Walmart Hierarchical Time-Series Sales Forcasting

Shaun Niemeyer

Fall 2022

Introduction

For this project we utilized historical sales data from a Kaggle competition for 45 Walmart stores located in different regions. Each store contained many departments. The goal was to predict the future weekly sales for each department in each store based on the historical data.

Data

The initial data set, train_ini.csv, provided the weekly sales data for February 2010 to February 2011. Based upon this data, we needed to predict the weekly sales for 2011-03 and 2011-04. After performing a prediction on this data set, the actual sales for 2011-03 and 2011-04 data were added to the training set, and we were then required to predict weekly sales for the next two months (2011-05 and 2011-06). This process was repeated on each subsequent set of two-month blocks of data for a total of 10 test folds. Accuracy on the 10 test folds was evaluated based upon WMAE, which weighted holiday weeks at 5 times the weight of non-holiday weeks.

Strategy

Our approach was to create more than one model and average the predictions over these models; specifically, we created a linear regression model based on the code provided by Prof. Liang, as well as a boosted trees (XGBoost) model. The combined performance of the two models exceeded the performance of either model individually. The theory behind this approach was that if we could produce two models whose errors were uncorrelated, the average of the two predictions would better distinguish the signal from the noise in the data. XGBoost was chosen to complement the linear model, as it uses a very different approach to prediction and therefore was expected to produce uncorrelated errors.

Pre-Processing

Different pre-processing steps were used for each of the two models.

For the XGBoost model, we created integer Year, Week and Month variables. In this case, the week was the corresponding week of the year, 1 - 52. In order to ensure that the weeks from each year aligned (so that the holiday weeks of one year matched those of the other years), it was necessary to subtract 1 from the weeks in 2010. This simple transformation worked very well for the XGBoost model.

Another feature engineering technique that was key to improving the performance of the XGBoost model was to supply the algorithm with a “weights” vector. As the WMAE scoring algorithm weighted holidays at 5 times the weight of other weeks, we assigned a weight to each week based upon whether or not it was a holiday week; non-holiday weeks received a weight of 1, while holiday weeks received a weight of 5.

However, rather than using these weights as predictors in the model, we instead made use of XGBoost's ability to attribute greater weight to errors on certain observations. While the boosting algorithm naturally assigns more weight to observations on which it produced errors on the previous iteration, the user is also able to supply the algorithm with a fixed set of weights that overlay the weights computed on the basis of prior errors. These are passed in as the `weight` parameter of the `xgb.DMatrix` training matrix. So, in this respect, the XGBoost algorithm is well-suited to this specific predictive task, as it allows us to align the objective function of the algorithm with the scoring function used to evaluate our model's performance (which XGBoost would otherwise be unaware of).

For the linear regression model, we employed several pre-processing steps. The first step of pre-processing was to apply SVD to the training data in order to obtain a reduced rank version of the original data, selecting only the top 8 components. The implementation of SVD was based upon the Kaggle competition winner's `preprocess.svd` function (https://github.com/davidthaler/Walmart_competition_code), but modified to process each department individually. In other words, we identified the distinct set of departments in the training data and iteratively looped through each department-specific set of observations, applying SVD to those observations and appending to our result set. If the number of selected components was less than the number of rows (i.e the number of stores), then we excluded the department. Before applying SVD, we computed the mean of `Weekly_Sales` for each store/department combination, and then subtracted these means from each week. The means were added back to the data after applying SVD. Also, prior to SVD, the values for any weeks with `NA` sales dollars were replaced with zeros.

After obtaining our reduced rank version of the data, integer "Wk" (week) and "Yr" (year) variables were created, with "Wk" representing each week in the year 1 - 52 (or 53). Weeks in 2010 were then adjusted by subtracting 1 from each week, so that the weeks from each year were aligned. We then translated each of the weeks into dummy variables, so that there was a column for each week, and the value for each column was `0` or `1`. Additionally, we also added the quadratic term of `Yr` to the model. This left us with a dataset with the variables `Store`, `Dept`, `Yr`, `I(Yr^2)`, `Wk1` through `Wk52`, and `Weekly_Sales`. The test data was transformed in the same way, but without the application of SVD.

