Grupo Bimbo Inventory Demand

Udacity Machine Learning Engineer Capstone Project

Thomas Hepner – September 27th, 2016

Definition

Project Overview

Grupo Bimbo is a Mexican multinational company that specializes in bakery product manufacturing. It runs the largest bakeries in North America, and has wide distribution networks in the United States and Mexico. The company places fresh bakery products in over 1 million stores along 45,000 routes in Mexico.

Grupo Bimbo has hosted a competition on Kaggle, tasking machine learning practitioners with building predictive models to accurately forecast inventory demand for its products.

Problem Statement

From the competition description: *"In this competition, Grupo Bimbo invites Kagglers to develop a model to accurately forecast inventory demand based on historical sales data. Doing so will make sure consumers of its over 100 bakery products aren't staring at empty shelves, while also reducing the amount spent on refunds to store owners with surplus product unfit for sale."*

In other words, the objective is to minimize forecasting error for net sales adjusted for returns (Adjusted Demand) in future sales data. My plan is to use the XGBOOST algorithm (a variant of gradient boosting) to build my machine learning model, and predict Adjusted Demand in future weeks.

Metrics

The evaluation metric for the competition is Root Mean Squared Logarithmic Error (**RMSLE**). The goal of the competition is to minimize the RMSLE of model predictions on the held-out competition data. It is calculated as follows:

$$\epsilon = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i+1)-\log(a_i+1))^2}$$

Where:
- ϵ is the RMSLE value (score)
- n is the total number of observations in the (public/private) data set
- p_i is your prediction
- a_i is the actual response for i.
- $\log(x)$ is the natural logarithm of x.

It is important to note that **RMSLE** penalizes under-predicted estimates greater than an over-predicted estimates. In other words, RMSLE will be greater for predictions that overestimate the target than ones that underestimate it.

Data Exploration

The data for the competition consists of 6 tables: **train**, **test**, **town_state**, **cliente_tabla**, **producto_tabla**, and **sample_submission**. The schema for each of the tables is displayed below in Figure 1.

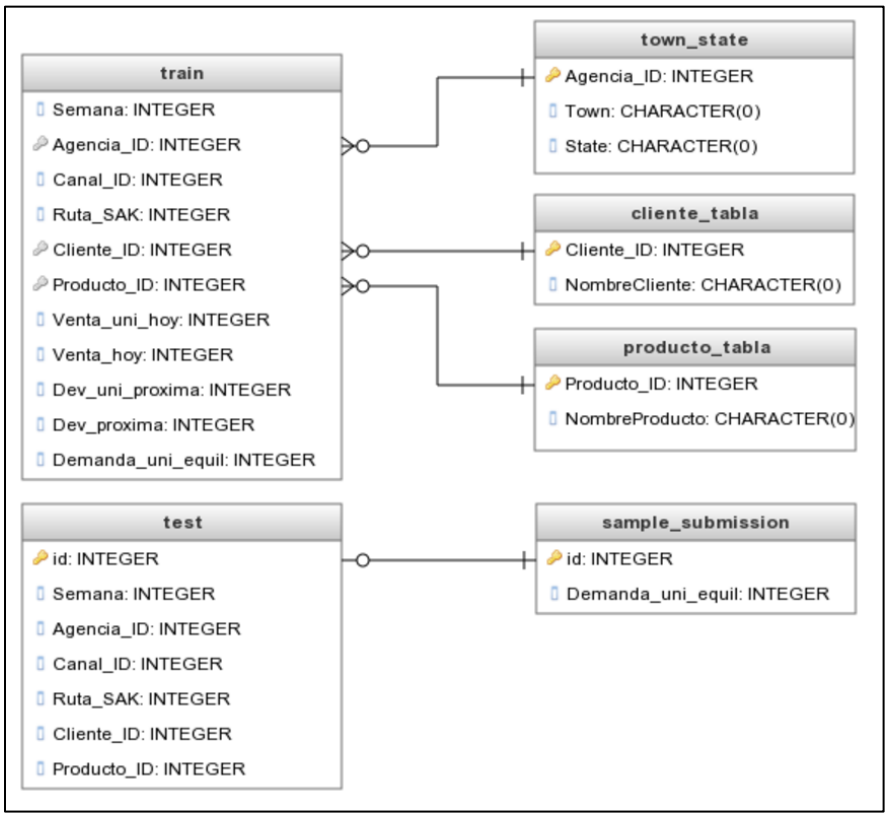


Figure 1: Schema of competition data

Because Grupo Bimbo is a Mexican company, the features are labelled in Spanish instead of English. English descriptions of each of the features are shown in Figure 2 below.

