Project 2: Walmart Store Sales Forecasting

CS598: Practical Statistical Learning

Naomi Bhagat - nbhagat3, Michael Miller - msmille3, Joe May - jemay3

12 November 2023

Assignment Data

Program: MCS-DS
Assignment post: campuswire
Team contributions:

Person	Contribution
Naomi Bhagat	Report
Michael Miller	Algorithm
Joe May	Algorithm

Overview

Given historical sales data from 45 Walmart stores spread across different regions, our task was to predict the future weekly sales for every department in each store. The dataset is from https://www.kaggle.com/c/walmart-recruiting-store-sales-forecasting.

The measure we use for evaluation is the weighted mean absolute error (WMAE) between our predictions and the actual values of the weekly sales. In our case, holiday weeks are weighted higher than non-holiday weeks by a factor of 5.

Section 1: Technical Details

Pre-Processing

Our function `process_fold` is the meat of the prediction algorithm. We first read the training and test data in the given file directory parameter `file_dir`.

We clean the test data by introducing a new variable `Wk` to numerically represent each week of the year, ranging from 1-52. We do the same thing with a new variable `Yr` to represent the year of the already existing `Date` in the test dataset.

We then initialize an output matrix, a variable for counting, and a variable to keep track of the number of departments in the data. In general, our strategy is to only train models for combinations of stores and departments that exist in both the training data and the test data. Department-stores not present in the test data will never be used, and department-stores not present in the training data cannot be predicted since there is no data. When we are unable to make a prediction, we simply guess 0 as the weekly sales. We use `dept_to_eval` to track which departments exist in both datasets.

From here, we perform the same steps for each department in `dept_to_eval`. Our first call is to a custom function called `spread_df`, which spreads the data from a "tall" format, where each weekly sales value is on its own row, to a "wide" format. Our wide format matrix `X` has each store as a column and each date as a row, with weekly sales as the values.

Once the data is in wide format, we are able to smooth it. There is significant variance in the dimensions of `X` since there are varying numbers of stores and departments in the dataset. When `X` is sufficiently large (both dimensions greater than 8), we preform SVD smoothing. SVD process keeps dimensions the same

but reduces noise and populates values that would otherwise be `NA`. SVD is implemented with standard methodology. If SVD is not required, we continue the algorithm using `X` as is.

After smoothing, we call another helper function called `gather_mat`. This function is essentially the opposite of `spread_df` above. It takes the wide format `X` and converts it back into a tall format dataframe. `gather_mat` also re-applies the metadata that `spread_df` stripped.

Now, for each unique store associated with the current department, we filter the newly smoothed training data and the testing data by the current store, and convert this data to a design matrix to be used for model training. To train the model, we use the linear model `lm`, the `Weekly_Sales` from the filtered training data, and the design matrix created from the filtered trainign data. The coefficients are extracted from the model, and any `NA` values are zeroed out.

Prediction

For each department-store combination, we train a linear model using weekly sales as the response, and `Yr`, `Wk` and `Yr^2` as features. We set any coefficients with value `NA` to 0 (for example in the case where the train dataset contains only one year). We use this model to predict weekly sales for each week in the test dataset.

Post-Processing

To handle the edge case in fold 5 - Christmas falling earlier in the week - we pivot some sales, removing the "too high" value from its current week and shifting it to be counted in the week after. We can then join the results for the current department-store combination into the output dataframe with all the `Weekly_Pred` values stored in the same place. Finally, we write the predictions to `mypred.csv`.

Evaluation

For evaluation, we use the provided function `myeval` on each generated `mypred.csv` file. The function calculates the WMAE per fold. For each fold, the prediction values are read, at which point the `Weekly_Sales` and `Weekly_Pred` are read. Then, weights are added to holiday datapoints, and finally, WMAE is calculated with the formula `sum(weights * abs(Weekly_Sales - Weekly_Pred)) / sum(weights)`.

Section 2: Performance Metrics

The computer system this was run on was a Dell Inspiron 3501, 1.00 GHz with 8.00 GB of installed RAM for all 10 training/test splits. Our implementation loops over both departments and stores, so it is not necessary optimized for speed. We chose to prioritize code readability and function over performance, and our run times were deemed "good enough". Below is a summary of the fold #, the WMAE for each fold, and the time it took to train the models:

Fold	WMAE	Time (s)
1	1941.586	69.40
2	1363.507	67.85
3	1382.469	69.29
4	1527.293	80.14
5	2156.606	80.31
6	1634.892	78.32
7	1613.908	70.70
8	1355.016	68.13
9	1336.911	71.56
10	1334.010	73.67

The average WMAE over all folds is 1564.62.