Predict Demand for Bakery Products Using Regression Models

Erin Choi, Vicky Lin (Breadiction)

New York University Center for Data Science Undergraduate Program
Advanced Topics in Data Science: Machine Learning for Climate, Spring 2023

Abstract

Food waste is a large contributor to greenhouse gas emissions worldwide. To reduce emissions from food waste at the retailer and company level, this project applies several regression techniques to build a model that accurately predicts demand for bakery products. We extracted and encoded features from data on Grupo Bimbo bakery products and compared six different regression models' ability to map these predictors to demand values. The XGBoost Regressor model achieved the best prediction performance with the lowest RMSLE, MAE, and MedAE. Our model may serve as a prototype for a food product prediction framework that can be improved with further tuning and more input features, provided the availability of more computing resources.

1. Background

The problem we address in this work is predicting demand for perishables, specifically food, as a way to mitigate climate change. Food is an integral part of our daily lives that is produced in surplus; about one-third of food produced is wasted worldwide [1], and 30 to 40% of food is wasted in the United States [2]. This food waste, including that lost in supply chains and not consumed, makes up 6% of global greenhouse gas emissions [3] [4]. If there were a country consisting of food waste, it would be the third largest producer of carbon dioxide, producing less than only the United States and China [1]. Therefore it is critical to reduce the amount of food wasted worldwide to decrease greenhouse gas emissions that drive climate change and rising global temperatures.

Since it is difficult to prevent individuals' food waste, this project targets the step directly before individuals, which includes food retailers and companies. To reduce food waste, retailers must accurately predict demand for the products they sell. Quantifying demand prevents stockpiling of products at stores and overproduction by companies, which is especially important when selling food products since they have a limited shelf life.

Machine learning methods are widely used in the food industry to monitor product inventory and sales. Past research, including that by A. Garre, M. C. Ruiz, and E. Hontoria [5] and by K. Aishwarya, et al. [6], has produced models or tools to predict product inventory or demand by comparing a variety of regression methods including linear regression, random forest, boosting, and bagging. Previous work by N. Shue, et al. [7] has also used a time-series forecasting approach to predict products' hourly sales by using convolutional neural network (CNN) models, particularly long short-term memory (LSTM) networks. In this project, we use a similar approach to the former works. Several regression methods are compared based on their ability to accurately predict demand for bakery products using product characteristics, client information, and sale locations.

2. Methods

2.1 Data Source and Features

The data used for this project is the Grupo Bimbo Inventory Demand dataset from Kaggle Competitions [8]. Grupo Bimbo is a Mexican multinational food company that sells bakery products. Since the company is based in Mexico, the data feature names and string inputs are provided in Spanish, posing difficulties during the preprocessing step.

The dataset includes information on weekly product sales and returns in Mexico over nine weeks. The original features for each transaction include the week number, the product's ID and full name, the client's ID and name, the sales location's ID, town, state, the number of units sold and returned, and the total sales and returns in Mexican pesos. The target variable, product demand, is defined as the number of units sold during that week subtracted by the number of units returned during the following week.

Because the data covers a short amount of time and does not include dates for the weekly timestamps, we chose a regression model comparison approach over time-series analysis. Each transaction was treated as an individual entry for the regression methods.

2.2 Data Cleaning and Preprocessing

2.2.1 Feature Engineering and Joining Tables

The feature for the week for each transaction was dropped, as it was provided as integers and thus did not contain any helpful information about the actual date of the transaction. The features related to returns were also removed since demand had already been calculated and provided as a data feature.

Three tables were joined to the main dataset: the client, product, and town/state tables. Each table was individually cleaned or processed for feature extraction before being merged with the main data. Figure 1 visualizes each table's original columns and their columns after feature engineering.

