Project 2 Report

CS 598 Practical Statistical Learning, Fall 2023

Team Members

- Kurt Tuohy (ktuohy): SVD implementation
- Neal Ryan (nealpr2): most data preprocessing, OLS implementation
- Alelign Faris (faris2): mymain.py

Technical Details

We followed almost exactly the approach the instructors took in Campuswire posts 363, 364 and 366. The main processing is done exactly the same as in the efficient implementation of Approach III in post 363; we added a by-department SVD step as per post 364. We gained insight on how to implement SVD from 521.

Data Preprocessing

Like in the efficient implementation of Approach III, we only process store + department combinations that are in both the training and test datasets.

SVD is implemented on the target variable (weekly sales) of the training set to perform some denoising (To be explicit: this matrix is NOT used at inference, only to train the OLS model with the regular training features). The number of components used for SVD is 8.

For each department, we pivot the sales data so that stores are rows and dates are columns. Missing values are converted to zero. We then perform SVD on this matrix.

If SVD returns more than 8 components, we drop all but the first 8 and use the reduced SVD matrices to construct smoothed sales figures. The result has the same number of rows (stores) and columns (dates) as the original pivoted matrix.

If SVD returns 8 or fewer components, we use the original sales figures.

Finally, we unpivot the sales matrix back to its original form: sales figures by store, department and date. All departments are concatenated into a single dataset.

Next, the Date feature is converted to a Wk and Yr. We one-hot encode Wk to generate 52 features, while Yr is left as an integer. Yr^2 is also added.

Models Used

Per the Campuswire posts, we use OLS (ordinary least squares). Specifically, we use the statsmodels.api OLS model from Approach III in Campuswire post 363.

Data Postprocessing

Post inference, any negative predictions are replaced with a $0. The justification for this is that after looking through the model's predictions, some of the weeks have fairly large negative values (-$500). As we are predicting weekly sales numbers, we felt this was out of line with the training data (quantitatively; where we rarely see negatives, and when we do, they're very small values) and our expectation for stores in general

(qualitatively; all sales should generate value, so we would have to take a lot of returns or losses to get this negative). This is the only deviation from the Campuswire documentation.

We can see this postprocessing is particularly helpful in fold 5, which is easily our worst performance. To be conservative, we did not explore the potential gain of going with positive predictions, but there is likely further optimization possible here.

Performance Metrics

Weighted mean absolute error (WMAE) and execution time

Fold	WMAE	Run time (seconds)
1	1943.432	48.695982694625854
2	1360.761	51.95070695877075
3	1381.853	52.14070701599121
4	1526.242	52.82891488075256
5	2316.271	53.67688059806824
6	1630.953	53.128939390182495
7	1611.095	53.17904257774353
8	1353.463	54.27442383766174
9	1335.697	55.05745339393616
10	1332.011	55.27997374534607
Average	**1579.178**	**53.0213025093078**

Computer Specs

- CPU: AMD Ryzen 7 5700G
- Cores/threads: 8/16
- Clock/boost speed: 3.8/4.6GHz
- Cache Size: L1: 512 KB, L2: 4.0 MB, L3: 16 MB
- 32Gb Memory
- GPU not used