Time-Series Forecasting of Weekly Retail Sales using SVD Smoothing and Linear Regressiong

Daniela Nieto Prada

Introduction

Retail sales forecasting is complicated by irregular seasonal patterns, limited historical observations for major events, and varying behavior across stores and departments. The Walmart Recruiting — Store Sales Forecasting dataset provides weekly sales for 45 stores and multiple departments, along with selected holiday markdowns. Although originally released for a Kaggle recruiting competition, the dataset is used here solely for analysis and can be obtained from the Walmart Recruiting — Store Sales Forecasting competition on Kaggle.

In the competition, predictions were evaluated using the Weighted Mean Absolute Error (WMAE), where holiday weeks received higher weight, emphasizing the need to model holiday effects accurately.The WMAE is defined as:

$$\text{WMAE} = \frac{\sum_{i=1}^{n} w_i \, |y_i - \hat{y}_i|}{\sum_{i=1}^{n} w_i}$$

where

- n is the number of rows
- $\hat{y}_i$ is the predicted sales
- y_i is the actual sales
- w_i are weights. w = 5 if the week is a holiday week, 1 otherwise

In this report, a forecasting pipeline was developed to produce store-level weekly sales predictions. Singular Value Decomposition (SVD) smoothing was applied to denoise the sales time series and extract more stable underlying patterns for each department. On the smoothed data, linear regression models were fit using encoded temporal features such as week-of-year and year. Postprocessing adjustments were incorporated to account for known holiday shifts and align with competition-style evaluation.

The goal of this analysis is to present a clear, reproducible workflow in which denoising, feature engineering, modeling, and evaluation are integrated to produce stable and interpretable forecasts with competition-level accuracy. The repository contains the full implementation in R, including data preprocessing, model training, and prediction generation, providing a complete and reproducible pipeline for weekly sales forecasting.

Technical Details

Data Preprocessing

1. SVD Smoothing

To capture consistent trends in weekly sales for each department across stores, Singular Value Decomposition (SVD) was applied separately to each unique department. The training dataset was loaded into the *train* dataframe, and unique departments were identified. For each department, the data were filtered and stored in *train_dept_ts*, then rearranged into a matrix representing weekly sales for each store across dates. The *Store* and *Date* variables were removed from the matrix to facilitate clean matrix transformations but were retained for later reintegration of the smoothed data.

The sales matrix was centered, and the column means were stored for reconstruction. SVD was then applied to the centered matrix.Instead of fixing the number of components in advance, the number of retained components d was determined using an energy threshold criterion. Specifically, d was chosen as the smallest integer satisfying

$$\frac{\sum_{i=1}^{d} \sigma_i^2}{\sum_{i=1}^{r} \sigma_i^2} \geq \tau,$$

where τ represents the predefined energy threshold (e.g., $\tau = 0.95$). This ensures that the reconstructed matrix preserves a specified proportion of the total variance while removing lower-energy noise components.

Using the selected components, the smoothed weekly sales matrix was reconstructed and the column means were added back to restore the original scale. The reconstructed data were then reshaped into long format and merged back into the train dataframe as *Weekly_Sales_Smooth*.

2. Date Encoding

The *Date* variable was transformed into two categorical features, *Wk* (week of year) and *Yr* (year), to capture seasonal and temporal variation in weekly sales. The ISO week number was extracted and stored as *Wk*. Because holiday weeks in 2010 are shifted relative to later years, one week was subtracted from all 2010 observations to align holiday effects across years. The resulting week values were then encoded as factors with levels 1–52, ensuring that week-specific patterns were treated as discrete categories rather than continuous values.

Similarly, the calendar year was extracted and encoded as *Yr* using predefined factor levels *yr_levels*. For test data containing years not present in the training set, these years were mapped to the maximum training year, allowing the model to generate consistent predictions even when no direct historical data were available. This encoding preserves seasonal structure and inter-year differences without imposing linear relationships in time.