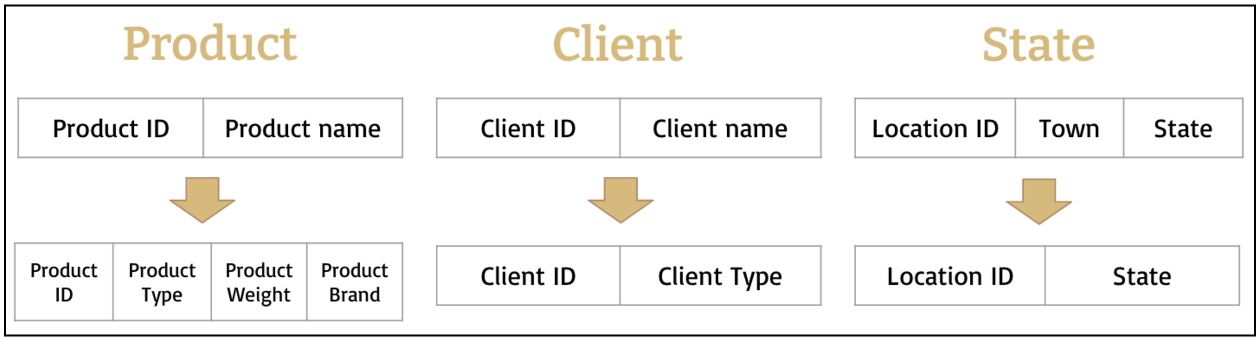

Figure 1. Product, client, and town/state tables' columns before and after feature extraction.

The client table contained a large number of unique client names, so they were categorized into a smaller number of client types. To do this, the text was split on whitespace, then Spanish stopwords and common Spanish names were removed from the list of unique words. Stopwords were obtained from Python's Natural Language Toolkit (NLTK) library, and common Spanish names (male, female, and surnames) came from the National Statistic Institute (Instituto Nacional de Estadistica, INE), with data compiled by J.V. Rama on GitHub [9]. Among the remaining words, specific terms that appeared frequently were used as search terms for forming client categories, including supermarkets, convenience stores, restaurants, and schools. Clients with participles or numbers in their names were labeled generally as "Small franchises," and the remaining clients with identified names were categorized as "individuals."

In the product table, features for the product brand, product weight in grams, and product name were extracted from the full product name. Over 700 unique brands were identified, so all brands with fewer than 9 unique products were labeled "Other." Similar to the process of client categorization, words that appeared frequently in the value counts of product names were used as search terms to group products into a smaller number of categories, such as cookies, cakes, and pastries. The town variable was dropped from the town/state table because we chose to use states as the categories for sale locations.

After preprocessing the individual tables, they were joined with the main data table. One additional feature for product price was created by dividing the sales amount in pesos (Venta_hoy) by the number of units sold (Venta_uni_hoy). Columns that were no longer needed—product, client, and sales location IDs and the sales-related features—were dropped.

2.2.2 Data Downsizing and Handling Missing Values

Due to the large sample size and lack of computational power, one-hot encoding of the original data failed. To proceed, it was necessary to downsize the original data, which contained over 75 million rows of transactions. Any rows with unidentified clients (labeled "NO IDENTIFICADO") or "Other" brands were dropped. These steps did not decrease the dataset size by much, so we aggressively downsized the data by randomly sampling about 250,000 rows of transactions from each of the 33 states. The number of samples came from the number of transactions from the state with the smallest number of rows.

There were some null values in the product weight and price features. We attempted to impute the missing data by performing linear regression predictions between weight and price or by imputing median weight and price values after grouping transactions by product type, but computational limitations prevented these methods from working. The transactions with missing values made up less than 1% of our data, so they were dropped for simplicity. The final data consisted of about 8 million transactions.

2.2.3 Data Splitting, Standardization, and Encoding

The data was split into training and test sets with an 80 to 20 ratio. For each set, we standardized the numerical predictors using RobustScaler and one-hot encoded the categorical features. One column was dropped from each set of binary variables for a predictor to prevent overdetermination of the regression models. There were 107 features in total, including product weight, product price, 52 brand variables, 32 state variables, 11 client type variables, and 10 product type variables.

