DSC440 Final Project Report:

Yonghao Duan

Georgen Institute for Data Science
University of Rochester
P.O. Box 270125
United States
yduan14@ur.rochester.edu

Yankun Gao

Georgen Institute for Data Science
University of Rochester
P.O. Box 270125
United States
ygao55@ur.rochester.edu

1 Introduction

1.1 Project choice and rational

The project is about the prediction of store sales of Rossmann, which operates thousands of drug stores in European countries [1]. The daily sales of each Rossmann store in Germany will be predicted for up to six weeks in advance. A robust prediction model will greatly help the store managers to create effective staff schedules that increase productivity and motivation.

We are interested in the project due to the following reasons. Firstly, compared to the housing price prediction project (listed on the blackboard), this one is a real-world problem, and analyzing a real-world case is always exciting. Secondly, this is a huge data set, and it contains lots of features. Dealing with all kinds of features can help us to learn better how to make predictions of store sales. Thirdly, since this project is a competition we got online, our result can be compared with the results from other excellent data scientists which will urge us to work harder to get a better prediction score.

1.2 Team composition

Yonghao Duan and Yankun Gao from the graduate program of Data Science are the two members of our team. In this project, both members will work together to finish the process of data preprocessing, exploratory data analysis, feature engineering and modeling and final report. The contribution to the final project will be equal.

1.3 Goal of analysis

The goal of this project is to create a prediction model which can accurately predict stores daily sales for up to six weeks in advance for 1115 stores located across Germany. The robust prediction model can help store managers to stay on the most important things.

The evaluation formula of the result is the Root Mean Square Percentage Error (RMSPE) shown below [1].

$$\mathrm{RMSPE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{y}_i}{y_i}\right)^2},$$

where y_i denotes the sales of a single store on a single day and $\hat{y}_i$ denotes the prediction values.

2 Related Work

2.1 Boosting

Boosting is an ensemble method that create a strong classifier based on weak classifiers, according to how correlated are the learners to the actual target variable. The errors of the previous model are corrected by the next predictor, by adding models on top of each other iteratively until the training data is accurately predicted or a maximum number of models are added. This process consists three steps: (1) an initial model F0 is defined to predict target variable y (2) a new

model h1 is fit to the residuals (y-F_0) from last step (3) F_1, the boosted version of F_0 is generated by combining F_0 and h_1. Therefore, the mean squared error from F_1 is lower than that from F0: $F_1(x)$ <- $F_0(x) + h_1(x)$. This can be repeated for *m* iterations until residuals are maximumly minimized: $F_m(x)$ <- $F_{m-1}(x) + h_m(x)$.

2.2 AdaBoost

AdaBoost is used for classification other than regression. It is commonly used on binary classification problems to boost the performance of decision trees. AdaBoost is most suited with decision trees having only one level, since AdaBoost is best used with weak learners. These models achieve accuracy just above random chance on a classification problem.

Each instance in the training dataset is weighted and the weight is initially set to: weight(x_i) = 1/n, where x_i denotes the i'th training instance and n denotes the number of training instances. In binary classification problems, each weak learner assigns an input either 1.0 or -1.0 for the first- or second-class value. The misclassification rate (error) is calculated as: error = (correct – N)/N, where correct denotes the number of training instance predicted correctly by the model and N denotes the total number of training instances. To modify the misclassification rate by using the weighting of the training instances: error = sum(w(i)*terror(i))/sum(w), where w denotes the weight for training instance i and terror denotes the prediction error for training instance I (1 if misclassified; 0 if correctly classified).

A stage value for a trained model which provides a weighting for predictions is calculated as: stage = ln((1-error)/error), where ln () is the natural logarithm. The training weights of training instances are updated by formula: w = w*exp(stage*terror), where terror denotes the error the weak classifier made on prediction (terror = 0 if y == p, otherwise 1).

To make predictions with AdaBoost, each weak learner calculates a predicted value for a new input instance as either 1.0 or -1.0. The sum of the weighted predictions is calculated. If sum>0, then the first class is predicted, otherwise the second class is predicted. We need to keep in mind that this algorithm is highly affected by outliers since it tries to fit every point perfectly.