Model Implementation

Design Matrix Construction

For both the training and test datasets, design matrices for each store were constructed using the features *Weekly_Sales_Smooth*, *Dept*, *Wk*, and *Yr* for model fitting. Categorical variables (*Wk* and *Yr*) were expanded into dummy-encoded columns.

Linear Regression Model

A linear regression model was fitted separately for each *Store* using the `lm.fit` function. This low-level linear modeling function was chosen for efficiency and for its ability to operate directly on the constructed design matrices. To ensure model stability, any coefficients returned as NA were replaced with zeros. Test-set predictions were computed through matrix multiplication using the estimated coefficients and the corresponding test-set design matrix.

Data postprocessing

After generating weekly predictions, an additional postprocessing step is applied for rows flagged by the *Shift* indicator. This step implements the Christmas shift correction used by the original Kaggle competition winner.

1. Identifying Weeks Requiring Adjustment

The affected predictions are temporarily reshaped to isolate weeks 48–52, since these weeks are influenced by holiday calendar misalignment in the dataset. Only rows where all five week values are present are considered for adjustment.

2. Applying the Christmas Shift Logic

Following the original winning solution, predictions for weeks 48–52 are recalibrated using two adjustments:

- All predictions for weeks 48–52 are multiplied by 12/14.
- Each week (49–52) receives an additional 2/14 of the previous week's predicted sales.
- Week 48 receives 2/14 of week 52's prediction to maintain circular consistency.

This produces a corrected block of adjusted weekly values for weeks 48–52.

Model Training

Load Libraries

```
#install.packages("dplyr")
#install.packages("lubridate")
#install.packages("tidyr")
#install.packages("ggplot2")
#install.packages("gridExtra")
library(dplyr)
library(lubridate)
library(tidyr)
library(ggplot2)
library(gridExtra)
```

Function to Preprocess the Data

```
smooth_department <- function(df_dep, energy_threshold = 0.95) {
  # Convert long format (Store-Date) into wide matrix:
  # rows = stores, columns = dates, values = Weekly_Sales
  wide = df_dep %>%
    select(Store, Date, Weekly_Sales) %>%
    pivot_wider(names_from = Date,
                values_from = Weekly_Sales,
                values_fill = 0)
  # Remove Store column to obtain numeric matrix for SVD
  X = wide %>% select(-Store)

  # Center data
  X_scaled = scale(X, center = TRUE, scale = FALSE)
  xmean = attr(X_scaled, "scaled:center")

  # Compute SVD
  svd_fit = svd(X_scaled)
  sv = svd_fit$d
  # If no usable signal, return original sales without smoothing
  if (length(sv) == 0 || all(abs(sv) < 1e-12)) {
    return(df_dep %>%
        transmute(Store, Date, Dept, Weekly_Sales_Smooth = Weekly_Sales))
  }

  # Select smallest rank d such that cumulative explained energy reaches
  # the desired threshold
  d = max(2, which(cumsum(sv^2) / sum(sv^2) >= energy_threshold)[1])

  # Reconstruct matrix using first d singular components
  # Then add back the column means
  X_smooth = svd_fit$u[, 1:d, drop = FALSE] %*%
              (t(svd_fit$v[, 1:d, drop = FALSE]) * sv[1:d]) +
              matrix(rep(xmean, each = nrow(X)), nrow = nrow(X))

  # Convert back to long format and attach Dept information
  as_tibble(X_smooth) %>%
    setNames(colnames(X)) %>%
    mutate(Store = wide$Store) %>%
    pivot_longer(-Store,
                 names_to = "Date",
                 values_to = "Weekly_Sales_Smooth") %>%
    mutate(Dept = unique(df_dep$Dept))
}
```

```
preprocess_data = function(train_path, test_path) {
  # Load data
  train = read.csv(train_path)
  test  = read.csv(test_path)

  # Smooth weekly sales for each department
  smooth_all = train %>%
```

```
    group_split(Dept) %>%
    lapply(smooth_department) %>%
    bind_rows()
  train = train %>%
    left_join(smooth_all,
              by = c("Store", "Date", "Dept")) %>%
    filter(!is.na(Weekly_Sales_Smooth))

  # Get the unique dates from the training data to use for the encoding
  yr_levels = sort(unique(year(train$Date)))

  # Get the maximum Yr from the training data
  most_recent_yr = max(yr_levels)

  # Transform the date into Wk and Yr
  # Wk and Yr are encoded
  train = train %>%
    mutate(Wk = factor(ifelse(year(Date) == 2010, week(Date) - 1, week(Date)),
                  levels = 1:52),
           Yr = factor(year(Date), levels = yr_levels))
  test = test %>%
    mutate(Wk = factor(ifelse(year(Date) == 2010, week(Date) - 1, week(Date)),
                  levels = 1:52),
           Yr = factor(pmin(year(Date), most_recent_yr),levels = yr_levels))

  # Create model matrices for each store
  train_split = train %>%
    group_split(Store) %>%
    lapply(function(df) {
      X = model.matrix(~ 0 + Dept + Wk + Yr, df)
      as_tibble(X) %>%
        mutate(Weekly_Sales_Smooth = df$Weekly_Sales_Smooth, Store = df$Store,
          Date  = df$Date)
    })
  test_split = test %>%
    group_split(Store) %>%
    lapply(function(df) {
      X = model.matrix(~ 0 + Dept + Wk + Yr, df)
      as_tibble(X) %>%
        mutate(Store = df$Store, Date  = df$Date)
    })

  return(list(train_split = train_split, test_split = test_split, test = test))
}
```

