding 365 days naively didn't necessarily return dates in the next year). This splitter came in handy when I used LightGBM, a fast gradient boosting framework based on decision trees with low memory usage, to complete this project. With the real data without a well-defined assumption, I used Poisson Regression as the objective function by experimentation. On the public leaderboard, this model achieved 0.46 Weighted Root Mean Squared Scaled Error with the unseen data.