3 Methodology

3.1 Linear regression

Linear regression is a type of widely used predictive analysis. It is used to determine whether an outcome variable can be predicted by a set of predictor variables. It can also be used to quantify the strength of relationship between one dependent variable and one or multiple independent variables. Given a data set $\{y_i, x_{i1}, \ldots, x_{ip}\}_{i=1}^n$ of n statistical units, the linear model takes the form

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \ldots + \beta_p X_{ip} + \varepsilon_i, \qquad i = 1, \ldots, n$$

,where ε indicates a random variable that adds "noise" to the linear relationship between predictor variables and outcome variable. In order to summarize the trend between two quantitative variables, we need to choose the "best fitting line" based on the "least squares criterion":

$$\sum_{i=1}^{n} \varepsilon_i^2 = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2$$

$$\text{argmin}\beta \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2$$

where y_i denotes the observed response for experimental unit *i*.

When we have a high dimensional data set, using all the variables would be highly inefficient since some variables may contain redundant information. Therefore, we need to select the right set of variables which will give us an accurate model. To do this, we can use either forward selection, which starts with most significant predictors and adds variable for each step, or backward selection, which starts with all predictors and removes the least significant variable for each step. R-square can be used as a selecting criterion.

Some time, compared to the simple linear equation, the polynomial equation fits the data better. But it is not the higher order polynomials the better, since this would cause a problem called over-fitting which indicates that our training model cannot generalize on the new data.

To overcome overfitting, we can either reduce the model complexity or do regularization. The common ways to do regularization include Ridge regression, LASSO regression and Elastic regression [2]. All these methods try to penalize the Beta coefficients to help us to select the important variables. The ridge regression

$$\sum_{i=1}^{M}(y_i-\hat{y}_i)^2=\sum_{i=1}^{M}\left(y_i-\sum_{j=0}^{p}w_j\times x_{ij}\right)^2+\lambda\sum_{j=0}^{p}w_j^2$$

uses L2 regularization technique and alters the cost function by adding the square of the magnitude of the coefficients as a penalty. The LASSO regression

$$\sum_{i=1}^{M}(y_i-\hat{y}_i)^2=\sum_{i=1}^{M}\left(y_i-\sum_{j=0}^{p}w_j\times x_{ij}\right)^2+\lambda\sum_{j=0}^{p}|w_j|$$

uses L1 regularization technique and alters the cost function by adding the magnitudes of coefficients as a penalty. Elastic net regression is a combination of both L1 and L2 regularization which uses both L1 and L2 penalty terms. The elastic net penalty is described as $J(\beta)=\alpha\|\beta\|^2+(1-\alpha)\|\beta\|_1$, where $\alpha=\frac{\lambda_2}{\lambda_2+\lambda_1}$. Elastic net regression can stabilize the L1 regularization and remove the limitations of L1 regularization on number of variable selection and grouping effect.

3.2 XGBoost

EXtreme Gradient Boosting (XGBoost) is an ensemble machine learning method, which has been very popular since its introduction in 2014. It is a powerful tool that can work through most classification, regression and ranking problems. Different from Adaboost which is mainly for classificatons with exponential loss, XGBoost works for generic loss functions and it has more customizable parameters. XGBoost is a single model which gives the aggregated output by combining the predictive power of multiple learners. The individual machine learning model that form the ensemble are known as base learners and they are either from the same learning algorithm or different learning algorithms.

XGBoost follows the principle of gradient boosting [3]. In gradient boosting, we first define the F0(x) function to initialize the boosting algorithm:

$$F_0(x)=argmin_\gamma\sum_{i=1}^{n}L(y_i,\gamma),$$

then the gradient of the loss function is computed iteratively:

$$r_{im}=-\alpha\left[\frac{\partial(L(y_i,F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)},$$

where α denotes the learning rate. At each step, each hm(x) is fit on the gradient and the boosted model $F_m(x)$ is defined:

$$F_m(x)=F_{m-1}(x)+\gamma_m h_m(x),$$

where γm denotes the multiplicative factor for each terminal node.

While XGBoost is similar to gradient boosting algorithm, it has some unique features which make it so interesting. XGBoost can penalize complex models through both L1 and L2 regularization to prevent overfitting; XGBoost can handle different types of sparsity patterns in the data; XGBoost can effectively handle weighted data; XGBoost can make use of multiple cores on the CPU to allow parallel learning. Generally, compared to gradient boosting, XGBoost is fater and has a wider range of application [4].