Function to Postprocess the Data

```
postprocess_data = function(combined_pred) {
  # Transform date into Wk
  combined_pred = combined_pred %>%
    mutate(Wk = factor(ifelse(year(Date) == 2010, week(Date) - 1, week(Date)), levels = 1:52))
```

```
  # Creates X matrix with the Weekly_Sales where the columns represents the
  # week and the rows are the different stores/department
  # Select columns for weeks 48:52
  # Filter out rows where their values are NA
  test_dept_ts = combined_pred %>%
    select(Store, Dept, Wk, Weekly_Pred) %>%
    pivot_wider(names_from = Wk, values_from = Weekly_Pred) %>%
    select(Store, Dept, `48`, `49`, `50`, `51`, `52`) %>%
    drop_na(`48`, `49`, `50`, `51`, `52`)

  # Apply shift using the same logic as described by the winner of the Kaggle competition
  shift = (12/14)*test_dept_ts[,3:7]
  shift[,2:5] = shift[,2:5] + (2/14)* test_dept_ts[,3:6]
  shift[,1] = shift[,1] + (2/14)*test_dept_ts[,7]

  # Replace the recalculated weekly sales into the matrix
  test_dept_ts[,3:7] = shift

  # Re-organize the data to its previous format
  test_dept_ts = test_dept_ts %>%
    gather(key = "Wk", value = "Weekly_Pred_Shift", -Store, -Dept)

  # Replace the predicted weekly sales with the adjusted ones
  combined_pred = left_join(combined_pred, test_dept_ts, by = c("Wk", "Store", "Dept")) %>%
    mutate(Weekly_Pred = ifelse(!is.na(Weekly_Pred_Shift), Weekly_Pred_Shift, Weekly_Pred)) %>%
    select(-Wk, -Weekly_Pred_Shift)

  return(combined_pred)
}
```