Data fields	
Semana	Week number (From Thursday to Wednesday)
Agencia_ID	Sales Depot ID
Canal_ID	Sales Channel ID
Ruta_SAK	Route ID (Several routes = Sales Depot)
Cliente_ID	Client ID
NombreCliente	Client name
Producto_ID	Product ID
NombreProducto	Product Name
Venta_uni_hoy	Sales unit this week (integer)
Venta_hoy	Sales this week (unit: pesos)
Dev_uni_proxima	Returns unit next week (integer)
Dev_proxima	Returns next week (unit: pesos)
Demanda_uni_equil	Adjusted Demand (integer) (This is the target you will predict)

Figure 2: English description of data features

Discussion of Features:

1. **Time:** The feature, Semana (Week), is always represented in the data as an integer value ranging from 3 to 9 in train, and 10 to 11 in test; it is a time series variable. The goal of the competition is to use sales records in weeks 3 to 9 (train data) to predict future sales in weeks 10 and 11 (test data); the final competition standings are determined by the RMSLE of my predictions for week 11.

2. **Numerical:** Venta_uni_hoy (gross unit sales), Venta_hoy (gross sales revenue), Dev_uni_proxima (return units), Dev_proxima (return revenue), and Demanda_uni_equil (net sales adjusted for returns) are all numerical features represented as integers in the data sets.

3. **Categorical:** Agencia_ID (Sales Depot), Canal_ID (Sales Channel), Ruta_SAK (Route), Cliente_ID (Client), NombreCliente (Client Name), Producto_ID (Product), and NombreProducto (Product Name) are all categorical features. The ID variables are represented as integers and the name variables as strings. Some of these features have very high cardinality; Cliente_ID has 880,604 unique values and Producto_ID has 1,799 unique values. Extracting meaningful information from these variables will require further exploratory analysis.

A First Examination of the Data:

I examined the first four rows of the train data in Figure 3 below as a first, rudimentary visualization of the data.

Figure 3: Header of Train Data

	Semana	Agencia_ID	Canal_ID	Ruta_SAK	Cliente_ID	Producto_ID	Venta_uni_hoy	Venta_hoy	Dev_uni_proxima	Dev_proxima	Demanda_uni_equil
0	3	1110	7	3301	15766	1212	3	25.14	0	0.0	3
1	3	1110	7	3301	15766	1216	4	33.52	0	0.0	4
2	3	1110	7	3301	15766	1238	4	39.32	0	0.0	4
3	3	1110	7	3301	15766	1240	4	33.52	0	0.0	4
4	3	1110	7	3301	15766	1242	3	22.92	0	0.0	3

Inspection of the header gave me an initial sense of what the data types of the features might be so that was the next analysis I did in Figure 4 below.

Figure 4: Train Data Types

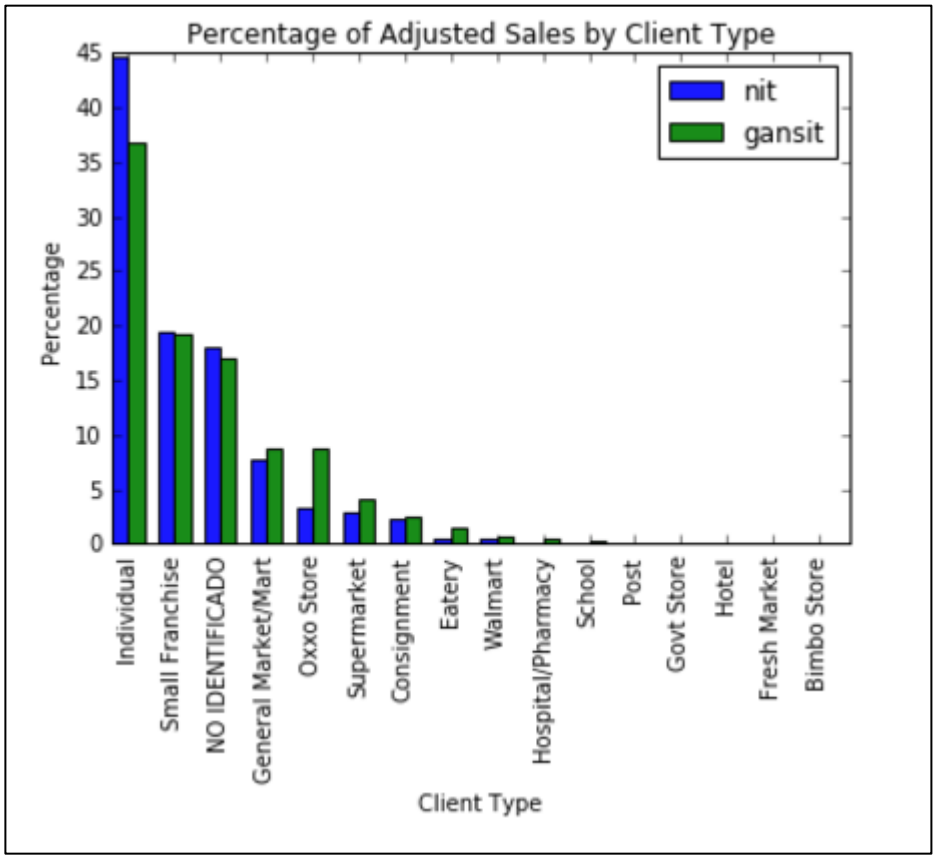
Semana	int64
Agencia_ID	int64
Canal_ID	int64
Ruta_SAK	int64
Cliente_ID	int64
Producto_ID	int64
Venta_uni_hoy	int64
Venta_hoy	float64
Dev_uni_proxima	int64
Dev_proxima	float64
Demanda_uni_equil	int64

