Kaggle Challenge: Predicting Rossmann Store Sales

Mingu Jo
University of California, Berkeley

Wonjohn Choi
University of California, Berkeley

1 Abstract

Our project attempted to apply linear regression and various machine learning techniques to a real-world problem of predicting drug store sales. Rossmann, Germany's second-largest drug store chain, has provided past sales information of 1115 Rossmann stores located across Germany. We preprocessed, feature-engineered the data, and examined 4 different statistical / machine learning analysis for forecasting sales of each store: multivariate linear regression, random forest, gradient boosting, and TODO. Then, we compared the methods' predictive power by computing Root Mean Square Percentage Error (RMSPE). We found that TODO:X performed the best with a RMSPE score of TODO:XXX.

2 Introduction

The objective of this project is to predict 6 weeks of daily sales for 1115 Rossmann stores located across Germany using the data which was provided by Rossmann through Kaggle. The motivation of this project is the following: by reliably predicting daily sales, store managers may decrease operational costs, and create effective staff schedules (ex. by assigning more staffs during busy days). Also, we would like to identify which type of techniques be effective in a real-world sales prediction task. Before building any regression or machine learning models for the prediction, we attempted to define which features are significant to determine daily sales of stores. By performing exploratory data analysis, we found how some features such as promotion and competitor distance affect the sales.

(TODO: literature review) Is the literature review in the Introduction informative and organized? To confirm our assumption and apply various techniques for the prediction, we first explored several similar efforts that made for sales prediction task in the past in order to set the basis of our study.

- Linear Regression: The professor at Western Illionois University suggests multiple linear regression with feature selection and multicollinearity diagnostics to be a popuplar technique in sales forecasting. This model was applied for predicting the number of subscribers that a cable station can expect, which is closely related to our task. We decided to apply this technique along with partial t-test and stepwise regression to select significant features.

- Random Forest: the random forest is a tree-based ensemble method where all trees depend on a collection of random variables. (Breiman). Breiman suggests that random forest algorithm is actually from a classification viewpoint but the approach is applicable to regression as well. Therefore, we consider the algorithm is also applicable to our regression task. Nils Adriansson and Ingrid Mattsson in their thesis applies the algorithm into a GDP forecast task, showing its power in time-series forecasting (Adriansson). We assume this algorithm can outperform linear regression because Breiman states that the RF does not need to assume any distribution for variables. (Breiman)

- Gradient Boosting: We also found useful article by Jay Gu on gradient boosting regression. In his article, he asserted that, for predictive analytics, the gradient boosted trees model (GBM) is one of the most effective machine learning models. As he did, we have also witnessed a good number of winning solutions for various kaggle competitions that contain boosted tree models as a crucial component. We got the basic framework from the textbook ( ). Especially, we noticed that Extreme Gradient Boosting (xgboost) is similar to gradient boosting framework but more popular and efficient. We decided to fine-tune our dataset and apply the algorithm on it for better prediction.

2.1 Dataset and Preprocessing

Data sets are given by Rossmann through Kaggle. There are three files: test.csv, train.csv, and store.csv. train.csv contains 1017209 observed daily sales of different stores from 2013 to 2015. store.csv contains supplementary information for each store (41088 lines because there are 41088 stores). train.csv and store.csv both contain Store id that we can use to join the train and store data sets. Finally, test.csv has 41088 lines of daily sales of different stores but the value of Sales column is omitted. We are expected to predict the value of Sales column for the test set (test.csv) by using the training set (train.csv) and store data (store.csv).

The following describes fields in test.csv and training.csv:

- Store - the unique id for each store (positive integer ranging from 1 to 1115)
- DayOfWeek - the day of week (positive integer ranging from 1 to 7 where 1, 2, ..., 7 correspond to Monday, Tuesday, ..., Sunday)
- Date - the date (string of format YYYYMMDD)
- Sales - the total amount of revenue for this given day (positive number between 0 and 41551). This value is what needs to be predicted, so only exists in training.csv.
- Open - an indicator of whether the store was open on the given day (0 if closed, 1 if open)
- Promo - an indicator of whether the store was running a promotion on the given day (0 if not running, 1 if running)
- StateHoliday - an indicator of whether the given day was a state holiday (0 if not a state holiday, a if public holiday, b if eastern holiday, c if christmas)
- SchoolHoliday - an indicator of whether the store was affected by a nearby school holiday on the given day (0 if not affected, 1 if affected)

