425 Final Report

Predicting Walmart Sales in Stormy Weather

Yuqi Zhang (yuqi6)

Shuyu Jia (shuyuj2)

Jing Huang (jingh7)

Ziang Zhao (ziang3)

1. Introduction:

Data

Walmart is the world's largest company with more than 514 billion dollars in revenue. The big data sets it collects are mined for use in predictive analytics, which allows the company to optimize operations by predicting customer's habits. (McCoy, Max, 2006) Extreme weather events may have a large impact on sales both at the store and product level. We may expect that items like umbrellas have a larger possibility to increase in sales before or even during a big storm. However, it seems difficult for replenishment managers to correctly predict the sales of weather-sensitive items and be well-prepared not suffering from out-of-stock or overstock when extremes weather conditions occur. Therefore, predictive analytics and time-series methods are highly recommended for demand forecasting on independent products (Cheikhrouhou et al., 2011; Li et al., 2012).

We have been provided with sales data from Walmart on Kaggle platform with stores and dates in the training set, as well as NOAA weather data of nearest climate stations of the stores. The sales data from Walmart contains 45 stores in different locations and each store has 111 weather-sensitive products whose sales may potentially be affected by the weather condition (such as umbrellas, bread, and milk). The weather data was gathered from 20 different weather stations covered 45 locations. Some of the stores are nearby and share a weather station.

Goal

Our goal is to predict the units of each product sold around the time (±3 days) of extreme weather events. For the purposes of this analysis, the extreme weather event was defined as any day in which more than an inch of rain or two inches of snow was observed.

Methods

In this report, we will use both linear regression and nonlinear regression methods to predict the Walmart sales data. For linear regression, we use ordinary least square algorithm together with a log transformation. We also use normalization linear regression

like Ridge and Lasso, and a combination of random forest classification. Our best prediction score for this part is 0.1192.

For nonlinear models, we use random forest regression and a light GBM algorithm. We will explain the concepts of these two models and how we tune the parameters. Best prediction score for this part is 0.1161.

2. Data Preprocessing

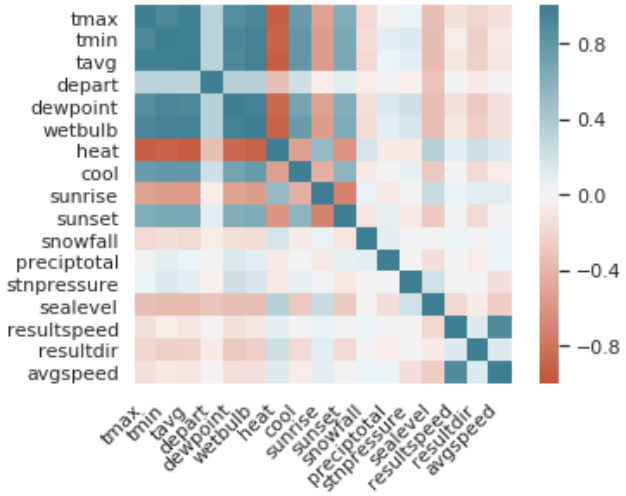
Data cleaning

The weather dataset contains 20 weather variables observed by 20 different stations from 2012 to 2014. These variables include different forms of entries, and many of them are not strings. Our first step is changing all entries that are not numerical data to numerical forms.

We run a basic check to see what kinds of strings are included in these variables. There are four types of non-numerical values, 'M', '-', whitespace and 'T'. The NOAA weather documentation indicates that M means 'none', T means 'a little', '-' as well as whitespaces are missing values. We believe that the weather of two successive days are similar, so we implement a forward filling method. We change all of the strings to NA, and then use the first non-NA value in previous days to fill following NA values.

We found that there are two abnormal data in our training dataset (units > 4000), while others are lower than 500. We believe that

	if M	if -	if whitespace	if T
station_nbr	False	False	False	False
date	False	False	False	False
tmax	True	False	False	False
tmin	True	False	False	False
tavg	True	False	False	False
depart	True	False	False	False
dewpoint	True	False	False	False
wetbulb	True	False	False	False
heat	True	False	False	False
cool	True	False	False	False
sunrise	False	True	False	False
sunset	False	True	False	False
codesum	False	False	True	False
snowfall	True	False	False	True
preciptotal	True	False	False	True
stnpressure	True	False	False	False
sealevel	True	False	False	False
resultspeed	True	False	False	False



these two observations will severely influence our prediction accuracy, so we drop them.

Feature Extraction

1. Weather dataset

There is duplicated information contained by the weather dataset, especially in first several features. For example, the tmax, tmin and tavg all indicate the overall temperature of a day. For model efficiency, we only need one of those linearly correlated features.