Implementation

For the XGBoost model, after pre-processing we had a dataset with the variables `Store`, `Dept`, `IsHoliday`, `Year`, `Week`, and `Month`, as well as a vector of weights for each observation, as previously discussed. An `xgb.DMatrix` was created using these variables and the weights were passed to the `weight` parameter. The optimal parameters for XGBoost were found to be:

- `nrounds = 200`
- `max_depth = 20`
- `eta = 0.02`
- `min_child_weight = 4`
- `eval_metric = "mae"`

So we used a fairly small number of iterations (`nrounds = 200`), but created very deep trees (`max_depth = 20`, vs. the default depth of `6`) with a small learning rate (`eta = 0.02` vs. the default of `0.3`) and a large minimum leaf node size (`min_child_weight = 4`, vs. the default of `1`). The large tree size allowed each tree to provide a low-bias estimate (utilizing many different splits to zero in on the signal), while the slow learning rate and large leaf node size mitigated overfitting (reducing variance). The weights provided to XGBoost allowed us to minimize errors on the holidays, so that, together with the appropriate evaluation metric (MAE), the xgboost algorithm was aligned with the evaluation methodology (WMAE).

We then ran XGBoost on all training samples of the current fold, rather than attempting to split the data into chunks for each specific store or department. This proved to be both more efficient and more accurate in the case of XGBoost.

For the linear regression model, we took a different approach, essentially creating a separate model for each distinct store/department combination. We first determined the unique set of stores and departments that appeared in both the training and test sets of the current fold. Then, within a `for` loop, we extracted the training and test observations for the current store and department. The last step before fitting a model was to account for the bulge in sales around Christmas in fold 5 of the test set. We accomplished this using the "shift trick" described by Prof. Liang on Campuswire. However, unlike the implementation posted on Campuswire, we applied the shift to weeks 48 through 52. Specifically, for weeks 48 through 51, we decreased the Week's sales by 1/7th, and then added 1/7 of the following week's sales. Then for week 52, we just decreased the week's sales by 1/7th.

After shifting the data, we ran basic linear regression (using `lm.fit`) on the training set for the batch of observations associated with the current store and department. So the dataset used for regression only included dummy variables for each week `Wk1` to `Wk52`, `Yr`, `I(Yr^2)` and (in the case of the training set) `Weekly_Sales`. The coefficients of the linear model were obtained from `lm.fit` and then the non-intercept coefficients were matrix-multiplied by the design matrix. The intercept coefficient was added to this data set to produce the predictions for the current department. The predictions from each store/department-specific model were then aggregated to produce our final set of predictions.

Finally, after computing predictions on the test set from both XGBoost and the linear model, we **combined both sets of predictions** into a single `data.frame` and then took the average of each store/department/week combination. These averaged predictions were then returned by `mypredict()`.

Results

The accuracy of our model on the test data, measured as the average of the WMAE over the 10 folds, was $1,453$, well below the benchmark of $1,580$.

Fold	WMAE
1	1,711.535
2	1,273.624
3	1,299.542
4	1,378.306
5	2,016.839
6	1,535.286
7	1,543.518
8	1,263.348
9	1,253.099
10	1,253.509
Overall Average	1,452.861

The running time of our code was $1,783$ seconds, i.e. just under 30 minutes. XGBoost accounted for the vast majority of the run time.

This performance was achieved using the following system specifications:

- Dell Latitude 5520 Laptop
- Processor: 11th Gen Intel Core i7-1185G7 @ 3.00GHz 1.80 GHz
- Installed RAM: 16.0 GB (15.7 GB usable)

- System Type: 64-bit operating system, x64-based processor
- Operating System: Windows 10 Pro, Build 19042.1706