The following describes fields in store.csv:

- Store - the unique id for each store (positive integer ranging from 1 to 1115)
- StoreType - the store's model (4 types: a, b, c, d)
- CompetitionDistance - the distance in meters to the nearest competing store (positive number franging from 20.0 to 75860.0, and NA's if no competitors)
- Assortment - the store's assortment level (a = basic, b = extra, c = extended)
- CompetitionOpenSinceMonth - the approximate month of the time when the nearest competing store opened (positive integer ranging from 1 to 12, and NA's if no competitors)
- CompetitionOpenSinceYear - the approximate year of the time when the nearest competing store opened (positive integer ranging from 1900 to 2015, and NA's if no competitors)
- Promo2 - an indicator of whether the store was participating in a promotion that runs over a period of time (0 if not participating, 1 if participating)
- Promo2SinceWeek - the approximate week of year when the store started to participate in promo2 (positive integer ranging from 1 to 50, and NA's if no promo2)
- Promo2SinceYear - the approximate year of time when the store started to participate in promo2 (positive integer ranging from 2009 to 2015, and NA's if no promo2)
- PromoInterval - the months when the promo2 started (comma-separated list of string months. ex. "Feb,May,Aug,Nov")

Table 2 describes fields in store.csv.

The training set is composed of two different parts: one is ĩ0 million observed daily sales of different stores from 2013 to 2015, and the other is supplementary information for each store. We merged the two different parts by stores to have the training set.

Here are the set of features with explanation:

1. Store - unique id for each store (positive integer ranging from 1 to 1115)

The training set had originally 15 features, but we extended to 19. We separated out year, month of the year, and day of the month from the feature d̈ate.

Also, we computed average sales for each store. In addition, we noticed the feature C̈ompetitionDistanceïs highly skewed to right. For the sake of linear regression, we decided to take log transformation on the feature C̈ompetitionDistanceïo make sure that the relationship between explanatory variables and dependent variable is approximately linear.

Here are the set of features:

1. fdsfsd

Because of several missing values in the training set, we filled them based on our logic. - There are some stores on some date which do not have any competitors. In this case, the competitor information

such as C̈ompetitionDistanceör this specific observation are missing. Therefore, we assigned some big number as 1000000 for C̈ompetitionDistanceötо weaken its impact. In the similar way, we filled missing values for C̈ompetitionOpenSinceYeraänd CompetitionOpenSinceMonth - There are some stores on some date which are not in the continuing promotion. In this case, the feature P̈romo2ïs 0, and Promo2Since

$$Year/Week$$

are missing. We filled them with the lowest possible value to weaken their impact.

2.2 Exploratory Data Analysis

To check any distinctive relationships between variables, we drew a pearson correlation coefficient plot using corrplot library. There are several interesting facts we captured. - D̈ayOfWeekänd S̈alesäre negatively correlated. This implies that, as D̈ayOfWeekäpproaches to Saturday and Sunday, sales would decrease because most drugstores would close on these days. - L̈ogCompetitionDistanceänd S̈alesäre negatively correlated. In other words, lower distance to the competitor implies slightly higher sales. We assume this could occur because stores with a close distance to the competitor are mostly located in crowded regions such as a downtown city with higher sales overall.

3 Methods

We started off by setting a benchmark to obtain a baseline for the prediction. Then, we took 3 different approaches to outperform the benchmark and former model.

3.1 5-fold Cross Validation

To generalize our predictions to limit problems of overfitting on the training set, we decided to ramdomly partition the original training observations into 5 equal sized subsamples. The training set is comprised of 4 equal sized subsamples and the rest as the validation set. For the purpose of reproducibility and comparison of the predictive powers of different models, we set the seed (with a value of 42). Our general process would be as the following: 1. Repeat cross-validation process 5 times with each of the 5 subsamples used exactly once as the validation set. 2. Compute the average of 5 different validation error rate. 3. Predict sales on the test set using the model. 4. Submit on Kaggle website to get the test error rate. 5. Compare the average validation error rate and the test error rate among differnent models.