2.2.4 Evaluation Metric Choice

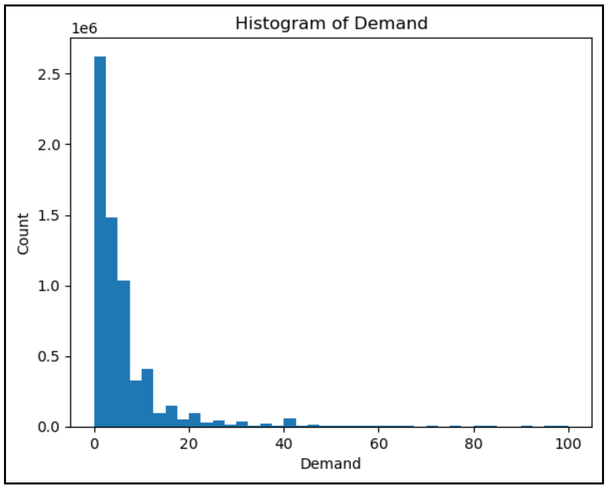

Figure 2. Distribution of the demand values in the training set.

The target variable values in the training set were graphed to understand the distribution of demand. Figure 2 shows that the demand values are part of a non-Gaussian distribution, with many values being close to or equal to 0. Thus root mean squared logistic error (RMSLE) is a better evaluation metric for the data than root mean squared error (RMSE). To be able to compare regression model performance more holistically and relate the results to previous works, the models were also evaluated using mean squared error (MAE), median squared error (MedAE), and mean absolute percentage error (MAPE).

2.3 Model Hyperparameter Tuning

Because we selected a regression approach to the prediction problem, we selected 6 regression methods suitable for mapping the features to demand values. Each model took all features as inputs for fitting. Predictions were made using both sets to detect and prevent overfitting; none of the models exhibited notable changes in performance between training and test predictions. Both time and computational constraints prevented the use of a formalized hyperparameter tuning and cross-validation process such as GridSearch. Instead, hyperparameter tuning was done manually by repeatedly fitting and predicting with new hyperparameters. Specific values were compared for some hyperparameters, with chosen values generally centered around model defaults to observe the general effect of increasing or decreasing hyperparameters on performance.

2.3.1 Linear Regression

As one of the most fundamental classical regression methods, the linear regression model does not require any hyperparameter tuning. The model was fitted once and tested on the training and test sets.

2.3.2 Lasso and Ridge Regression

Both Lasso and Ridge regression models required tuning of the alpha hyperparameter. They were each fitted and tested using four alpha values: 1.0 (default), 10, 0.1, and 0.01. An alpha value of 0.01 resulted in the best performance for both models.

2.3.3 XGBoost Regressor

The XGBoost Regressor from the XGBoost Python package has a large number of hyperparameters that can be tuned. Of these, three were varied for this project. The objective hyperparameter was set to either squared error (default) or squared log error. Max_depth was tested with values of 5, 6 (default), and 7. The learning rate was changed between 0.2, 0.3 (default), 0.4, and 0.5. XGBoost resulted in the best performance when using the default values of the varied hyperparameters: objective = reg:squarederror, max_depth = 6, and learning_rate = 0.3.

2.3.4 Bagging Regressor

The Bagging Regressor also has a large number of tunable hyperparameters, especially since the parameters of the base estimator can also be varied. It took much longer to fit this model than XGBoost, so fewer hyperparameters were tuned using fewer potential values. Using a Decision Tree Regressor as the base estimator, the tree's max_depth was changed between the default of none, 1, 2, and 3. The Bagging Regressor's n_estimators hyperparameter was changed between 10 (default), 50, and 100.

The best performance came from the Bagging Regressor model using n_estimators = 100 and a Decision Tree base estimator with max_depth = 3.

