Store Sales - Time Series Forecasting

Ritu Pravakar pravakar@usc.edu
Charlene Yuen yuenchar@usc.edu
Sreenidhi Iyengar Munimadugu munimadu@usc.edu
Mahek Savani msavani@usc.edu

Abstract

The paper outlines the process of developing a solution and the final outcome from the Kaggle Competition Store Sales. Given dates, store and product information, promotions, sales numbers, as well as additional information such as gas prices and holidays, we utilized what was given to make a prediction on the sales of product families at different Favorita stores in Ecuador. Below we detail our progress with different pre-processing methods, feature engineering, and models.

1 Introduction

In current society, there is a multitude of applications for forecasting, ranging from the weather to the economy. One particular application is forecasting store sales, allowing grocery stores to anticipate how much inventory to purchase. Given this information, grocers can stock exactly how much is in demand from customers without running out or having extra. Therefore, there lies great potential in reducing food waste, as well as saving time and other resources. For the purposes of the competition, we outline our thought process of feature engineering and developing a model for forecasting store sales.

2 Data Preprocessing

The provided data included dates, store and product information, promotions, sales numbers, as well as additional information such as gas prices and holidays. To begin, we read in the data from the provided CSVs and perform some initial cleaning. We extract the date from the index and create a new column 'days_from_2013' for easy shifting of the time series data. We also group the data by store and product family to calculate the mean sales for each group. In the data containing daily oil prices, there were some NaN values, which we filled with interpolation backward based on the next valid value, which manages to capture the overall trend yet fill in missing points. We later also drop NaN values to keep the most relevant data. The same happened with the data containing transactions from different dates and stores, so a similar filling of missing values was done by setting null values to 0. We also grouped transactions by their store number and date which made taking moving averages over transactions and merging with the rest of the dataframe easier.

3 Feature Engineering

Next, we perform feature engineering on the data to create additional features that may help improve the accuracy of our forecasts.

3.1 Oil

The Ecuadorian economy has been shown to be dependent on exports, one of which is oil, making up a hefty 36% of total exports [1]. Due to this dependency, it follows that when the economy is poor, high oil prices would result in plummeting sales. Correspondingly, when plotting oil prices and sales with respect to each other, we found a negative trend (Figure 1).

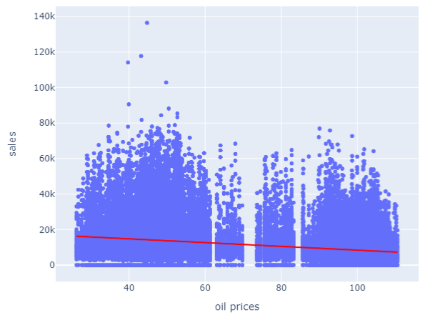


Figure 1: Correlation between oil and sales

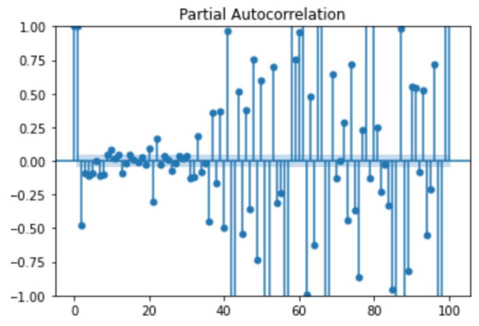


Figure 2: 100 oil lags and partial autocorrelation

We then took the moving average over a week, two weeks, and a month to capture any possible correlations to sales. We also used the partial autocorrelation plot of oil lags from 1 to 100 and found that there were many lag features that contributed to new correlations, so we simply picked lags with the largest values (Figure 2).

3.2 Holidays and events

We expected holidays to be correlated with an increase in sales, particularly near the end of the year (Figure 3).