For the next step, I performed a basic statistical summary of the data, examining the mean, standard deviation, minimum and maximum values, and the 25% / 50% / 75% quartiles for each of the features in the data. This is visualized in Figure 5 below. The high standard deviations of the Agencia_ID, Ruta_SAK, Cliente_ID, and Producto_ID stands out from the rest of the analysis suggesting that understanding these features is a key component in solving the problem.

Figure 5: Statistical Summary of Features

	Semana	Agencia_ID	Canal_ID	Ruta_SAK	Cliente_ID	Producto_ID	Venta_uni_hoy	Venta_hoy	Dev_uni_proxima	Dev_proxima	Demanda_uni_equil
count	7.418046e+07	7.418046e+07	7.418046e+07	7.418046e+07	7.418046e+07	7.418046e+07	7.418046e+07	7.418046e+07	7.418046e+07	7.418046e+07	7.418046e+07
mean	5.950021e+00	2.536509e+03	1.383181e+00	2.114855e+03	1.802119e+06	2.084081e+04	7.310163e+00	6.854452e+01	1.302577e-01	1.243248e+00	7.224564e+00
std	2.013175e+00	4.075124e+03	1.463266e+00	1.487744e+03	2.349577e+06	1.866392e+04	2.196734e+01	3.389795e+02	2.932320e+01	3.921552e+01	2.177119e+01
min	3.000000e+00	1.110000e+03	1.000000e+00	1.000000e+00	2.600000e+01	4.100000e+01	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	4.000000e+00	1.312000e+03	1.000000e+00	1.161000e+03	3.567670e+05	1.242000e+03	2.000000e+00	1.676000e+01	0.000000e+00	0.000000e+00	2.000000e+00
50%	6.000000e+00	1.613000e+03	1.000000e+00	1.286000e+03	1.193385e+06	3.054900e+04	3.000000e+00	3.000000e+01	0.000000e+00	0.000000e+00	3.000000e+00
75%	8.000000e+00	2.036000e+03	1.000000e+00	2.802000e+03	2.371091e+06	3.742600e+04	7.000000e+00	5.610000e+01	0.000000e+00	0.000000e+00	6.000000e+00
max	9.000000e+00	2.575900e+04	1.100000e+01	9.991000e+03	2.015152e+09	4.999700e+04	7.200000e+03	6.473600e+05	2.500000e+05	1.307600e+05	5.000000e+03

Figure 6: Adjusted Sales of Top Selling Products by Client Type



The client and product id variables were of particular interest due to their high cardinality and intuitive importance – it seemed straightforward that forecasting inventory demand would need to be done at the store and product level.

Before analyzing the data, the features wereI modified and transformed using the corresponding names of the ids. Both feature names had a significant number of duplicates that needed to be removed. In addition, two competition participants discovered insightful ways of parsing the features that simplified the exploratory analysis and greatly reduced feature cardinality.

The client and product name variables were transformed into **Client_type** and **short_product_name**. Client type buckets all unique client ids into 16 separate categories; short product name reduces the number of product ids approximately 60% by grouping products with similar names; For instance, any product name that contains the description 'nito' is grouped into 'nit' as this refers to a distinct Grupo Bimbo product.