Also, since we are predicting the sales under extreme weather events, for training data we need information that indicates whether or not for any specific station and day there is extreme weather. Binary variables would be better than numerical variables for this purpose. As a result, we convert *preciptotal* and *snowfall* to binary variables. We drop the *codesum* variable, which indicates the specific type of extreme weather happened on that day. We think that this column contains similar information with our binary versions of *preciptotal* and *snowfall*, but it loses simplicity.

2. Train dataset

For the training set that contains the store, item and date information, we keep the store id and item id, and extract day, month, year, is weekend or not, is holiday or not and distance to black Friday (-3 days to +3 days). We believe that these types of information to some extent have independent influence on the sales, so we extract them to different columns. Of course, there are still interactions between them, as a result, we also consider interaction terms.

3. Summary

Data dictionary can be found in the appendix. It is a table that generalizes our feature engineering process. Variables not shown in the table are dropped, and variables left are converted to numerical forms.

Only Use Sold Items: Drop Rows

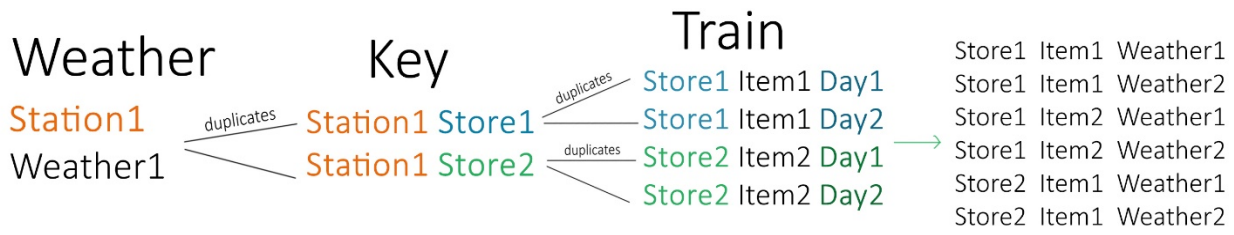
Before merging the dataset together, we also constrain the dataset to lower row spaces. The original train dataset contains more than 4 million of rows and it is quite sparse. For model efficiency, we need to shrink this number. One assumption is made: if an item is

never sold in a store, it doesn't exist in this store (at least during the two years). Hence the prediction for these observations would be 0 whenever the time. This assumption helps us shrink both training and testing data because we don't need to train and predict all the data.

The shrinkage process helps us decrease the row number of train set from 4 million to 0.2 million, and test set from 526 thousand to 26 thousand.

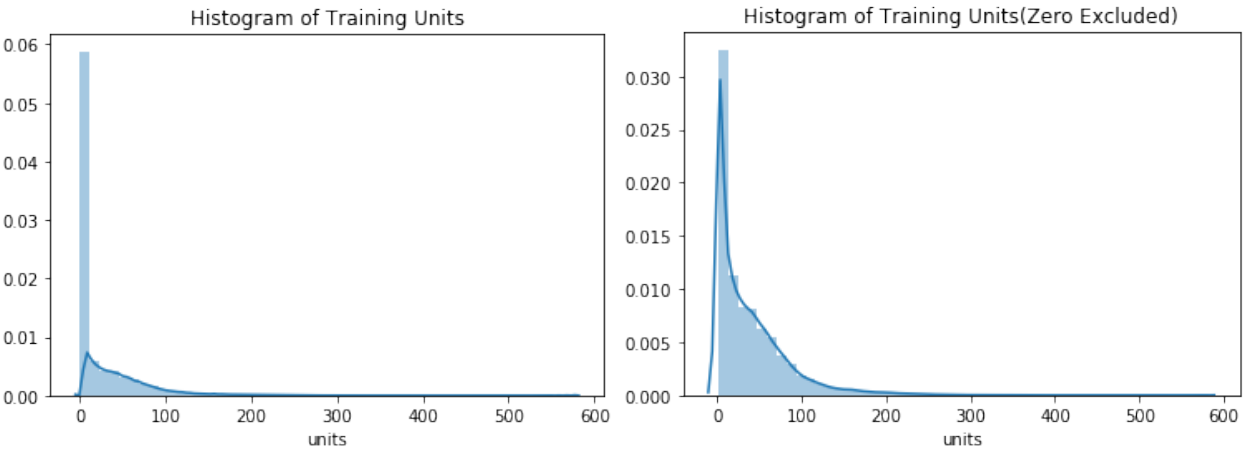
Merge Dataset

We first merge the weather and key dataset by matching the station. One station corresponds to many stores thus rows will be duplicated. Then the new dataset is merged to train set, with store number and date being matched.



