Forecasting Grocery Store Product Sales

Austin Lackey, Tomy Sabalo Farias, and Sam Herold

DSCI 478, Colorado State University

February 20, 2024

GitHub Repository

Abstract

This paper presents a methodology for forecasting grocery store sales across many stores and product subsets. The data is used in this project from Favorita Grocery Stores in Ecuador from 2013 to 2017 and is provided by Kaggle. A link to the data can be found in the references at the end of this paper. The work entailed individual, predictive models that were narrowed down to our best and final model. In this project, we use a variety of models including linear & polynomial regression, lag series, and gradient-boosting. Our methodology is split into two main sections: the characteristics surrounding the problem and our model solution. We will be centering our write-up around a gradient-boosting model that was most effective.

1 Introduction

1.1 Background

Time series data is data collected and marked by time. Because the sequence of the data is meaningful, there may be trends and patterns in time-series data that are different to traditional data. Time is typically the main predictor variable, but we can cleverly add other features to increase our models accuracy[1].

The Store Sales time series forecasting dataset contains sales data from Favorita Grocery Stores in Ecuador from 2013 to 2017[2]. Sales data is broken down by store and product category for each day (1,782 pair-wise combinations = $stores \times products$).

1.2 Problem Statement

The goal of this project is to forecast sales for each store and product category combination one day into the future while reducing $RMSE$. $RMSE$ is defined as the square root of the

average of the squared differences between the forecasted and actual sales.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

1.3 Data

Variable	Type	Unique Counts	Description
date	datetime64	1684	Date-stamp of the sales data
store_nbr	int64	54	Store number
family	categorical	33	Product family/category (Grocery, Beverages, Deli, etc.)
city	categorical	22	City of the store
state	categorical	16	State of the store
type	categorical	5	Type of store (A, B, C, D, E)
cluster	int64	17	Cluster of the store (Similar stores grouped together)
promotion	int64	2	Product Promotion (0 or 1)
oil_price	float64	decimal	Oil price for the day

Table 1: Variable Descriptions

For each day, we are making predictions for each store and product category combination (1,782 pair-wise combinations). This is a challenging problem since sales are influenced by more than just the seasonality. Below is an example that shows the sales over the course of a three-month period for only the top 5 product categories for the store with the most sales.