3.2 Error Metric

As the metrics that Kaggle uses to compute test error is the Root Mean Squared Percentage Error (RMSPE), we used the same metrics to compute our validation error. It's the following:

$RMSPE = \sqrt{\frac{1}{n}\sum(\frac{y_i - \hat{y}_i}{y_i})^2}$

where y_i is a store's sales on a single day, $yhat_i$ is the daily sales prediction for the store on the day, and n is the number of (store, date) pairs in the data set.

3.3 Benchmark

Our benchmark is simply the average sales of each store. Precisely, we predict daily sales of a store for a day by the average daily sales the store had in the training data set. This method does not account for many factors (whether the day was holiday, whether the store was running a promotion on the day, etc), so we expected its error rate to have a higher error rate than any other methods we applied. This method had a the validation error rate of TODO and the test error rate of TODO.

3.4 Linear Regression

To apply linear Regression on the data, we decided to use R's *lm* function. *lm* is an implemention of multivariate linear regression. If we supply $Y \sim X$ as the input, *lm* builds and returns a model that can be used to predict Y using X. Under the hood, it basically solves the normal equation $\beta = (X^T X)^{-1} X^T Y$.

We first attempted to use all features available. With all features (Holiday, ..., promo2), we obtained a validation error rate of TOD and the test error rate of TODO.

3.4.1 Feature Selection

Before working on feature select, we took a look at the resulting lm with R function $summary(lm)$ to get some general idea of how important each variable is. DayOfWeek, Open, Promo, StateHoliday, SchoolHoliday, StoreType, Assortment, Promo2SinceWeek, Average.Sales, LogCompetitionDistance, day, month, and year seemed to have high t-value with p-value less than 0.05. This intuitively made sense because each variable had some ways to affect daily sales:

1. DayOfWeek: On Sunday,
2. StateHoliday: On a holiday, it makes sense that more people come to shop. Indeed, the coefficient is TODO.
3. sdfas

To remove useless features, we first applied Akaike information criterion (AIC) by applying R function *stepAIC*. However, this failed to remove any features, so decided to manually remove features with p value more than 0.05. This resulted in a linear model with a validation error rate of TODO and test error rate of TODO.

3.4.2 Lasso

3.4.3 Ridge

3.5 Random Forest

Random Forest is an ensemble learning method that constructs decisions tree using training set. By aggregating many decision trees, Random Forest can make predictions while avoiding overfitting. Here is our Random Forest Regression steps:
1. Construct random forest trees (number of trees is defined by hyper parameter) 2. Classify the incoming data into the decision tree node 3. Calculate the mean value for each node for prediction In particular, we used H2Os RandomForest package to carry out the training and prediction. Then, we optimized parameters to improve on our prediction model. H2O is the open source math engine for huge dataset (TODO). Since H2O package enables us to compute Random Forest Regression algorithm in parallel distribution, we decided to use the package for faster computation compared to R's Random Forest package.

TODO HYPER PARAMETERS

3.6 Gradient Boosting

The similarity between Random Forest and Gradient Boosting is that both generate decision trees. However, Random forest uses bootstrapping method for training while Gradient Boosting builds decision tree over the residuals. In other words, Gradient Boosting Tree generates the decision tree as it optimizes on a proper loss function ( ). This refers that we can establish our own loss function for optimization, which we really couldn't do in Random Forest. Our decision trees here are regression trees. Each regression tree divides the data point at each non-leaf node according to a constraint on a feature. The leaf node value of a data point decides its score, and a data point is predicted by averaging the scores of the leaf node it belongs to.

In particular, we used R's xgboost package. XGBoost also known as Extreme Gradient Boosting is a powerful supervised learning algorithm (). This package enables an automatic parallel computation on a single machine. For regression, we put r̈eg:linear.̈ Our training objective here is :

4 Results

4.1 Random Forest

The two main parameters we tuned for RF is the number of trees and the size of the random subsets of features to consider when splitting a node. We used 5 fold cross validation to get RMSPE while varying these parameters. Two plots are shown below. From these plots, we could see that RMSPE doesnt change too much after tree number reaches 30 and after feature number reaches 20.