4 Experiment

The processes of data science contain data preprocessing, exploratory data analysis (EDA), feature engineering and modeling. For data preprocessing, the python packages which contains numpy, pandas, matplotlib and seaborn are used.

4.1 Data overview

4.1.1 Features

In this project, three tables containing different store information are given. Table train contains 1017209 samples and 9 features: Store, DayOfWeek, Date, Sales, Customers, Open, Promo, StateHoliday and SchoolHoliday. Table test contains 41088 samples and 8 features: Id, Store, DayOfWeek, Date, Open, Promo, StateHoliday and SchoolHoliday. Table store contains 1115 samples and 10 features: Store, Storetype, Assortment, CompetitionDistance, CompetitionOpenSinceMonth, CompetitionOpenSinceYear, Promo2, Promo2SinceWeek, Promo2SinceYear and PromoInterval. The meanings of differenct features are listed below:

- Id - an Id that represents a (Store, Date) duple within the test set
- Store - a unique Id for each store
- Sales - the turnover for any given day (this is what you are predicting)
- Customers - the number of customers on a given day
- Open - an indicator for whether the store was open: 0 = closed, 1 = open
- StateHoliday - indicates a state holiday. Normally all stores, with few exceptions, are closed on state holidays. Note that all schools are closed on public holidays and weekends. a = public holiday, b = Easter holiday, c = Christmas, 0 = None
- SchoolHoliday - indicates if the (Store, Date) was affected by the closure of public schools
- StoreType - differentiates between 4 different store models: a, b, c, d
- Assortment - describes an assortment level: a = basic, b = extra, c = extended
- CompetitionDistance - distance in meters to the nearest competitor store
- CompetitionOpenSince[Month/Year] - gives the approximate year and month of the time the nearest competitor was opened
- Promo - indicates whether a store is running a promo on that day
- Promo2 - Promo2 is a continuing and consecutive promotion for some stores: 0 = store is not participating, 1 = store is participating
- Promo2Since [Year/Week] - describes the year and calendar week when the store started participating in Promo2
- PromoInterval - describes the consecutive intervals Promo2 is started, naming the months the promotion is started anew. E.g. "Feb,May,Aug,Nov" means each round starts in February, May, August, November of any given year for that store

4.1.2 Missing values

We first describe the min and max values of each feature and counted the missing values by using pandas, data.isnull().sum(). We find there are missing values for the seven features listed below.

Table 1. Summarization of missing values of the data set.

Attributes	No. of Missing Values
CompetitionDistance	3
CompetitionOpenSinceMonth	354
CompetitionOpenSinceYear	354
Promo2SinceWeek	544
Promo2SinceYear	544
PromoInterval	544
Open	11

4.1.3 Distribution of sales

In this project, we are required to predict the sales, which is a time-series feature. Therefore, we plot

graphs to see the overall distribution of sales and the changing of sales over time.

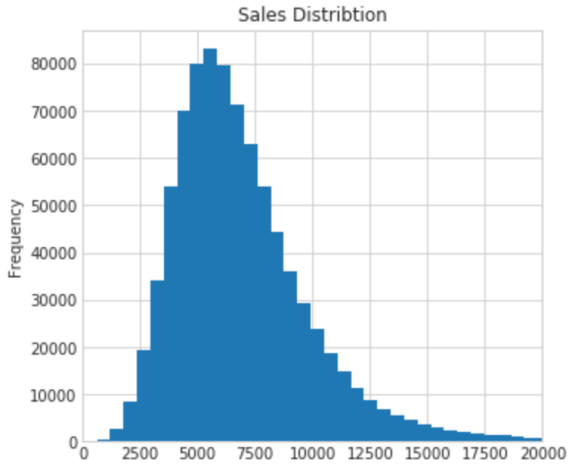

Fig 1. The distribution of Sales values. The Sales values are ranging from less than 2500 to around 20000. The distribution is skewed to the right.