Figure 6 shows the percentage sales by client type of each of the 2 top selling products (nit and gansit) in the train data. It shows that the proportion of sales by client type differs greatly for each product. Nit is sold much more in Individual stores and greater percentages of the gansit product are sold in Oxxo stores, supermarkets, and eateries. Additional detail about this visualization and analysis can be found in the exploratory analysis.ipynb file in the GitHub repository.

Algorithms and Techniques

The gradient boosted trees algorithm was used to train models for predicting adjusted sales in the test data (weeks 10 and 11). More specifically, the XGBOOST ("Extreme Gradient Boosting") library implementation of gradient boosted trees was utilized. Predictive models were built using weeks 6, 7, and 8 for the train data and week 9 as a held-out validation data set. Model performance was evaluated using the RMSLE metric on the held-out validation data.

XGBOOST, like gradient boosting, builds an ensemble of "weak" (better than randomly guessing) learners. It does this by creating a learner with the data to predict the outcome. The algorithm notes the observations in the data where error was greater, and builds another learner to better predict these outcomes. It repeats this process for a pre-specified number of iterations (*number of estimators)*, and then combines the learners in the way that optimizes the error metric.

The algorithm has many parameters, but the most important are the *number of estimators (or learners), the learning rate, and the max depth*; these were the parameters I tuned in order to minimize the objective function, the RMSLE metric, of my model predictions on the test data. The *learning rate* shrinks the contribution of each "weak" learner in the model, and *max depth* limits the depth, or number of nodes, in each individual tree.

XGBOOST also goes a step further than standard gradient boosting algorithms by formally incorporating regularization in the model to control complexity and avoid overfitting. A more detailed explanation of how the algorithm works is available here.

Benchmark

The ranking in the Kaggle competition was determined by the RMSLE of my model predictions on the private leaderboard which corresponds to week 11 in the test data. This was my ultimate benchmark, but not a useful one during the course of the competition, as my ranking on the private leaderboard was not shown until after the competition ended! However, I was able to use the public leaderboard to evaluate my predictions on week 10 of the test data. To build the best model for predicting the week 11 data on the private leaderboard, I defined my best model as the average of the RMSLE scores from my model predictions on the held-out validation set and the public leaderboard score. The formula for the benchmark described is illustrated below in Figure 7.

Figure 7: Estimate of Best Performing Model on Private Leaderboard

$$Best\ Model\ Estimate\ =\ \frac{Public\ LB\ Score\ +\ Internal\ CV\ Score}{2}$$

Other benchmarks used were (1) the public leaderboard scores of other competition participants and (2) the public leaderboard score of the sample submission (all predictions in the sample submission are 7, which is the rounding down of the mean average Adjusted Demand in the train data). The best public leaderboard RMSLE score during the competition was **0.42691;** the RMSLE of the sample submission on the public leaderboard was **0.96225**.

Methodology

Data Preprocessing

1. **Remove sales variables from data excluding Demanda_uni_equil (Adjusted Demand):**
    - Venta_uni_hoy (sales units this week)
    - Venta_hoy (sales this week)
    - Dev_uni_proxima (return units this week)
    - Dev_proxima (returns this week)

These variables were removed for several reasons: (1) they are highly correlated with Adjusted Demand, (2) they are used in the calculation to derive Adjusted Demand, (3) they would unnecessarily make my model more complex, and (4) they take up a lot of memory and slow down runtime.

2. **Created and merged additional product features into data:**
    - short_product_name
    - weight
    - pieces
    - weight_per_piece

Inspired by a Kaggle kernel by user Andrey Vykhodtsev, these features were derived from the NombreProducto (Product Name) feature which was subsequently dropped. They provide additional information which the XGBOOST algorithm would not have been able to detect and incorporate before they were added into the model.

3. Created new client variable and merged into data:
    - Client_type

This feature was also inspired by a Kaggle kernel from another user, AbderRahman Sobh. It uses NombreCliente (Client Name) to group clients into 16 larger categories like Oxxo stores and schools.

4. Generate times series lagged variables:
    - Demanda_uni_equil_tminus2 (Adjusted Demand from two weeks prior)
    - Demanda_uni_equil_tminus3 (Adjusted Demand from three weeks prior)
    - Demanda_uni_equil_tminus4 (Adjusted Demand from four weeks prior)
    - Demanda_uni_equil_tminus5 (Adjusted Demand from five weeks prior)

These features were created because it seemed intuitive that there would be a time series component to Adjusted Demand. The variables were built by taking the mean of Demanda_uni_equil (Adjusted Demand) for a specific Cliente_ID (Client ID) and Producto_ID (Product_ID) for a given Semana (week).