4.2 Gradient Boosting

Unfortunately, Gradient Boosting Tree has its own drawbacks. First, It is more likely to overfit than random forest. It trusts every data point and tries to find optimal linear combination of trees given the train data. Second, there are more parameters in Gradient Boosting Tree. We have to tune the number of trees, maximum depth and the learning rate at the same time. For the first drawback, we can try to avoid over- fitting by using the cross validation provided by sklearn. For the second method, we have to spend more time to tune it, which is tradeoff between time and accuracy

One of the major problems of all machine learning algorithms is to know when to stop, i.e., how to prevent the learning algorithm to fit esoteric aspects of the training data that are not likely to improve the predictive validity of the respective model. This issue is also known as the problem of overfitting.

The gradient boosted trees model is a type of additive model that makes predictions by combining decisions from a sequence of base models. More formally we can write this class of models as:

g(x)=f0(x)+f1(x)+f2(x)+...

g(x)=f0(x)+f1(x)+f2(x)+...

Random forest runtimes are quite fast, and they are able to deal with unbalanced and missing data. Random Forest weaknesses are that when used for regression they cannot predict beyond the range in the training data, and that they may over-fit data sets that are particularly noisy.

Now we're going to cite somebody. Watch for the cite tag. Here it comes [**?**, **?**]. The tilde character (˜) in the source means a non-breaking space. This way, your reference will always be attached to the word that preceded it, instead of going to the next line.

5 Supplementary Methods

5.1 First SubSection

Here's a typical figure reference. The figure is centered at the top of the column. It's scaled. It's explicitly

Figure 1: Wonderful Flowchart

placed. You'll have to tweak the numbers to get what you want.

This text came after the figure, so we'll casually refer to Figure 1 as we go on our merry way.

5.2 New Subsection

It can get tricky typesetting Tcl and C code in LaTeX because they share a lot of mystical feelings about certain magic characters. You will have to do a lot of escaping to typeset curly braces and percent signs, for example, like this: "The `%module` directive sets the name of the initialization function. This is optional, but is recommended if building a Tcl 7.5 module. Everything inside the `%{, %}` block is copied directly into the output. allowing the inclusion of header files and additional C code."

Sometimes you want to really call attention to a piece of text. You can center it in the column like this:

`_1008e614_Vector_p`

and people will really notice it.

The noindent at the start of this paragraph makes it clear that it's a continuation of the preceding text, not a new para in its own right.

Now this is an ingenious way to get a forced space. `Real *` and `double *` are equivalent.

Now here is another way to call attention to a line of code, but instead of centering it, we noindent and bold it.

```
size_t :  fread ptr size nobj stream
```

And here we have made an indented para like a definition tag (dt) in HTML. You don't need a surrounding list macro pair.

> `fread` reads from `stream` into the array `ptr` at most `nobj` objects of size `size`. `fread` returns the number of objects read.

This concludes the definitions tag.

5.3 How to Build Your Paper

You have to run `latex` once to prepare your references for munging. Then run `bibtex` to build your bibliography metadata. Then run `latex` twice to ensure all references have been resolved. If your source file is called `usenixTemplate.tex` and your `bibtex` file is called `usenixTemplate.bib`, here's what you do:

```
latex usenixTemplate
bibtex usenixTemplate
latex usenixTemplate
latex usenixTemplate
```

5.4 Last SubSection

Well, it's getting boring isn't it. This is the last subsection before we wrap it up.

6 Results

7 Supplementary Results

A polite author always includes acknowledgments. Thank everyone, especially those who funded the work.

8 Discussion

It's great when this section says that MyWonderfulApp is free software, available via anonymous FTP from

`ftp.site.dom/pub/myname/Wonderful`

Also, it's even greater when you can write that information is also available on the Wonderful homepage at

`http://www.site.dom/~myname/SWIG`

Now we get serious and fill in those references. Remember you will have to run latex twice on the document in order to resolve those cite tags you met earlier. This is where they get resolved. We've preserved some real ones in addition to the template-speak. After the bibliography you are DONE.

9 References Cited

Dehkordi-Vakil, Farideh. ”Multiple Regression Analysis.” DS-533. Illionis, Macomb. Lecture.

Breiman, L., (2001a). Random Forests, Machine Learning 45 (1) pp. 5-32. R.D. https://www.diva-portal.org/smash/get/diva2:785776/FULLTEXT01.pdf

https://documents.software.dell.com/statistics/textbook/boosting-trees-regression-classification

h2o package description

10 Author Contributions