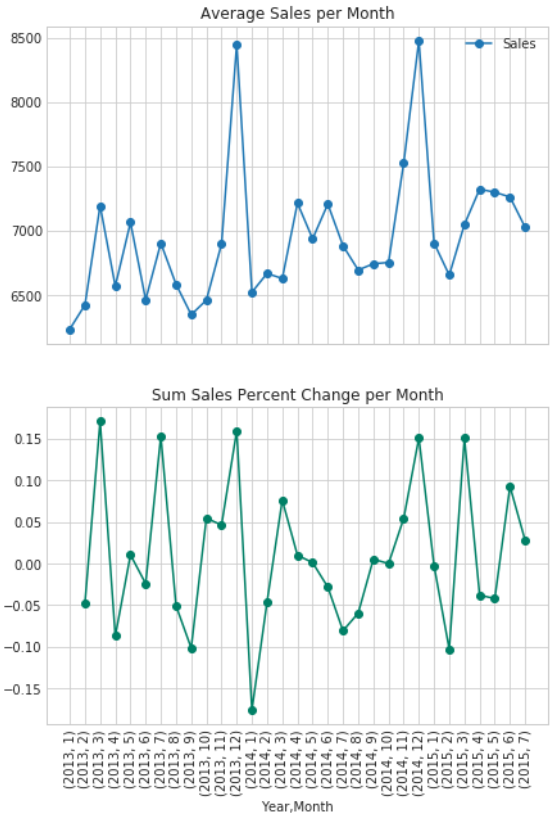

Fig 2. Changing of Sales values over time. The upper picture shows the changing of average Sales per month, and the lower picture shows the percentage change of sum Sales per month.

4.1.4 Different features and sales

We plot graphs to show the relationship between sales and other individual features by using different visualization functions of matplotlib and seaborn packages. The features contain DayOfWeek, Promo, Promointerval, StateHoliday, SchoolHoliday, StoreType, Assortment and CompetitionDistance.

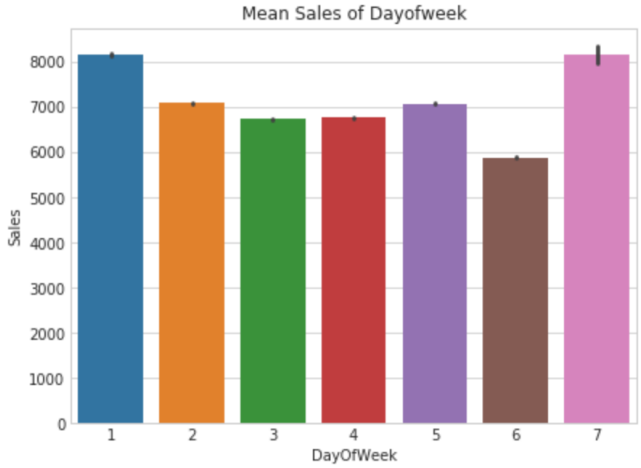

Fig 3. Relationship between mean Sales and DayOfWeek. The mean sales on different days of a week are similar.

Fig 4. Relationship between mean Sales and StateHoliday. The mean sales on different types of stateholidays are all between 6000 and 10000 dollars.

Fig 5. Relationship between mean Sales and SchoolHoliday. The mean sales on stateholidays and non-stateholidays are similar.

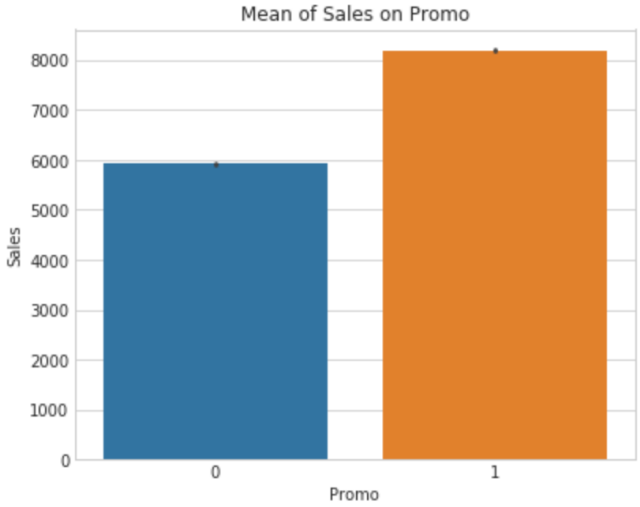

Fig 6. Relationship between mean Sales and Promo. The mean sales on Promo are higher than the mean sale not on Promo.