5. Add mean of weekly id counts:
    - Agencia_ID_count
    - Canal_ID_count
    - Ruta_SAK_count
    - Cliente_ID_count
    - Producto_ID_count
    - short_product_name_count
    - Client_type_count

During my research, I found a presentation by a top Kaggle competitor, Owen Zhang, that included helpful tips on feature engineering and using the XGBOOST algorithm. On slide 15, one of the tips is that it can be highly beneficial convert categorical features with high cardinality (such as zip codes, text, etc.) into numerical features. Taking this tip, I created the features listed above by calculating the counts of a specific ids by Semana (week), and taking the mean of the counts.

6. Encode categorical variables:
    - Short_product_name
    - Client_type

For the machine learning model to work correctly, it was important that the train and the test data have the same set of values for the two categorical features: short_product_name, and Client_type. Classes in these features with less than 5 occurrences were grouped into a single, separate category. Afterwards, the categorical variables were encoded as numerical so that the XGBOOST algorithm could use them.

7. Manually set data types to shrink data allocated in memory:

By examining the minimum and maximum values of features, along with consideration of whether they were integer or numerical values, I determined the data type for each feature that would minimize the space allocated in memory. This dramatically reduced the amount of RAM on my PC taken up by the data. Each new type assignment for each feature can be seen in Figure 8 below.

Figure 8: Data Type Assignments

Variable	Type	Numpy Data Type
Semana	*Time*	*uint8*
Agencia_ID	*ID*	*uint16*
Canal_ID	*ID*	*uint8*
Ruta_SAK	*ID*	*uint16*
Cliente_ID	*ID*	*uint32*
Producto_ID	*ID*	*uint16*
Client_Type	*Client*	*uint32*
Short_product_name	*Product*	*uint32*
Weight	*Product*	*float16*
Pieces	*Product*	*float16*
Weight_per_piece	*Product*	*float16*
Demanda_uni_equil	*Time Series*	*float16*
Demanda_uni_equil_tminus2	*Time Series*	*float16*
Demanda_uni_equil_tminus3	*Time Series*	*float16*
Demanda_uni_equil_tminus4	*Time Series*	*float16*
Demanda_uni_equil_tminus5	*Time Series*	*float16*
Agencia_ID_count	*Weekly count*	*float32*
Canal_ID_count	*Weekly count*	*float32*
Ruta_SAK_count	*Weekly count*	*float32*
Cliente_ID_count	*Weekly count*	*float32*
Producto_ID_count	*Weekly count*	*float32*
Client_Type_count	*Weekly count*	*float32*

8. Filtered data to only include Semana (Weeks) greater than 5:

This was done because weeks less than 6 do not have values for some of the time series lag variables, and it also had the secondary, but very important, benefit of reducing the space allocated by the train data in memory. This reduced the memory allocated by the train dataframe from 5.7+ GB to 2.5 GB, a reduction of over 50%.

Implementation

My implementation process was split into 4 phases:

1. **Build**
2. **Predict**
3. Submit
4. Evaluate

This was an iterative approach, where I would execute the process, examine the results, and then tweak parts of the process before repeating again-and-again. The **Build** process is encapsulated in the build.ipynb file, the **Predict** process in the predict.ipynb file, the **Submit** process was completed on the Grupo Bimbo Kaggle competition submission page, and the **Evaluate** step was done by considering the results from the **Predict** and **Submit** phases.

Build:

1. Load data into memory (train, test, products, clients).
2. Perform some, or all, of the data preprocessing steps.
3. Write modified train and test dataframes to CSV files.

The greatest complication with this part of the implementation process was dealing with (1) program runtime, and (2) memory usage. Because the data was so large, data processing steps were very slow, especially for merging dataframes, and would often use much more memory than the size of the dataframes would suggest. This made troubleshooting and optimizing code difficult and slow at many times. I dealt with this issue by modifying the data types in my data in order to reduce memory consumption.

Predict:

1. Load data and some or all of the features into memory (train, test).
2. Split train data into a training set (filtered by weeks in 6, 7, 8) and a validation set (week 9).
3. Fit XGBOOST model to train data, using validation data for model evaluation.
4. Generate RMSLE score for validation data, and save for later.
5. Generate Adjusted Demand predictions for test data.
6. Write test predictions to CSV file.

One complication that occurred during this process was that RMSLE is not an objective function available in the XGBOOST library. However, I was able to handle this by taking the natural logarithm of Adjusted Demand plus 1, the feature I was attempting to predict, and use Root Mean Squared Error (RMSE) for the objective function. Then, when making predictions on the test data, I would convert my predictions by taking their exponents and subtracting 1 from each, in order to back into predictions that minimized the RMSLE metric.