Function to Fit the Linear Model

```
fit_model = function(train_split, test_split) {
  # Pre-allocate a list to store the predictions for each (Store, Dept) combo
  test_pred = vector("list", length(train_split))

  # Perform regression for each store
  for (i in seq_along(train_split)) {
    # Get the training and test sets for the split
    tmp_train = train_split[[i]]
    tmp_test = test_split[[i]]

    # Create model matrices
    x_train = tmp_train %>%
      select(-Weekly_Sales_Smooth, -Store, -Date) %>%
      as.matrix()
    y_train = tmp_train$Weekly_Sales_Smooth
    x_test = tmp_test[, colnames(x_train), drop = FALSE] %>%
      as.matrix()

    # Obtain coefficients from the linear regression
```

```
    mycoef = lm.fit(x_train, y_train)$coefficients
    # Any NA coefficients are set to 0
    mycoef[is.na(mycoef)] = 0

    # Predict the Weekly Sales of the test data using the coefficients
    y_hat = x_test %*% mycoef

    # Determine if the date needs postprocessing
    to_shift = rowSums(tmp_test[, c("Wk47", "Wk52")]) > 0

    # Store the predictions with required formatting
    test_pred[[i]] = data.frame(
      Store = tmp_test$Store,
      Dept = tmp_test$Dept,
      Date = tmp_test$Date,
      Weekly_Pred = y_hat,
      Shift = to_shift
    )
  }
  # Combine the predictions for all stores
  combined_pred = do.call(rbind, test_pred)

  # Check if the data needs postprocessing
  if (sum(combined_pred$Shift) > 0) {
    combined_pred = postprocess_data(combined_pred)
  }

  # Remove the variable shift from the data
  combined_pred = combined_pred %>% select(-Shift)

  return(combined_pred)
}
```

Function to Calculate WMAE

```
calculate_wmae = function(actuals, preds, isHoliday){
  weights = if_else(isHoliday, 5, 1)
  wmae = sum(weights * abs(actuals - preds)) / sum(weights)
  return(wmae)
}
```

Load and Transform the Data, Train the Models, Make Predictions

```
num_folds = 10
# Create an empty list to store the errors
wmae = rep(0, num_folds)
# Load file with all test results
file_path = 'Data/test_with_label.csv'
test_with_label = read.csv(file_path)
```

```r
# Loop through the folder names and read the corresponding data files
for (idx in 1:num_folds) {
  train_path = file.path(paste0("Data/fold_", idx ) , "train.csv")
  test_path = file.path(paste0("Data/fold_", idx ) , "test.csv")
  pre_data = preprocess_data(train_path, test_path)
  y_hat = fit_model(pre_data$train_split,
                    pre_data$test_split)
  y = pre_data$test %>% select(-IsHoliday) %>%
      left_join(test_with_label, by = c('Date', 'Store', 'Dept'))
  wmae[idx] = calculate_wmae(y$Weekly_Sales, y_hat$Weekly_Pred, y$IsHoliday)
}
```

Analysis and Model Evaluation

Cross-Validation Performance

Model performance was evaluated using WMAE across ten cross-validation folds constructed on the test data. The table below reports the WMAE values for each fold. The average WMAE across folds is 1.4007×10^4, with a standard deviation of 918, indicating relatively low variability in performance given that errors are measured in weekly sales dollars. Individual fold errors range from approximately 1.3312×10^4 to $2{,}1.6508 \times 10^4$, reflecting differences in underlying demand patterns rather than instability in the modeling approach. The relatively small standard deviation across folds suggests that the forecasting pipeline generalizes consistently across subsets of the training data, producing stable predictions despite variability in weekly demand.

Notably, the observed WMAE is lower than that reported by the winning submission in the original competition. This difference is attributable to structural differences in the modeling setup rather than superior predictive performance. The competition setting required broader generalization across combinations and stricter constraints. As a result, the cross-validation results presented here should be interpreted as an upper-bound estimate of achievable accuracy under a more controlled data structure.