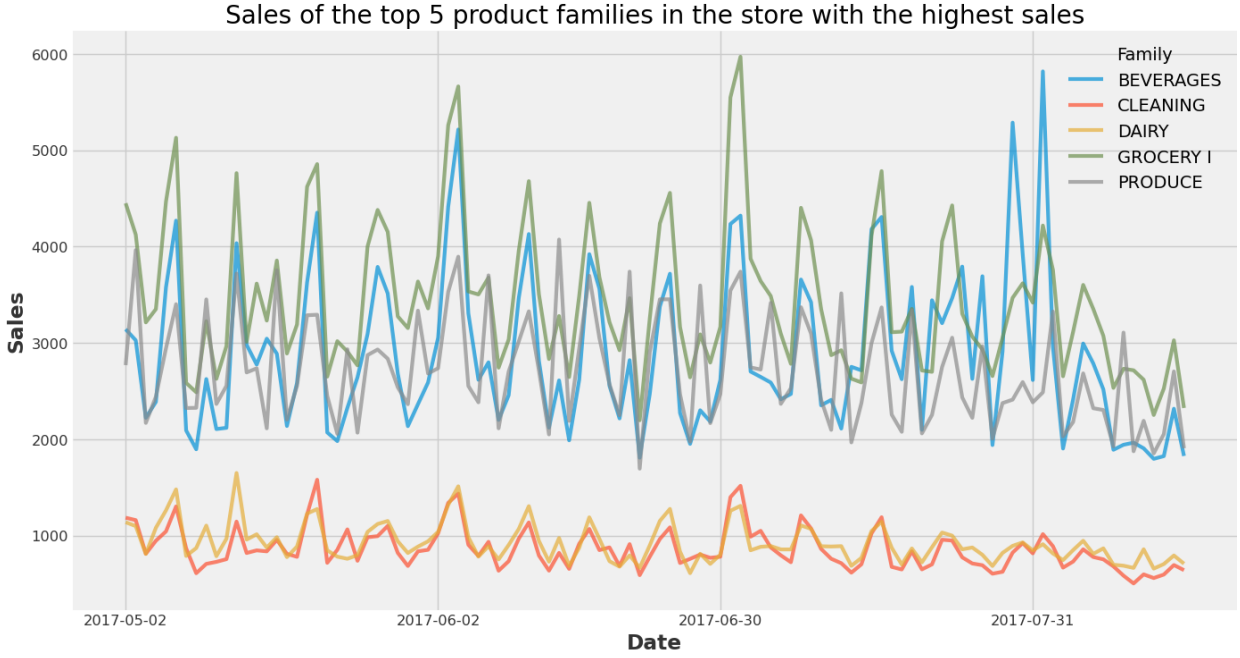


Figure 1: Top 5 Product Categories for Store 44

Date	Store Number	Product Category
2013-01-01	1	Grocery
2013-01-01	1	Beverages
2013-01-01	1	Deli
⋮	⋮	⋮
2013-01-01	54	Grocery
2013-01-01	54	Beverages
2013-01-01	54	Deli
⋮	⋮	⋮
2013-01-02	1	Grocery
⋮	⋮	⋮

Table 2: Example of Pair-Wise Combinations

2 Solution

2.1 Feature Engineering

The biggest challenge in this project was to create a feature set that would allow us to forecast sales for each store and product category. Accurately predicting price based on date alone would is near impossible. It is necessary to add more predictive features. We found that the most important part in reducing $RMSE$ was to create a feature set that captured as much information regarding seasonality and trends as possible. We created a feature set that included the following extra variables:

- **Datetime Features:** We created features that captured what day of the week it was, the day of the month, the month, the quarter, the week of the year and the year. It is clear that the day of the week, end of month, etc. can have some correlation with sales. As you can see in Figure 2 below, there is a clear seasonality in the data.
- **Lag Features:** A lag feature gives the model information about the previous day or days. We implemented a 7-day lag feature with the sales of pervious days. It makes sense that the sales of yesterday or two days ago would have some kind of indication on the sales of today. This is common practice in time-series forecasting. Lag features essentially give the model more context and significantly increases accuracy.
- **Store-Specific Features:** We joined the store data with the sales data to give the model more information about the store. This includes information as specific as which product categories had a promotion that day.
- **Outside Data:** We also joined the sales data with oil prices since Ecuador is heavily dependent on the oil industry[2]. We can see that there is a negative correlation between oil prices and the sum of sales for stores within Ecuador in Figure 3 below.