2.3.5 AdaBoost Regressor

The time it took to fit the AdaBoost Regressor was between that for the slow Bagging Regressor and the quicker XGBoost Regressor. Similar to the Bagging Regressor, its base estimator's parameters can also be changed. We tested the Decision Tree estimator with a max_depth of none (default) of 1, a criterion of squared error (default) or poisson, and a min_samples_split of 2 (default), 10, 25, and 50. The AdaBoost Regressor's n_estimators value was selected between 50 (default) and 100.

The best performing AdaBoost model used n_estimators = 100 and a Decision Tree base estimator with max_depth = 1, criterion = squared_error, and min_samples_split = 50.

3. Results

Model	RMSLE	MAE	MedAE	MAPE
Linear Regression	0.896453	6.322293	3.556854	4.580727e+14
Lasso Regression	0.899771	6.353492	3.661482	4.599242e+14
Ridge Regression	0.896465	6.322351	3.556877	4.580800e+14
XGBoost Regressor	**0.716399**	**4.884369**	**2.474169**	**3.752691e+14**
Bagging Regressor	0.931823	6.441883	4.831724	4.706274e+14
AdaBoost Regressor	1.257163	10.046502	8.493673	7.6034193+14

Table 1. The root mean squared logistic error, mean absolute error, median absolute error, and mean absolute percentage error of each model's predictions on the test set.

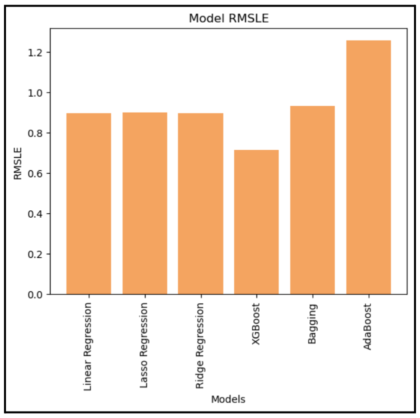

Figure 3. Bar chart of model RMSLE.

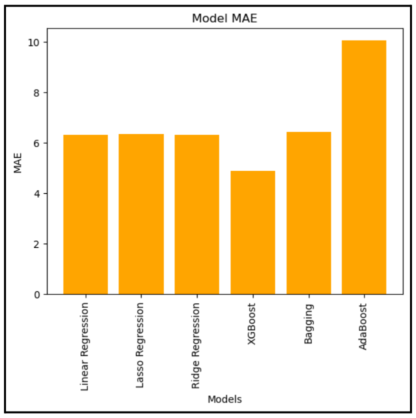

Figure 4. Bar chart of model MAE.

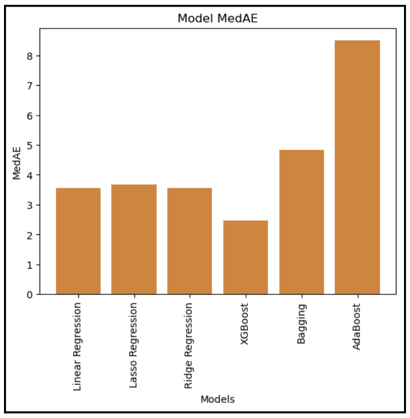

Figure 5. Bar chart of model MedAE.

Table 1 displays the test RMSLE, MAE, MedAE, and MAPE for each tuned model. The XGBoost model outperformed all other models, producing the lowest RMSLE of 0.716, MAE of 4.884, MedAE of 2.474, and MAPE of 3.752691e+14. MAPE values are large and not very interpretable because many demand values are close to 0, leading the MAPE to explode. Because we could only do manual hyperparameter tuning instead of using a formal tuning method, there is still potential for XGBoost to perform better with different hyperparameter values. Figures 3, 4, and 5 compare the six models' RMSLE, MAE, and MedAE values, respectively, in bar charts. With significantly shorter bars than all other models, XGBoost is our best regression model.