Table 1: Weighted Mean Absolute Error for Each Fold

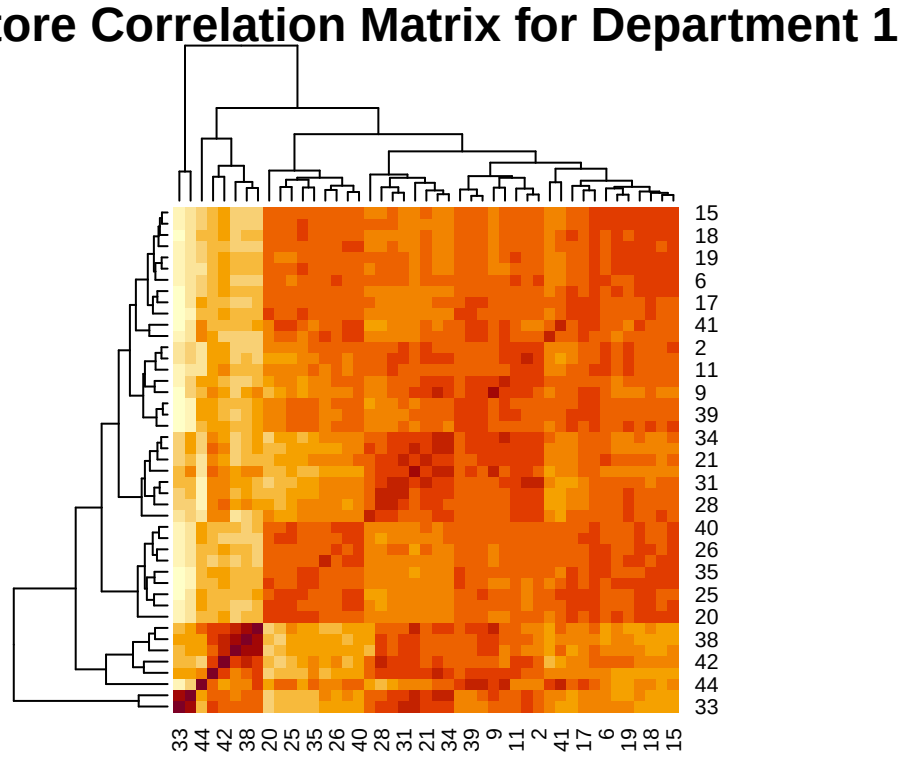
Fold	WMAE
1	13315
2	13720
3	13757
4	13312
5	16508
6	13813
7	13698
8	14138
9	14076
10	13738
Mean	14007

Correlation of Sales Between Stores

Highly synchronized sales dynamics across stores was observed for most departments. For example, for department 1, the mean pairwise correlation between stores is 0.85. Additionally, the correlation heatmap