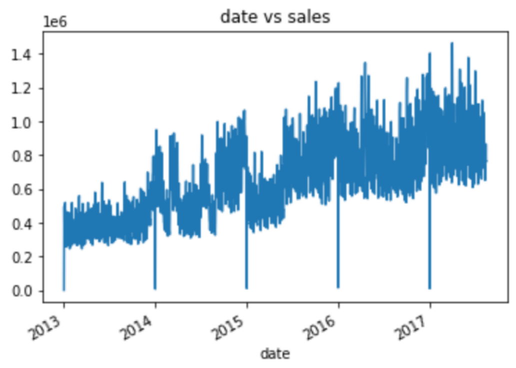


Figure 3: Total sales by year

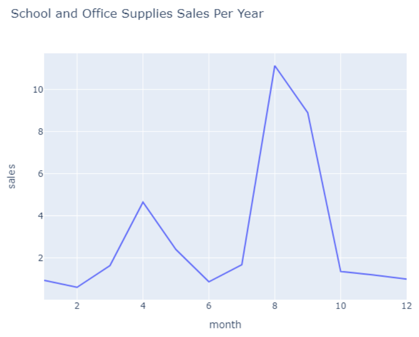


Figure 4: Sales of school by month

Thus, we separated workdays from holidays based on if the day is of type Bridge, Transfer, Additional and untransferred, as well as a holiday that is untransferred. One-hot encoding was also done for events, and features were tested for certain occurrences such as the New Year, Black Friday, the earthquake, or Easter. However, overall they did not seem to make a large improvement so not all of these features were included in our final solution. Other patterns found included sales being higher on weekends and sales of school supplies spiking during the back-to-school seasons of April-May and August-September, which is shown in Figure 4.

These findings led us to add features dictating if the date was a school season or the weekend which both correlate with higher sales.

3.3 Stores

From the given store data, we identified features such as certain stores being the only store of that type or a store that opened later, which could identify patterns in the data (Figure 5).

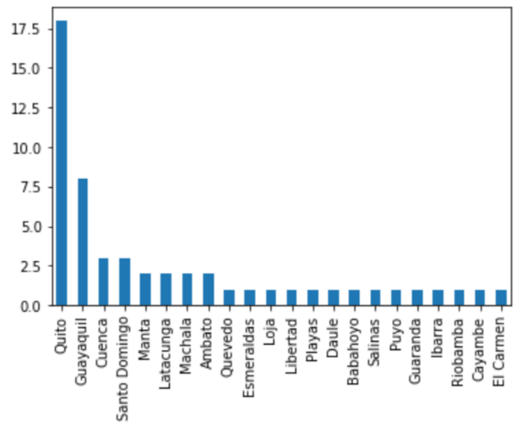
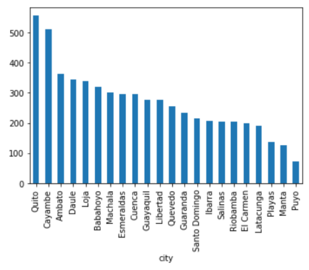
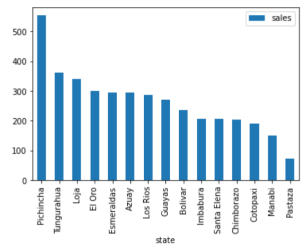


Figure 5: Count of different stores

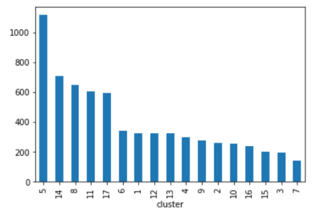
Merged with training data, we can also categorize stores and compare the average sales of each city, state, and cluster. Our findings concluded that the top cities with the best overall sales were Quito, Cayambe, Ambato, and Dalue; the states with the best sales were Pichincha, Tungurahua, Loja, and El Oro; the clusters with the best sales were 5, 14, 8, and 11. Plots can be found below in Figure 6.



(a) Total sales by city



(b) Total sales by state



(c) Total sales by cluster

Figure 6: Total sales by city, state, cluster from 2013-2017

With this knowledge, we tested new features for the best sales from each category, which gave a small amount of improvement though it did not make it into our final solution due to time constraints. Overall, the stores data gives good information on patterns based on location and groupings of the stores.

3.4 Transactions

From the transaction data, intuitively transactions should be correlated with sales, which we found had a correlation of 0.8. Apart from that, we filled in missing data with zeroes and grouped the data by store number and date so we could take moving averages and lags over the transactions to capture any similarities to sales. We also identified the total number of items on promotion in store, which generally proportional to sales excluding an outlier as shown in Figure 7.