4.2 Feature preprocessing

When we do the feature preprocessing, we first combined the train data and test data together since they have several features in common. We create a new feature Set to the dataframe and let train.Set = 1 and test.Set = 0. Therefore, after we finish feature preprocessing, we can separate the train data and test data easily according to their Set values.

4.2.1 Handling of missing data

First, the missing Open data are filling with zero based on official notification [5]. Second, we assume there is no competition and filled the missing values of competitionDistance, CompetitionOpenSinceMonth and CompetitionOpenSinceYear with zero. Third, missing values for Promo2SinceWeek, Promo2SinceYear and PromoInterval indicate that those stores were not participating in Promo2 at that time, and therefore, we fill those missing values with zero.

4.2.2 Log scale of sales data

The sales data varies through a wide range and the distribution is skewed. To improve model fits and accuracy, we do log transformation to the sales data to minimize variation.

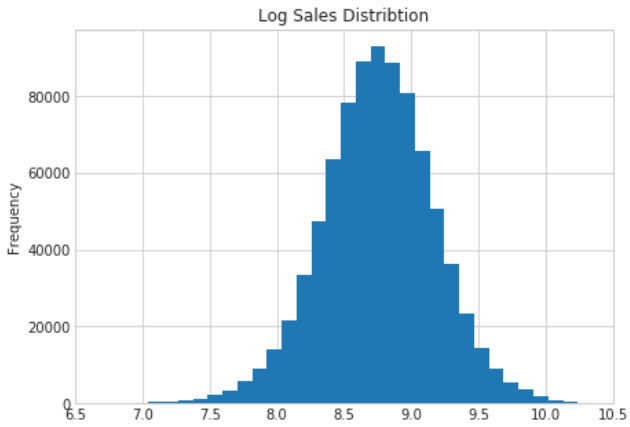

Fig 7. Log-transformation of Sales values. The log-transformed Sales values have a normal distribution.

4.2.3 Dealing with time-series features

Since this is a time-series-based-prediction project, the features related to time are very important. The Date feature is stored in YYYY-MM-DD format. Using *pandas* library, we split the feature Date into several new features: Year, Month, Day, WeekOfYear and Season.

Based on the features Year, CompetitionOpenSinceYear, Month and CompetitionOpenSinceMonth, we calculate and generate a new feature CompetitionOpen, which indicates how long the competition stores have been opened. The calculation formula is 12 *

(data.Year – data.CompetitionOpenSinceYear) + (data.Month – data.CompetitionSinceMonth).

Similar with the feature CompetitionOpen, a new feature PromoOpen is generated by using features Year, Promo2SinceYear, WeekOfYear and Promo2SinceWeek. The formula used is 12 * (data.Year – data.Promo2SinceYear) + (data.WeekOfYear – data.Promo2SinceWeek) / 4.0. For the stores whose Promo2SinceYear values are zero, the PromoOpen value are filled with zero.

4.2.4 Handling of categorical features

There are three categorical features: StoreType, Assortment and StateHoliday. To train the linear regression model, we convert these categorical features into dummy variables. One-hot encoding is used to transform the string variables to numeric values. Therefore, in the new dataframe, we drop the previous categorical features and add new dummy variables to train the bench mark model: StoreType_b, StoreType_c, StoreType_d, Assortment_b, Assortment_c, StateHoliday_a, StateHoliday_b and StateHoliday_c.

In addition, the PromoInterval feature is transformed to dummy variable, by setting all the NA values as a separate group.

After we finish the feature preprocessing, all the feature types are transferred to 'int16' to save memory.

4.3 Modeling

4.3.1 Benchmark: Linear regression

This project asks the prediction of sales based on previous data. We choose the linear regression model [6]. The features we use for this basic model are Store, DayOfWeek, Prome, SchoolHoliday, Year, Month, Day, WeekOfYear, Season, CompetitionDistance, Promo2, CompetitionOpen, PromoOpen, IsPromoMonth, StoreType_b, StoreType_c, StoreType_d, Assortment_b, Assortment_c, StateHoliday_a, StateHoliday_b and StateHoliday_c.