Submit:

1. Add test predictions to 7-Zip file to compress data before uploading to the submission page.
2. Submit 7-Zip file here: https://www.kaggle.com/c/grupo-bimbo-inventory-demand/submissions/attach
3. Record public leaderboard RMSLE score (The public leaderboard was determined to only include test data for week 10, and the private leaderboard included only test data for week 11. This is described in the competition forum.)

Evaluate:

1. My goal was to build a model that best predicted the Adjusted Demand for week 11. Since I wanted to avoid overfitting the data on the leaderboard, I determined my best model estimate was the model with the lowest average RMSLE score for the validation data (week 9) and public leaderboard data (week 10).

Refinement

Through each cycle of my implementation process, I modified the train and test data to include new features that I designed. Figure 9 below shows the validation scores on week 9 in the data, week 10 on the public leaderboard, and the average of both as well as the improvement in the average score.

Figure 9: XGBOOST Performance with Different Feature Sets

	XGBOOST Model[1]	Week 9 (CV Score)	Week 10 (Public LB Score)	CV+ Public LB Average	Δ Improvement
1	Categorical variables only[2]	0.648	0.645	0.647	
2	*plus client type*[3]	0.645	0.642	0.643	-0.004
3	*plus product features*[4]	0.599	0.596	0.598	-0.045
4	*plus time series lag*[5]	0.482	0.485	0.483	-0.114
5	*plus weekly counts*[6]	0.473	0.473	0.473	-0.010

[1] *Parameters are constant in all models (# of trees = 100, learning rate = 0.25, max_depth = 8, seed = 0)*

[2] *Semana, Agencia_ID, Canal_ID, Ruta_SAK, Cliente_ID, Producto_ID*

[3] *Client_type*

[4] *short_product_name, weight, pieces, weight_per_piece*

[5] *Demanda_uni_equil_tminus2, Demanda_uni_equil_tminus3, Demanda_uni_equil_tminus4, Demanda_uni_equil_tminus5*

[6] *Agencia_ID_count, Canal_ID_count, Ruta_SAK_count, Cliente_ID_count, Producto_ID_count, Cliente_ID_count*

Each set of features was tested on an XGBOOST model with the same parameters and starting seed for reproducibility purposes. The RMSLE score improvements from the addition of new features at every step are displayed in the rightmost column in Figure 9. The addition of the time series lag variables was the most influential set of new features added after the initial set of categorical variables, yielding an almost 20% reduction in the RMSLE even after already including new client and product features.

After finishing the processing of building and testing new features, I tuned the parameters of the XGBOOST model on my best feature set which was the set of all new features. Model tuning was performed on (1) the number of trees, and (2) the max depth. The results are shown in Figure 10 below.

Figure 10: XGBOOST Model Tuning - RMSLE Scores [1]

Validation		max depth		
		5	10	15
# of trees	100	0.484	0.469	0.466
	200	0.484	0.467	0.466

Public Leaderboard		max depth		
		5	10	15
# of trees	100	0.483	0.470	0.473
	200	0.483	0.468	0.473

Average		max depth		
		5	10	15
# of trees	100	0.483	0.469	0.469
	200	0.483	0.467	0.470

Runtime (minutes)		max depth		
		5	10	15
# of trees	100	13.5	31.3	53.1
	200	13.5	49.9	75.0

[1] *Includes all features. Learning rate is set to 0.25 and seed to 0.*

The XGBOOST model with max depth of 15 and number of trees set to 200 resulted in the lowest validation score, but did not improve performance on the public leaderboard; this appears to be a case of overfitting the training data. The model with max depth set to 10 and number of trees set to 200 resulted in the second lowest validation score, but also in the lowest public leaderboard score; this model was selected for additional tuning.

Results

Model Evaluation and Validation

Additional tuning was done on the model's learning rate and max depth parameters. The results of the tuning are displayed below in Figure 11.

Figure 11: Final XGBOOST Model Tuning - RMSLE Scores [1]

Validation		max depth	
		10	12
learning rate	0.15	0.468	0.466
	0.25	0.467	0.466

Public Leaderboard		max depth	
		10	12
learning rate	0.15	0.470	0.468
	0.25	0.468	0.468

Average		max depth	
		10	12
learning rate	0.15	0.469	0.467
	0.25	0.467	0.467

Runtime (minutes)		max depth	
		10	12
learning rate	0.15	58.2	60.6
	0.25	49.9	45.9

[1] *Includes all features. Seed is set to 0.*

The final model selected had a learning rate of 0.25 and a max depth of 12. The final evaluation of the model was the RMSLE score on the private leaderboard data (Week 11). The RMSLE score for week 11 was 0.473. This shows a very

modest amount of overfitting as the RMSLE scores for Week 9, 10, and 11 were 0.466, 0.468, and 0.473 respectively. If my final model had been submitted before the competition deadline, it would have been in the top 8% of all teams. Unfortunately, the competition ended before I could complete my project and my best submission only had a private leaderboard score of 0.483, in the top 14% of all teams.