Additional one-hot encoding was done for other time-related features such as days of the week, days, months, years, etc. For example, weeks can hold patterns in average transactions (Figure 8) and seasonal patterns may be caught by these features, such as sales rising towards the end of the year (Figure 3).

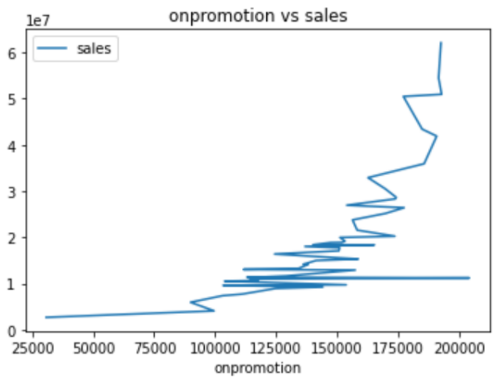


Figure 7: Items on promotion vs sales

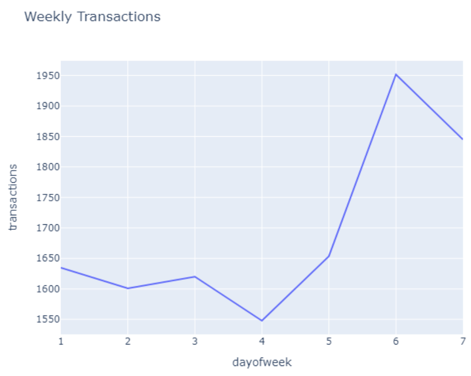


Figure 8: Average transactions per week

4 Models

4.1 XGBoost

In XGBoost a common decision is made by all decision trees where the results of the previous tree affect the results of the next decision tree. It is one of the fastest decision tree algorithms and predicts more accurately than a single decision tree.

4.2 Random Forest

Decision trees select important features and can give the relationship between features and prediction. This helps in a better interpretation of the model. A random forest is an ensemble of trees where at each node, the best split is chosen among the randomly selected subset of features. For the sales forecast, the previous value is used by the predictor variable.

4.3 Hybrid Models

We learned the components of the model in an iterative way using a hybrid model. Regression models like linear and ridge can extrapolate trends while ensemble methods like random forests cannot. Random forests can learn the interactions between different features. For our model, we first used ridge regression to extrapolate. In the next step, we transformed the target and removed the trend. We applied random forest to the detrended residuals, which was the difference between the target the model was trained on and the predictions it made [3].

Algorithm 1	Algorithm 2	Mean squared log error	Root mean squared log error
ExtraTreesRegression	RandomForest	0.00000289	0.00032
RandomForest	Linear	8.54	0.52
RandomForest	Ridge	8.48	0.51
Linear	RandomForest	6.80	0.47
Ridge	RandomForest	6.86	0.47

Table 1: Comparison of Mean squared log error and Root mean squared log error

4.4 LightGBM

LightGBM [4] is an ensemble model, where the result of all trees are combined in the output. XGBoost [5] pre-sorts the data by features and then traverses the data set to find the best segmentation point on features. This increases memory usage and time, hence LightGBM model is optimized in Gradient Facilitated decision tree algorithm.

We also compared the performance of the model using XGBoost, LightGBM, Linear Regression, and Random Forest.

5 Final Results

After modeling hybrid models we found out that it will take a lot of hyperparameter tuning for all the models at the same time to lower the RMSLE loss, so instead we decided to tune the hyperparameters of the Random Forest Regressor.

We tune the hyperparameters with Optuna, a hyperparameter tuning library that is used to find the best hyperparameters for the model. We split the data into training and validation sets, and apply log transformation to the sales data. We then fit the model on the training data, and make predictions on the validation set.

The parameters we tried to tune for random forest regressor are as follows:

Criterion: The function to measure the quality of a split. We used RMSE for this parameter

Bootstrap: Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree.

Max Depth: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

Max Features: The number of features to consider when looking for the best split.