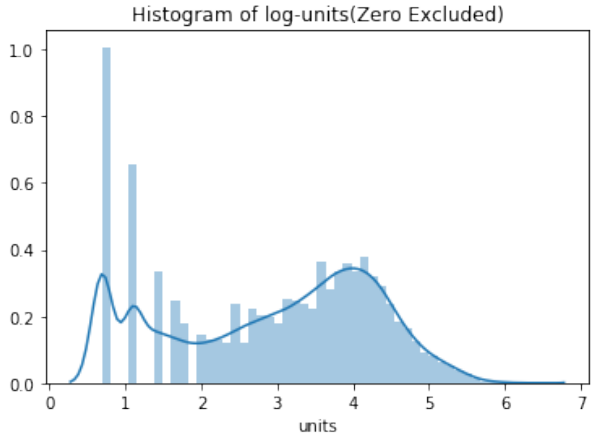
Explanatory Data Analysis

1. Distribution of Units



It is normal for variables like units being right skewed, because lower bound for units is zero and there is no upper bound. We plot the histogram of units of items that have been sold on observation level, and we see that it is right skewed heavily. Consider the fact that there are still many zero units in the training data, we do another histogram

plot that only counts non-zero units. And we can see that the skewness still exists. This is mainly caused by the huge number of observations that have small units.



2. Distribution of log(units)

We also do a log(x+1) transformation to the non-zero units and here is another plot. We can see that the right skewness has been corrected, but still there is an abnormal peak around log(unis+1) = 1. Which means the majority of the non-zero sales is around e-1, which is approximately 1.

3. Linear Model and Model Diagnostics

Fit Linear Regression

We fit a linear regression model (OLS) using all available predictors. For the purpose of this analysis, 236038 observations were used as the training dataset. From the Summary, we can learn that the residual standard error is 19.64 on 23854 degrees of freedom. As for the significance of the regression test, the F-statistics is 2874 and the p-value is < 2.2e-16. We can reject the null hypothesis that the units do not depend on all other predictors. That is, the probability is low that the observations could have come from the null model by chance. R-squared is 0.72, which means 72% variability of the units can be explained by our model. The adjusted-R-squared is 0.7197, which tries to penalize adding unnecessary predictors which only add noise to the model by measuring the degrees of freedom of both RSS and TSS.

Coefficients and Significance

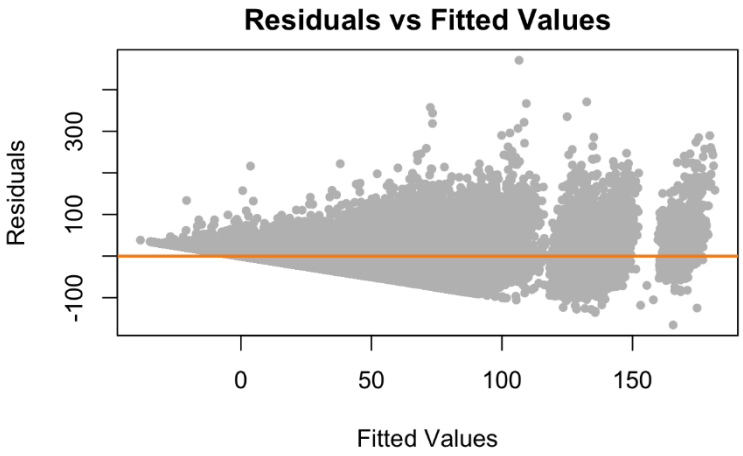
None of the four weather variables except for the temperature average is significant. That is, people go to Walmart for shopping no matter how weather fluctuates. For the date variables, year from 2012 to 2014, month from Jan. to Dec. except for Aug., day from 1st to 31st except 5th and are significant at the level of 99.9%. For instance, compared to the year 2012, the number of products people will buy is 6 less in the year 2014 on average.

As for the holiday information, days that are holiday and weekend, weekend, as well as days around Black Friday, have a significant influence on the units. Customers tend to purchase more on the holiday, especially during Black Friday. For example, products are more likely to sell 11.26 less on average if it is not Black Friday. Furthermore, all the store numbers from 1 to 45 except 29, 32, 37 and 42, as well as all the item numbers from 1 to 111 except 8, 30 and 68 will significantly affect the units of products. The reason why some certain stores and items or the combination of both have no significant inference on the units need more information provided by Walmart.