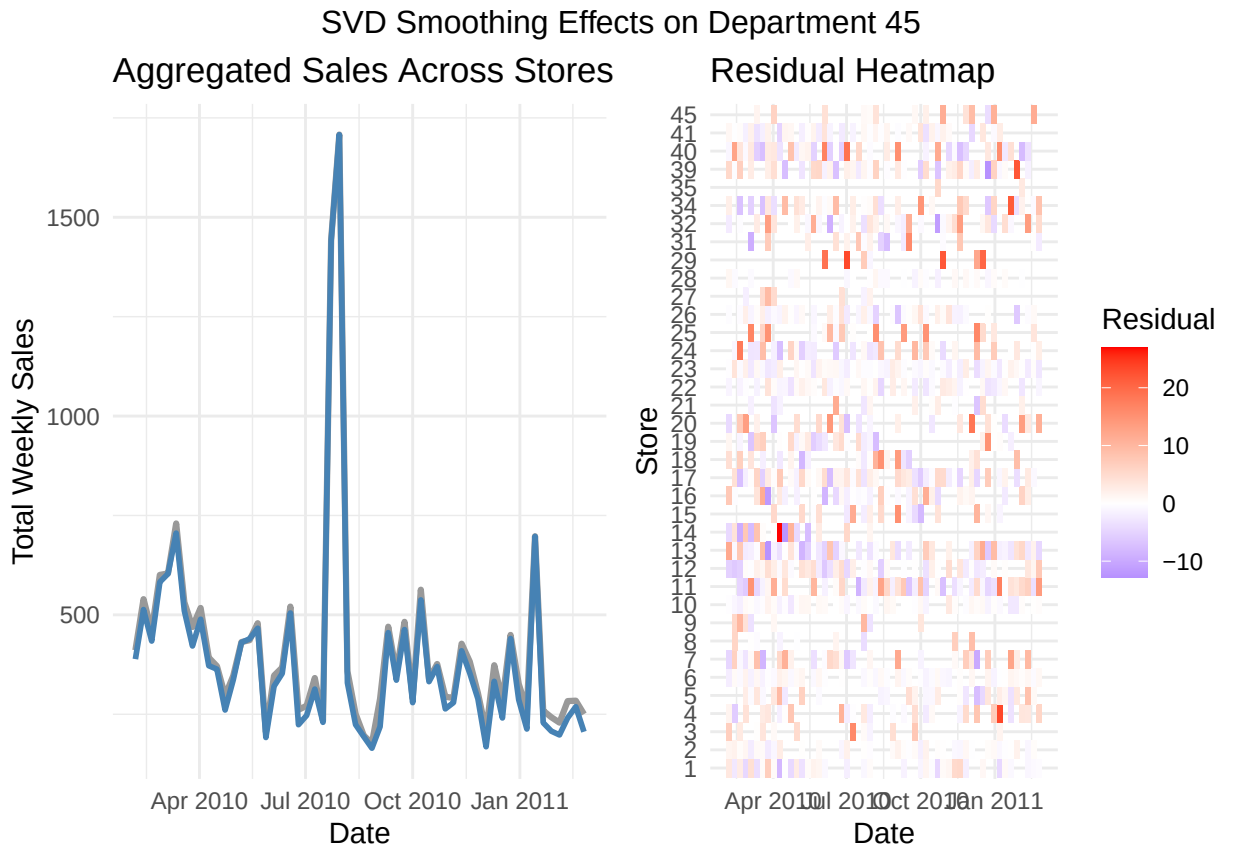
confirms this strong positive dependence, with most store pairs exhibiting consistently high correlations. This suggests the presence of a dominant common demand pattern shared across stores.

Although the first singular value captures most of the variation, a minimum rank of d = 2 was enforced to ensure that secondary structured effects were retained. This prevents over-compression when a single component dominates and allows the model to capture additional systematic variation beyond the primary shared trend.



Impact of SVD Smoothing

Applying SVD smoothing prior to model fitting had a noticeable impact on prediction stability. By denoising weekly sales at the department level, SVD reduces short-term volatility and isolates the dominant sales structure shared across stores. This preprocessing step is especially beneficial for departments or stores with sparse or irregular demand patterns, where raw sales data can be highly noisy. The smoothed sales signals provide a more reliable target for regression modeling, contributing to improved consistency across validation folds.



The variance across stores decreases after SVD smoothing. For example, for department 45, the variance of the weekly sales went from 736 to 734 which is approximately a 3.37% reduction. This change indicates that the SVD reconstruction preserves nearly all structured variation in weekly sales, while removing only a small amount of noise. The result is also consistent with the strong low-rank structure and high inter-store correlation observed in the department.

Conclusions

Overall, the analysis reveals a strong low-rank structure in department-level sales, driven by high inter-store correlation and a dominant shared temporal pattern. SVD smoothing preserves the core demand signal while reducing variance, indicating that primarily noise is removed and most structured variation remains intact.

Additionally, fitting a linear model separately for each store also produced strong performance, reinforcing the conclusion that sales dynamics are highly predictable and largely governed by common trend and seasonal components. Together, these results confirm that both low-rank decomposition and store-level modeling effectively capture the underlying structure of the data.