3.1 Feature Importance Analysis

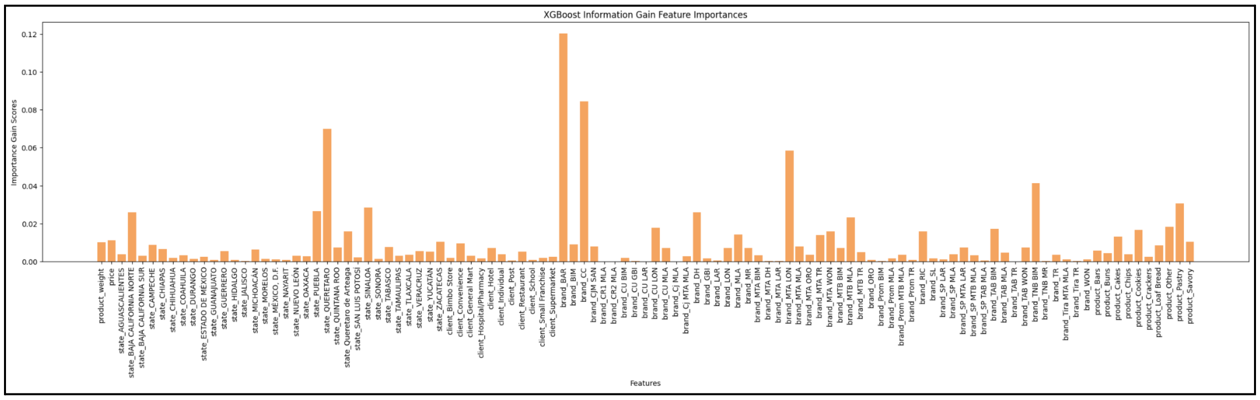

Figure 6. Bar chart of information gain feature importances of each input feature in the XGBoost Regressor model.

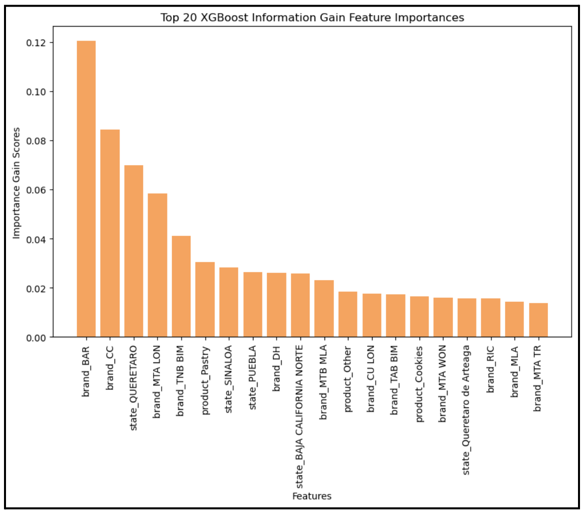

Figure 7. Bar chart of feature importances of the 20 most important features.

Feature	Importance Gain
brand_BAR	0.120298
brand_CC	0.084348
state_QUERETARO	0.069850
brand_MTA LON	0.058350
brand_TNB BIM	0.041204
product_Pastry	0.030590
state_SINALOA	0.028307
state_PUEBILA	0.026467
brand_DH	0.025971
state_BAJA CALIFORNIA NORTE	0.025872
brand_MTB MLA	0.023099

Table 2. Importance gain scores of the 11 most important features in descending order.

We explored feature importances in the XGBoost model by comparing the importance gain score of each feature, obtained from the model's feature_importances_ property. The more an attribute contributes to decisions made in trees, the higher its score. Figure 6 displays the feature importance of all 107 input features, and Figure 7 displays the same for only the 20 most important features. Table 2 shows the importance gains of the 11 features with the highest importance. Several brand, state, and product variables are the most important inputs, as identifiable in Figure 7 and Table 2. Ideally, the feature importance analysis would be more informative. However, due to almost all input features being binary dummy variables and different features having different numbers of associated variables (e.g., 55 of the 107 variables are related to state), it is difficult to gain a detailed insight into what feature is most important for the XGBoost model.