To avoid overfitting, we choose to do Elastic Net regularization, which uses both L1 and L2 penalty. For l1_ratio between 0 and 1, the penalty is a combination of L1 and L2, while l1_ratio=0 indicates L2 penalty and l1_ratio=1 indicates L1 penalty. We set a list of alphas values: 0.05, 0.005 and 0.0005. Accordingly, we set a list of three l1_ratio values. Since a good choice of list of values for l1_ratio is often to put more values close to 1 and less close to 1, we choose 0.7, 0.8 and 0.9.

While there is a list of l1_ratio values other than one, different values will be tested by cross-validation and the one giving the best prediction score will be chosen. Therefore, we set 'cv' value as 10, which means it is a 10-fold cross-validation.

To train the model, we use the data of opening stores (data['Open']==1) from table train (data['Set']==1). When test model, we use data from table test by selecting data['Set']==0. The result is evaluated by the RMSPE formula which is described in Python function. The testing score is 0.41635, which is not good. Therefore, we need to find a better model.

4.3.2 XGBoost

To improve competition scores, we use the gradient boosting algorithm (XGBoost), which provides a parallel tree boosting that solve many data science problems in a fast and accurate way [7].

4.3.2.1 Initial training with XGBoost

We split the data of opening stores from table train into training and validation two parts by setting 'test_size=0.015'. The parameter 'random_state' is set to a fixed value. The initial parameter setting are: 'objective':'reg:linear', 'booster':'gbtree', 'eta':0.01, 'max_depth':10, 'subsample':0.9, 'colsample_bytree':0.7, 'silent':1, 'seed':seed.

We apply the trained model to test data and get their predicted log-value of sales ('data_ypred'). The log value is transferred to regular numbers by using formula data_ypred['sale'] = (np.exp(ypred_bst)-1). Then the rmspe value is calculated and we get 0.11320 for this model, which is greatly improved compared to the linear model.

All features are recursively added into the model to test their importance and we get a list of feature importance from there.

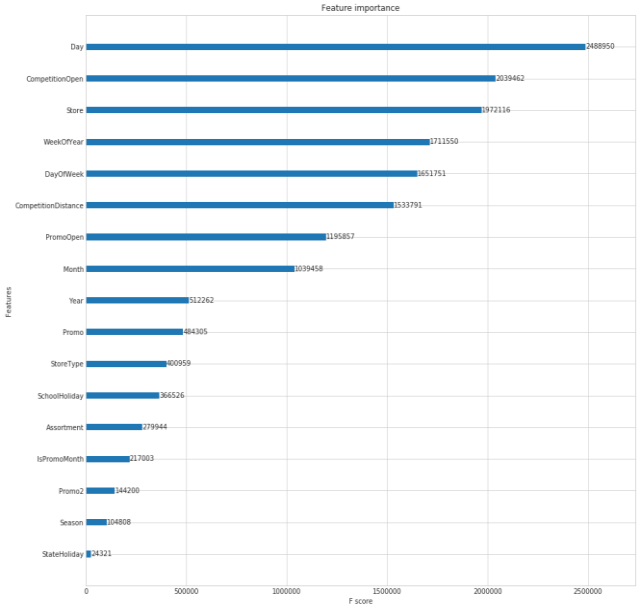

Fig 8. Feature importance list. The y-axis shows the names of different features and the x-axis indicates the importance of those features.

4.3.2.2 Adding more features

Based on the result of explorary data analysis, new features are created by combining features of Store, DayOfWeek, Promo and Year. We add MeanSales, MedianSales, MeanCustomers, MedianCustomers to the original feature set. State information of each store is added to the dataframe.

4.3.2.3 Handling of outliers

We define outlier based on median absolute deviation (MAD) for each store. All the outliers are assigned a distinguish label to be separated from the non-outliers.

The values of outliers are replaced by the predicted values getting from our model.

4.3.2.4 Tuning XGBoost parameters

The objective function for XGBoost is listed below.

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Training loss — Complexity of the Trees

The regularization is listed below.

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Number of leaves — L2 norm of leaf scores

The original parameters we use are:

Parameter	Values
booster	gbtree
eta	0.3
max_depth	13
cols_sample	0.9
subsample	0.7
train_test_split	trian_test_split
seed	42

while 'booster', 'train_test_split' and 'seed' have fixed values, we want to tune the values of the other four parameters to get a better result.