Justification

The benchmarks I established for the evaluation of my model were (1) the private leaderboard scores of my competitors, (2) the average of the week 9 held-out validation set and the week 10 public leaderboard score, and (3) the private leaderboard score of the sample submission.

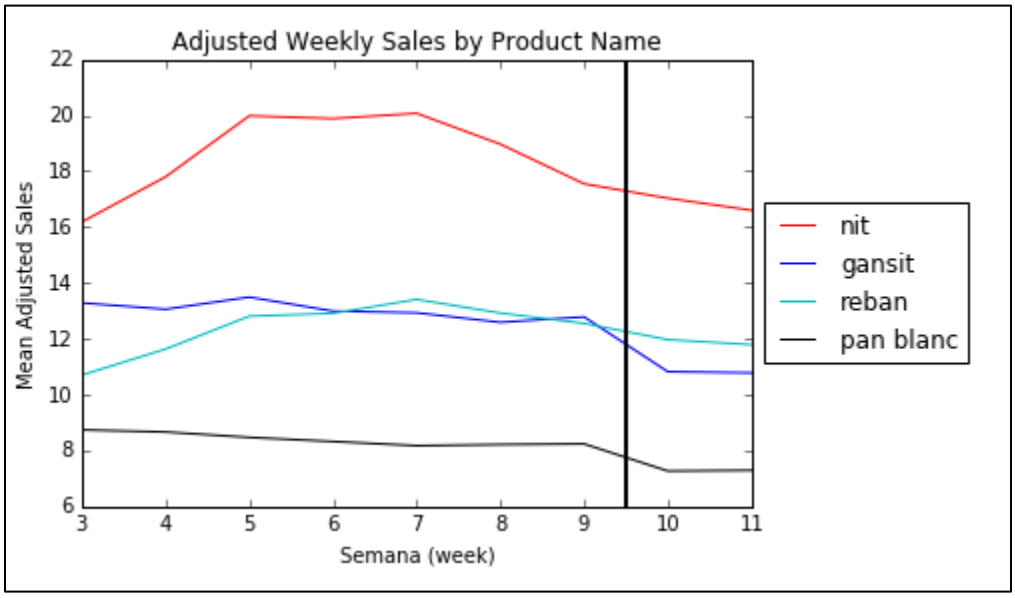
My best model had a private leaderboard score of 0.473, and an average Week 9 + Week 10 score of 0.467. This is an ~51% improvement over the sample submission on both the private leaderboard and public leaderboard scores, vastly outperforming the benchmark I defined for this project.

Compared to the median team, my model was 8.6% better on the private leaderboard. However, the best team's submission that won the competition scored 0.443 on the private leaderboard, which was 6.5% better than my final model; this suggests that I had a lot of room left to improve my best model.

Conclusion

Free-Form Visualization

Figure 12: Mean Adjusted Weekly Sales for Top Products



The visualization, created using Matplotlib in the free-form-visualization.ipynb file, shows average weekly sales for each of the top 4 best-selling products in the data. The thick, vertical black line between weeks 9 and 10 separates the train data from weeks 3 to 9 with the test data in weeks 10 and 11.

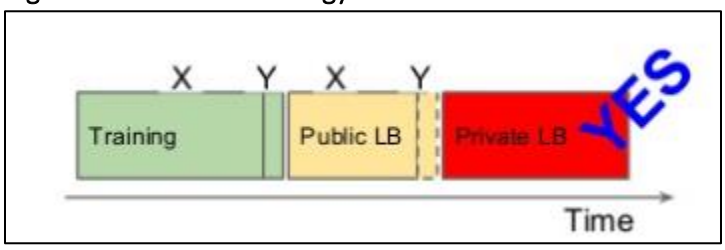
Reflection

Handling the size of the data was a major difficulty in this project. The original size of the train data was 3.1 GB and the test data was 245 MB. After adding all features to the train data, the data size exceeded 10 GB, and the code in build.ipynb would periodically fail when trying to join two tables together or create additional variables. The predict.ipynb file ran for several hours at a time as the XGBOOST algorithm attempted to handle the massive size of the data. Taking a tip from a forum post by Nathan George, I was fortunately able dramatically reduce the memory allocated by my data.