4. Discussion

From our results, we see that the XGBoost Regressor outperformed the other regression models, so it is most suitable for mapping the relationship between the input features and demand values. Of the 6 models tested, XGBoost is most efficient at handling data with high dimensionality like our encoded dataset. With a large training set size of roughly 6.5 million data points, XGBoost provided a faster execution speed than other models, which benefits the model's generalization as it can provide more accurate results with a shorter training time. It also implements a parallel tree-boosting technique that allows for more accurate approximations when finding the strongest tree model, improving predictive performance.

A previous paper using regression methods and product recipe data to predict demand [5] also found that their XGBoost model outperforms other models. Since the previous model is trained on more features than ours, it results in a lower MAE. We believe that training on more relevant features and data points would improve the performance of our current models and provide more insightful information when analyzing feature importances. Data that spans a longer time is also desirable, as we attempted to perform a time-series analysis but were limited by the short timespan covered by our dataset.

We believe the computing capabilities of our machines constrained our project. First, we were unable to use the full dataset we had access to and downsized the data by nearly 90% to build the regression models. With higher computing power, we would utilize the entire dataset and potentially add other data sources to increase the number of meaningful inputs to the models. Training on more data would likely lead to more robust and accurate models. Second, we attempted to build a Random Forest model but failed due to kernel crashes from lack of memory. More computing resources would allow us to train the model fully alongside our existing models. Lastly, we performed manual hyperparameter tuning. There is potential for the XGBoost model and our other methods to produce better results after comparing more hyperparameter values and applying a formal hyperparameter tuning method.

Our current best model is not very generalizable, as it depends on features specific to Mexico, including brands that may only exist or are only popular in Mexico, Mexican states, and prices in Mexican pesos. The model would require fitting on similar features from other countries, such as American dollars and states, to perform well outside of Mexico. A pipeline to normalize features from different countries could be added in a future implementation to extend model generalizability. Ultimately, our model may serve as a prototype for a food product prediction framework that can be improved with further tuning and more input features, provided the availability of more computing resources.

References

1. "5 facts about food waste and hunger." *World Food Programme*, 2 June 2020. www.wfp.org/stories/5-facts-about-food-waste-and-hunger.
2. "Food loss and waste." *U.S. Food and Drug Administration*, Feb. 14, 2023. www.fda.gov/food/consumers/food-loss-and-waste.
3. J. Poore and T. Nemecek, "Reducing food's environmental impacts through producers and consumers." *Science*, vol. 360, no. 6392, pp. 987–992, June 2018. www.science.org/doi/10.1126/science.aaq0216.
4. H. Ritchie, "Food waste is responsible for 6% of global greenhouse gas emissions." *Our World in Data*, March 18, 2020. ourworldindata.org/food-waste-emissions.
5. A. Garre, M. C. Ruiz, and E. Hontoria, "Application of Machine Learning to support production planning of a food industry in the context of waste generation under uncertainty." *Operations Research Perspectives*, vol. 7, no. 100147, 2020. www.sciencedirect.com/science/article/pii/S2214716019301988.
6. K. Aishwarya, A. N. Rao, N. Kumari, A. Mishra, and M. Rashmi, "Food demand prediction using machine learning." *International Research Journal of Engineering and Technology (IRJET)*, vol. 7, no. 6, pp. 3672-3675, June 2020. www.irjet.net/archives/V7/i6/IRJET-V7I6686.pdf.
7. N. Xue, I. Triguero, G. P. Figueredo, and D. Landa-Silva, "Evolving deep CNN-LSTMs for inventory time series prediction." *2019 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1517-1524, June 2019. ieeexplore.ieee.org/abstract/document/8789957.
8. A. Montoya, Grupo Bimbo, M. O'Connell, and W. Kan, "Grupo Bimbo Inventory Demand." *Kaggle.com*, 2016. www.kaggle.com/competitions/grupo-bimbo-inventory-demand.
9. J. V. Rama, "Spanish-names-surnames: Data set with Spanish names and surnames." *GitHub*, March 13, 2017. github.com/jvalhondo/spanish-names-surnames.