Parameter description:

1. eta: learning rate to prevents overfitting

2. max_depth: the max depth of the tree

3. cols_sample: the percentage of features can be chosen

4. subsample: the percentage of samples can be chosen

5. seed: used to initializate parameters so that you can repeat your model result

First, we use a fixed value 0.3 for 'eta' to have a higher learning rate and try to find the best 'max_depth' quickly. We initialize the range of 'max_depth' from 6 to 15 with a step of 2.

max_depth	eval-rmspe
6	0.095637
8	0.090162
10	0.088746
12	0.089783
14	0.091024

From the result above, we can get the least eval-rmspe value when max_depth is 10.

Sedond, using the fixed 'eta' and 'max_depth' value, we try to figure out the best values for 'subsample' and 'colsample_bytree'. We initialize a set of values inclusing 0.7, 0.8 and 0.9 for both parameters.

subsample	colsample_bytree	eval-rmspe
0.7	0.7	0.09119
0.7	0.8	0.089963
0.7	0.9	0.089316
0.8	0.7	0.089537
0.8	0.8	0.089607
0.8	0.9	0.089136
0.9	0.7	0.088746
0.9	0.8	0.089186
0.9	0.9	0.089178

We find 'subsample'=0.9 and 'colsample_bytree'=0.7 give us the least eval-rmspe value.

Next, we use the optimized features to tune 'eta' value. We choose a set of 'eta' values: 0.3, 0.1, 0.05, 0.02 and 0.01 to start the training.

eta	eval-rmspe
0.3	0.09041
0.1	0.086487
0.05	0.085552
0.02	0.084729
0.01	0.084853

We find lower 'eta' values, such as 0.02 and 0.01, give us better eval-rmspe results. We choose 0.01 as the final value for 'eta'.

4.3.2.5 Selecting of train data splitting

Our goal is to predict the sales between 2015-08-01 and 2015-09-17. We choose three candidate splitting point 2015-06-01, 2015-07-01 nad 2015-07-15 to split our training data.

Time Split Point	RMSPE Val	RMSPE Train
2015-06-01	0.138664	0.113628
2015-07-01	0.115334	0.086579
2015-07-15	0.098203	0.069941

We find splitting our data at 2015-07-15 gives us the best rmspe score.

4.3.2.6 Final settings of parameters

Finally, basic stacking technique is used to combine the results of models in different random seed to achieve best competition score. Based on all the tuning processing above, we finalized all the parameter settings for the XGBoost model. Using these settings, we finally get a rmspe value of 0.10878.

Parameter	Number
eta	0.02
max_depth	10
cols_sample	0.9
subsample	0.9
Time Split	2015-07-15

5 Conclusion

Different models are evaluated by using the RMSPE formula shown in the Introduction. We get RMSPE=0.41635 for the linear regression model and RMSPE=0.10878 for the XGBoost model. Therefore, the XGBoost model works much better than the linear regression model on sales prediction. In this competition, our result is the 12th best out of 3303 groups competitors.

Furthermore, other than obtaining good sales prediction, feature importance is also exhibited in our project. The information from the feature importance table allow the store managers to efficiently arrange their work based on those critical factors.

6 Role of Team Members

Team members have equal contribution to this project. The specific task arrangement is listed below:

Task	Yankun Gao	Yonghao Duan	Timeline
Proposal	40%	60%	11/8
Data Preprocessing	60%	40%	11/15
Feature Engineering	40%	60%	11/20
Modeling and Stacking	40%	60%	11/31
Report	60%	40%	12/10

7 References

[1]: https://www.kaggle.com/c/rossmann-storesales

[2]: https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/

[3]: https://hackernoon.com/gradient-boosting-and-xgboost-90862daa6c77

[4]: https://www.kaggle.com/c/rossmann-store-sales/discussion/17048#96969

[5]: https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/

[6]: Chen T, Guestrin C. XGBoost: A Scalable Tree Boosting System[J]. 2016:785-794.

[7]: Liaw A, Wiener M. Classification and Regression by randomForest[J]. R News, 2002, 23(23)