Max Leaf Nodes: Grow trees with max_leaf_nodes in the best-first fashion. Best nodes are defined as relative reduction in impurity. If None then an unlimited number of leaf nodes.

N Estimators: The number of trees in the forest.

Min Samples Split: The minimum number of samples required to split an internal node.

Min Samples Leaf: The minimum number of samples required to be at a leaf node.

```
criterion: 'squared_error',
bootstrap: [True, False],
max_depth: 1, 10000,
max_features: '['auto', 'sqrt', 'log2']',
max_leaf_nodes: 1, 10000,
n_estimators: 30, 5000,
min_samples_split: 2, 100,
min_samples_leaf: 1, 50
```

The best parameters after the tuning are as follows:

```
'criterion': 'squared_error',
'bootstrap': 'False',
'max_depth': 9733,
'max_features': 'auto',
'max_leaf_nodes': 4730,
'n_estimators': 700,
'min_samples_split': 3,
'min_samples_leaf': 8
```

We found out from parameter tuning that increase in the number of trees (n_estimators) in the Random Forest Regressor, gave the highest improvement in reducing the RMLSE error/loss.

Also, we found an interesting observation that after a certain point adding the number of trees didn't reduce the RMLSE error. We found out that the decrease of RMLSE Loss plateau at 700-800 estimators beyond that increase just overfits the model without any substantial decrease in RMLSE. Hence we choose 700 n_estimators to be a good choice for the data.

After training the model and making predictions on the validation set, we evaluate the performance using the mean squared logarithmic error (MSLE). We then apply the model to the test data, and make final predictions. We post-process the predictions by applying the inverse of the log transformation, and save the results to a CSV file.

The below figure gives the idea of an increase in the number of trees vs a decrease in RMLSE loss (Figure 9).

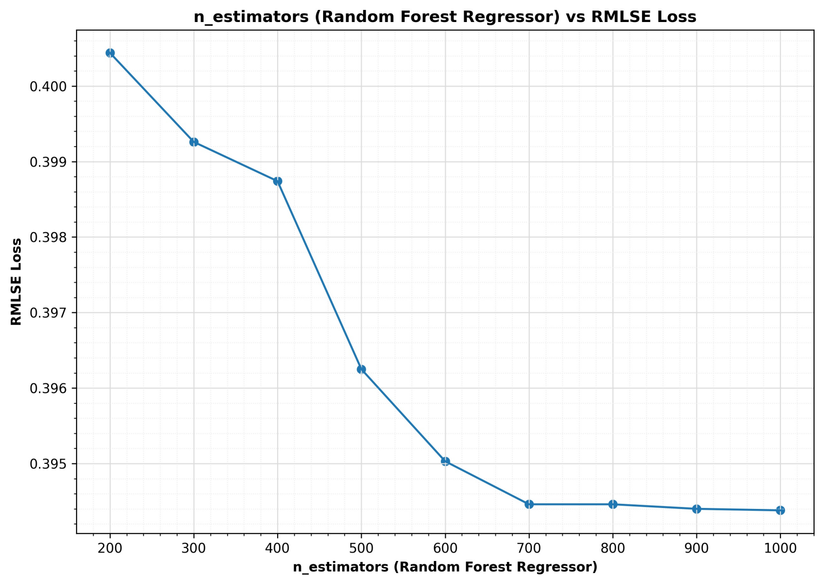


Figure 9: Hyperparameter tuning results

6 Running instructions

To run the code and generate the results, first ensure that all necessary libraries are installed and imported. Upload as a Kaggle notebook and run the code cells in the order they are presented in the notebook. The final predictions will be saved to the 'submission.csv' file.

References

[1] Ecuador's economy and oil: https://www.bmz.de/en/countries/ecuador/economic-situation-51990

[2] Kaggle Time Series Tutorial: https://www.kaggle.com/learn/time-series

[3] https://www.kaggle.com/code/ryanholbrook/hybrid-models

[4] Ke, Guolin, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. "Lightgbm: A highly efficient gradient boosting decision tree." Advances in neural information processing systems 30 (2017).

[5] Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, pp. 785-794. 2016.