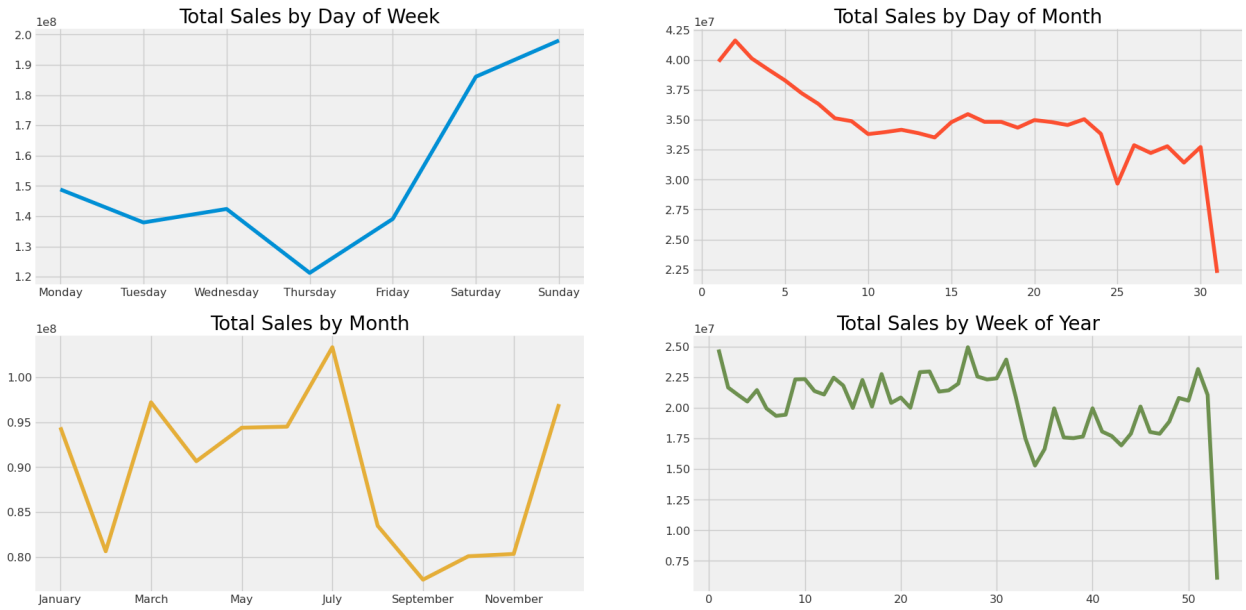


Figure 2: Seasonality

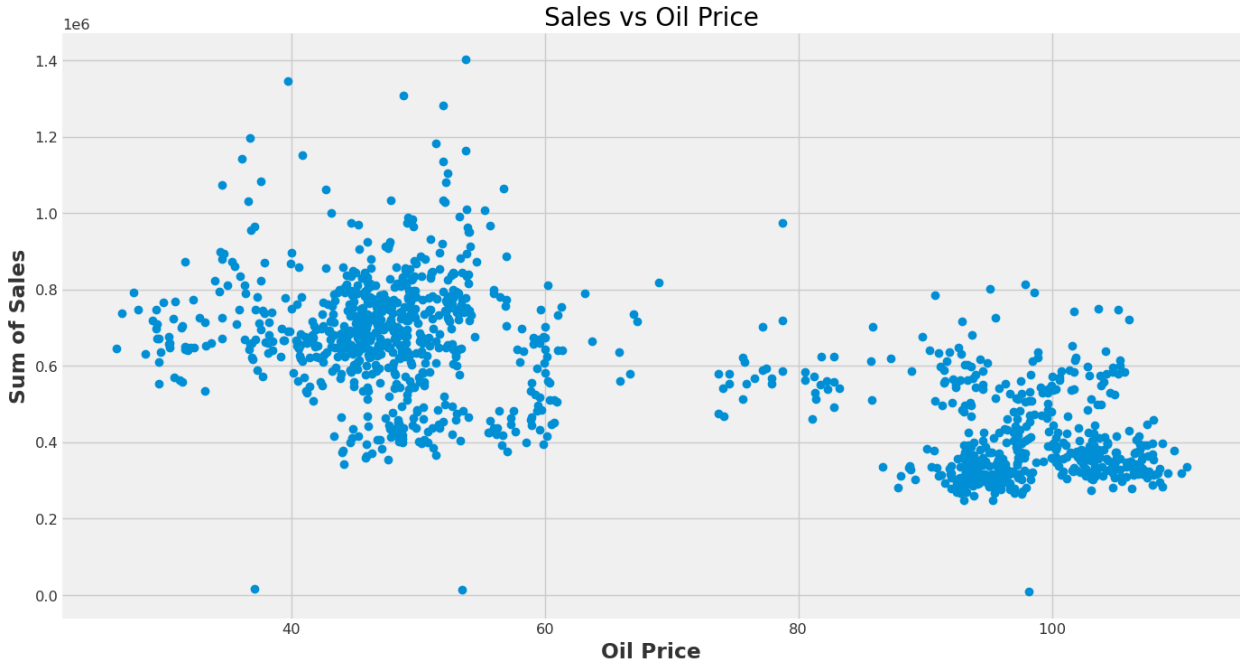


Figure 3: Oil Prices

2.2 Model Engineering

2.2.1 XGBoost Model

Although we tested multiple models including linear and polynomial regression, we found that the best model for this problem was gradient boosting. XGBoost is an open-source library that implements gradient boosting in an optimized way that makes it easy for data scientists to fine-tune their models[3]. The idea is that we are building a sequence of trees

that are trained to minimize a cost function. As mentioned in the previous section, we used $RMSE$ as our cost function, which is a very common metric for regression problems. We set our model up with the following parameters:

Parameter	Value
base_score	0.5
booster	gbtree
n_estimators	1000
objective	reg:squarederror
max_depth	15
learning_rate	0.1