Initially, the read_csv() function from pandas was defaulting all integer values to int64 and all numerical values to float64, and string data types took up more even memory than integer types. I was able to modify the data types, for instance using the uint8 datatype instead of int64 for the variable Semana since it only had integer values between 3 and 11. Overall, these changes reduced my RAM usage by ~80%.

Deciding upon model validation strategy was another tricky aspect of the project. Ultimately, I decided upon using week 9 from the train data as a hold-out validation set, using weeks 6, 7, and 8 to train my model. I decided upon this strategy based on my review of Owen Zhang's presentation on Kaggle competition validation strategies. In particular, I found this the diagram from the presentation in Figure 13 below particularly helpful for developing my validation strategy.

Figure 13: Validation Strategy when Time is a Factor



Unfortunately, I decided upon this strategy too late in the competition, and scored in the top 14% of teams in the private leaderboard when I could have scored in the top 8% using this validation strategy. Nevertheless, I learned a lot from this mistake and will be apply to apply the knowledge I gained in this project to my future career in data science…and future Kaggle competitions of course!

The most interesting aspects of the competition were designing the Client_type and short_product_name features. My university and career background is in economics and finance, so I was relatively unfamiliar with using text data in a predictive model. However, it was fascinating to tease out additional information from the NombreProducto and NombreCliente variables so that the XGBOOST model could make better predictions on the test data.

The robustness of the model was evaluated by comparing the model's performance on the internal validation data (week 9) to the out-of-sample test data in weeks 10 and 11. The RMSLE scores of my predictions on the public and private leaderboards were only 0.4% and 1.5% greater than my validation score respectively. This demonstrates the model I designed is fairly robust to predicting out-of-sample. It would be interesting to monitor and evaluate its performance on future data beyond week 11, but unfortunately Grupo Bimbo is not providing the data at this time.

Improvement

Obviously, there was a lot of room for improvement in my predictive model building approach as the top team was able to achieve an RMSLE of 0.443 on the private leaderboard compared to my 0.473, a 6.8% improvement. If I were to do this competition again with the knowledge I have now, or I were to continue improving by forecasting accuracy, I would have experimented or investigated several other aspects of the competition.

a) *Feature engineering:* Most of the top competitors used the XGBOOST algorithm in at least some part of their prediction process. However, large differences between teams on the private leaderboard mostly seemed to be the result of building better features, or *feature engineering*. This is apparent, from own results, where simply adding time series lag variables resulted in a 20% improvement from my previous best RMSLE score whereas tuning model parameters typically only improved the third decimal place in the RMSLE score. Some of the top competitors were able to generate and test hundreds of features. I could have spent more time on this aspect of the competition and likely improved my best private leaderboard score substantially.

b) *Cross Validation Strategy:* This was my first Kaggle competition where time series variables were so influential, and it greatly impacted the development of my cross validation strategy. Other competitors trained their models on weeks 6 and 7 and validated them on weeks 8 and 9, or even tried other validation strategies. I did not experiment with

other strategies due to time constraints, but I suspect that I could have improved the accuracy of my model by doing so.

c) *Additional Algorithms:* I would have liked to try a neural network based approach in addition to my gradient boosted trees approach. Neural networks have been shown to perform well with time series data in other applications, and I suspect it could have done well in this competition. Unfortunately, I did not have any experience using neural networks, but this is definitely something I want to learn, and add to my toolbox in the future.

d) *Model Stacking or Ensembling:* Model stacking and ensembling typically yield improvements in predictive performance, and I have used them before, but I had difficulty utilizing them for this specific application. I devoted a lot of time to utilizing ridge regression and random forests techniques for feature selection with the high cardinality categorical variables (Cliente_ID, Producto_ID, etc.), and then attempted to incorporate the results into my XGBOOST algorithm. Unfortunately, I did not have any success with this approach, but it's entirely possible that with enough experimentation this approach could have improved my best XGBOOST model.

e) Performance: As previously discussed, dealing with the large data sets was difficult and I was able to reduce their size substantially. However, it still took 30 to 60 minutes to fit XGBOOST models to the train data depending on the model parameters. Since I was doing all of the work on my personal computer, the runtime of my algorithms ate up a lot of my productive capacity. In the future, I would like to use tools like Amazon Web Services or Apache Spark to fit my algorithms more efficiently, or at least with rented computational capacity in order to free up more bandwidth for data exploration and development.