Rossmann Store Sales – Time Series Prediction

Data Overview 1
Import data 2
Split the data 2
Normalization 2
Feature Selection 2
Evaluation 2
Exploration of feature and model selection 2
Introduction 2
Cleaning and EDA 3
Feature Engineering 3
Model Selection 4
Setting up the model and different regression approaches 4
Understanding the shape of the problem 4
Model exploration and fit 4
Initial exploration 4
Extreme Gradient Boosting (XGBoost) 5
Error! Bookmark not defined.

Data Overview

Import data

We loaded the data from CSVs found in the following location:
https://www.kaggle.com/c/rossmann-store-sales/data
The data contains train, test and store tables. The train table was used for feature engineering and model validation. The store table was used for additional store-level feature engineering. The test table contains the predictor data for the final submission.

Split the data

After feature engineering, the train table was linearly split with the last 42 days being the validation data and the prior days the training data for the linear regression. The linear split was employed because the order of time series data is important unlike in cross-sectional regression approaches. Holdout validation was also employed for the ARIMA, KNN regressor and Prophet models. For the XGBoost we used a time series fold split implemented in sklearn's grid search CV package. This allowed for more splits for the model to cross-validate on.

Normalization

We employed extreme gradient boosting and linear regression so normalization was not necessary. We considered normalization for the KNN regressor but its performance on the indicator variables was so poor in comparison to the linear regression and XGBoost that we did not test it with additional predictors that would have required normalization.

Feature Selection

Feature selection was accomplished with estimator significance tests for linear regression. Feature selection was not needed in the gradient boosted tree model as the algorithm subsamples the features and makes splits based on reductions in mean squared error.

Evaluation

Models were trained and evaluated with root mean squared error as the Kaggle competition used root mean squared percent error.

Exploration of feature and model selection

Introduction

We researched the following interviews to inform our approach to model building and feature engineering. Both the first and second place finishers stressed the importance of exploratory data analysis and feature engineering.

First Place Finisher Interview
http://blog.kaggle.com/2015/12/21/rossmann-store-sales-winners-interview-1st-place-gert/

Second Place Finisher Interview
http://blog.kaggle.com/2016/02/03/rossmann-store-sales-winners-interview-2nd-place-nima-shahbazi/

Cleaning and EDA

Many of the observations in the training data exhibited unusual behavior across time. This was due to different promotions, competition and refurbishments. It was important to remove outliers so that our models did not overfit the training data. For example, we removed observations where the stores were closed to not bias our models downward. Additionally, we removed observations where the stores were open, but the sales were zero.

Feature Engineering

The creation of features for a predictive model is of tantamount importance. The model is only as good as the data it can learn from. This means creating variables that will help the model predict different levels of the dependent variable. Looking at the data we recognized that it had a cyclical shape. This wave-trough shape occurred at multiple temporal levels so we created multiple temporal indicator variables. Additionally, we hypothesized that previous store traffic would define future sales. Finally, we incorporated the store level data that we had been provided with. This classified stores into different groups and classified their inventories into different groups. The store level data also provided information about how long a specific store had a competitor for and how far that competitor was from the store. Full detail of the engineered variables is provided in the table below:

Variable	Definition	Location
Month	Integer for month	Created from train table
Year	Integer for year	Created from train table
Week	Integer for week	Created from train table
Trend	Counts up by one for each day	Created from train table
Competitor Active	Whether or not competitor exists	Merged from store table
Competitor Distance	How far the competitor is	Merged from store table
StoreType	Category of store	Merged from store table
Assortment	Category of offerings	Merged from store table

Mean Customers	Mean of customers in training data	Created from train table

In addition to the variables above, we also used the variables provided in the train table. These included promos, holidays and a 0,1 indicator for whether the store was open on the given day among other variables. We dropped the holiday indicators because they overlapped with the store open indicator variable.

Model Selection

Both of the top finishers used extreme gradient boosting to model the data. For this reason, we chose to take a similar approach. We also utilized a linear regression because of its simplicity and ease of creation. In addition to the models that we submitted results on Kaggle for, we also explored ARIMA, Prophet and KNN Regressor models.

Setting up the model and different regression approaches

Understanding the shape of the problem

Our data is at the store day level. We have 1,115 different stores in the training data and 865 different stores in the testing data. We needed a model that could predict the revenue of each store in the test set over a period of 6 weeks, 42 days.

Model exploration and fit

There were two primary ways to approach the competition. We could fit a model for each individual store, 865 fits and predict the revenue with each fit, or we could fit one model and predict the revenue of each store. The benefit of the multi fit approach was that it could capture the sales at an individual store level. However, this model is computationally intensive because we have to fit it so many times. The other approach uses store types and assortments to differentiate between per store sale behaviors. This model is much simpler to set up and benefits from a much greater sum of data to train on.

Initial exploration

We fully explored two model types, linear regression and extreme gradient boosted trees. However, for comparison, we also performed nested cross validation using K-Nearest Neighbors on normalized features and explored ARIMA and Prophet models. The K-Nearest Neighbors approach was thrown out early on in the process because it had poor performance within the cross-validation folds. We also attempted to fit an Autoregressive Integrated Moving Average (ARIMA) model, however, the acf and pacf plots suggested lag terms that far exceeded what our machines were capable of. Additionally, auto arima predicted a model of the form (0, 0, 4) which performed exceedingly poorly. Seasonal ARIMA was also explored and difficulty was also found in finding the right lag lengths. Finally, we also attempted to fit a time series Prophet

Model (https://facebook.github.io/prophet/docs/quick_start.html). However, this model had difficulty handling the seasonality of the data even after adding a monthly term.

Models Tested and Structure:

1. Linear Regression - Per Store and then recombine results (Best results)
2. ARIMA - Full results at once (not taken to completion)
3. Prophet - Full results at once (not taken to completion)
4. KNN Regressor - Per Store (not taken to completion)
5. XGBoost - Per Store (Third best results)
6. XGBoost - Full results at once (Second best results)

Extreme Gradient Boosting (XGBoost)

Given that the top finishers had all employed XGBoost, we were interested to use it as well. We used two primary approaches to fit the XGBoost models. We created a loop to fit an XGBoost Regressor for each store within the training data and then predict on the test data. We also created a model to run once on all stores.

The per store XGBoost utilized the following parameter grid:

```
p_grid = {'colsample_bytree':[.3,.5,.8],
        'learning_rate':[0.1,0.3],
        'max_depth':[5,8]}
```

The model utilizing the per store grouped grid search did not perform as well as the model that fit all stores at once. This is interesting because we hypothesized that a per store fit would produce a more tailored model. Ultimately, the per store approach appeared to overfit the data. This could have been because the data contains outliers that we missed. Given that this version of the regressor trained on fewer data points, these outliers would have a greater effect. The model reported an RMSPE of 0.18383 and 0.18127 on the private and public leaderboards.

The XGBoost Regressor utilizing all stores at once had better performance and ran much more quickly. This model had the added benefit of leveraging the store level metrics such as competition and assortment. The model reported an RMSPE of 0.16981 and 0.17526 on the private and public leaderboards. This indicated to us that the stores do not differ too much between each other, more important factors are the store type in general and the assortments they offer.