Table 3: Variable Descriptions

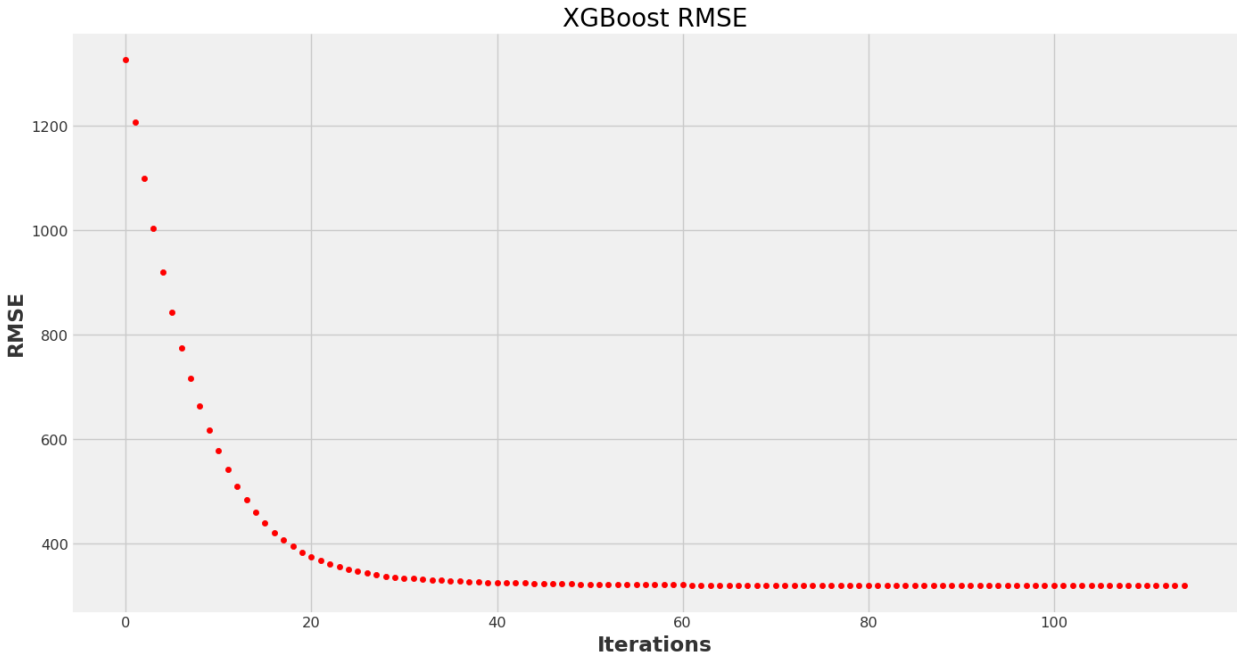
The base score is the initial prediction score for all instances, which is set to 0.5 by default[4]. We used the gradient boosting tree model with 1000 estimators(trees), although later in training a smaller number of estimators was found to be sufficient. We used a max tree depth of 15 and a learning rate of 0.1. Increasing the tree depth helps the model capture more complex patterns, however it can increase the risk of over-fitting[4]. In our case, we found that a tree depth of no more than 15 prevented over-fitting while maximizing the model's ability.

2.2.2 Training the XGBoost Model

After 114 iterations, our $RMSE$ converged to 319, which means that 114 estimators were enough to minimize the cost function. We can see this in the plot and table below, after 100 iterations, the $RMSE$ begins to yield diminishing returns.

Iterations	RMSE
0	1326.11
100	319.14
114	319.10

Table 4: Iterations and RMSE



2.2.3 Evaluating the Predictions

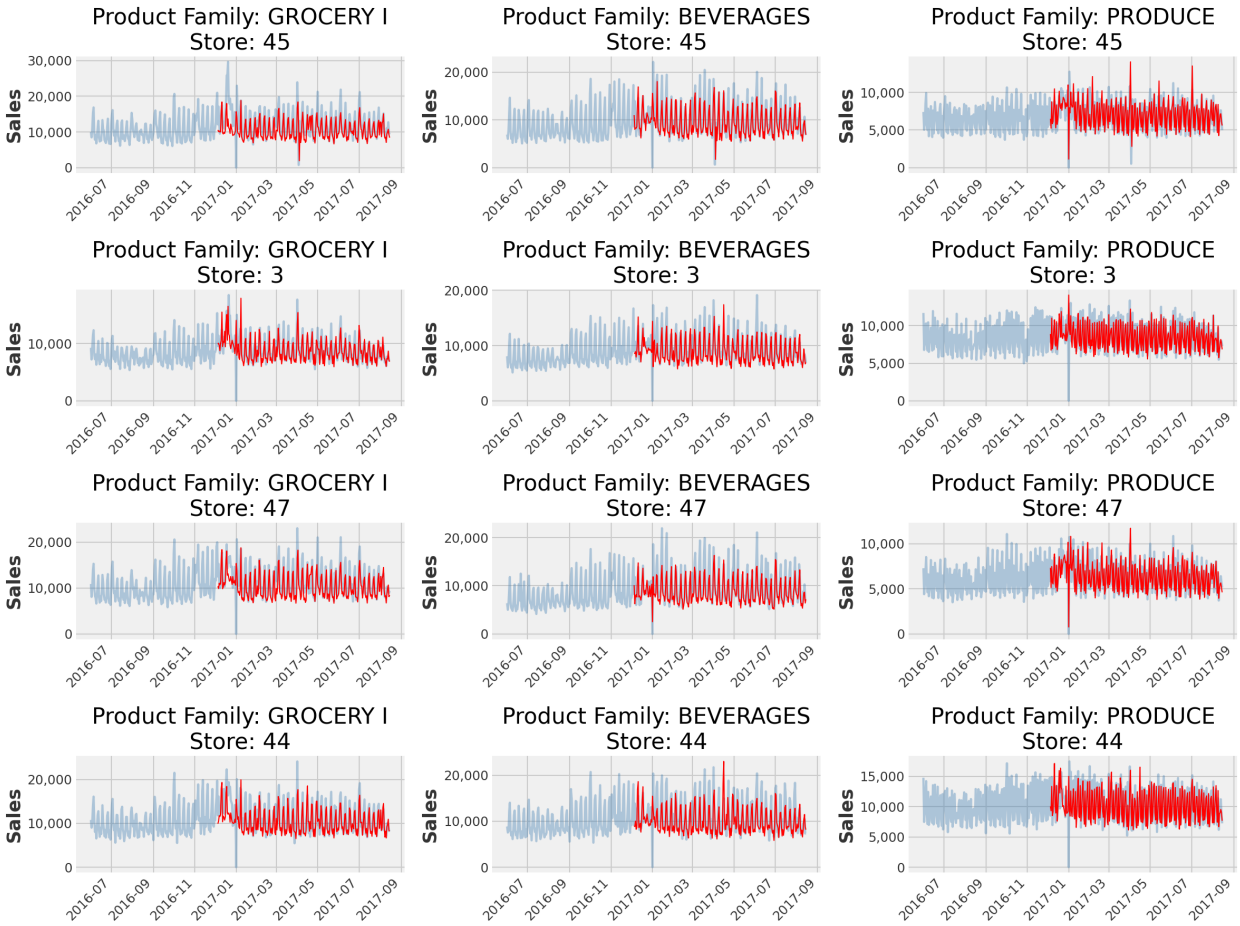


Figure 4: XGBoost Predictions

Since we are making 1,782 pair-wise predictions, let's just take a look at the top 3 product categories for the top 4 stores. More specifically, lets look at how the predictions compare to the actual sales for stores 3, 44, 45, and 47 in the top 3 product categories, Grocery I, Beverages and Produce. Our model does a decent job at capturing the the seasonality and trends in the data. There are certain product categories it does better, but overall the model is able to capture a lot of variation when it comes to different stores and product categories.

3 Conclusion

In this project, we were able to forecast sales for each store and product category using the XGBoost model. We found that the most important part in reducing $RMSE$ was to create a vast feature set that captured as much information regarding seasonality and trends as possible. This meant creating features that were specific to the stores, calendar features, lag and outside data such as oil prices. If we were to improve this model, we would look into capturing holidays and other events that could influence sales. It would also be interesting to see how the model performs with new LSTM models that have been recently developed.

References

[1] Time Series Kaggle Course. Kaggle. Retrieved from `https://www.kaggle.com/learn/time-series`

[2] Favorita Grocery Store Sales - Time Series Forecasting. Kaggle. Retrieved from `https://www.kaggle.com/competitions/store-sales-time-series-forecasting/data`

[3] XGBoost: A Complete Guide to Fine-Tune and Optimize your Model. Towards Data Science. Retrieved from `https://towardsdatascience.com/xgboost-fine-tune-and-optimize-your-model-23d996fab663`

[4] XGBoost Documentation. dmlc. Retrieved from `https://xgboost.readthedocs.io/en/stable/python/python_api.html`