Model Diagnostics

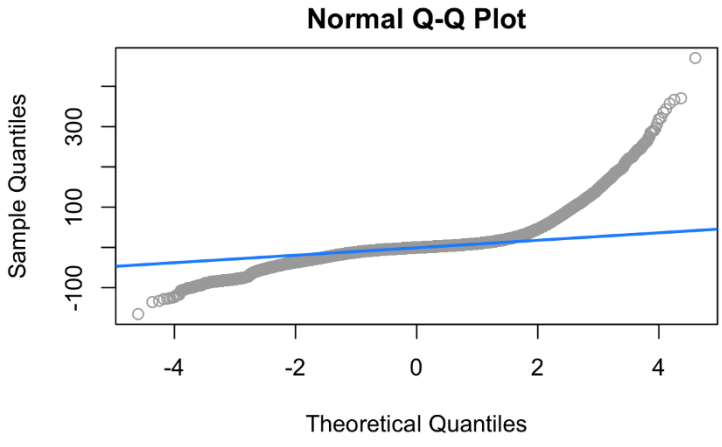
1. Error-constant variance

Fitted versus Residuals Plot was used to check the linearity and constant variance assumptions. For each fitted value, the residuals seem roughly centered at 0. The linearity assumption is valid. For larger fitted values, the spread of the residuals is larger. Moreover, the p-value of the Breusch-Pagan Test is < 2.2e-16, so we can reject the null hypothesis of homoscedasticity at the level of 95%. That is, the errors do not have constant variance.



2. residuals-normal distribution

From the Normal Q-Q Plot, the residuals do not follow a straight line. We can conclude that the residuals are not normally distributed.



3.outlier

Outliers are points that do not fit the current model well. An outlier test enables us to distinguish between truly unusual observations and residuals that are large, but not exceptional. Bonferroni Correction is used to detect outliers. There are 143 observations are outliers if we use Bonferroni Critical Value as the cutoff.

4. Influential point

An influential point is one whose removal from the dataset would cause a large change in the fit. A common measure of influence is Cook's Distance and it is often considered large when larger than 1 and the observation with a large Cook's Distance is considered to be influential. If using 1 as cutoff, there is no influential point.

Fit Linear Regression

First, we fit an ordinary least square regression with no transformation. Because the prediction generates values lower than 0, we set them to 0 manually, and this process will be done whatever model we fit. Here is our score:

Name	Submitted	Wait time	Execution time	Score
ols_submission (2).csv	just now	0 seconds	3 seconds	0.24339

It performs better than all-0 results, but not that good compared to the leaderboard results. Model diagnostics are important for us to find problems lying in this model.

4. Improvements

Log(x+1) transformation

We have already found that our units variable is heavily right-skewed, and the variances are not constant, hence we have enough motivation to try the log-transformed linear regressions. Here is the score of least square regression performed on all variables with transformation on y', where:

$$y' = \log(y + 1)$$

Here's our score:

Name	Submitted	Wait time	Execution time	Score
lm_log_submission_zyq_all.csv	just now	0 seconds	4 seconds	0.13193

The score directly decreases from 0.24 to 0.13, which means the log-transformation do play a big role in predicting. Based on it, we now consider further improvements.

Lasso

Maybe we can shrink the feature spaces. We are using 215 variables, which is a quite large number for linear regression and might possibly cause overfitting. To add shrinkage to the variables, we are going to run a lasso regression with log(x+1) transformation.

LASSO regression is a regularized or shrinkage form of linear regression. The main advantage of it is avoiding overfitting by adding upper limit to the size of coefficients when minimizing squared. Which is:

$$argmin\|y - X\beta\|^2 + \lambda|\beta|$$

The overall idea of regularization is that we are trying to reduce the variability amongst two datasets by trying to increase the bias. In other words, lasso regression works by attempting at sacrificing a little bit bias for reduce in variances and thus improve the generalization capability. Theoretically, the model performance will be a little poor on the training set but it will perform consistently well on both the testing dataset. For least square regression, we are trying to minimize the sum of the squared residuals.

Similarly, LASSO uses a different penalty term, which is the sum of the absolute values of the coefficients. Although it shrinks the coefficients in a different way, their goal is the same --- avoiding overfitting. We do cross validation in order to find the best shrinkage parameter for LASSO. Our search area is between 0.0001 to 0.001. We get our "best model" at alpha = 0.0005, which is a quite small shrinkage (nearly no shrinkage). Sadly, the result doesn't increase compared with our log transformed model.

Linear model with backward variable selection

According the Faraway book, the smallest model that fits the data adequately is best. This suggests that we need a quality criterion that takes into account the size of the model. We could sacrifice a small amount of goodness-of-fit for obtaining a smaller model. In

the full linear model, there were 213 predictors if convert categorical variables into different levels with 1 and 0 as output. Backward variable selection methods based on AIC was applied to help us select significant variables. Its procedures start with all possible predictors in the full model, then considers how deleting a single predictor will affect a chosen metric. Here AIC (Akaike Information Criterion) was used as the metric to measure the model quality.

$$AIC = -2logL(\hat{\beta}, \hat{\sigma}^2) + 2p = n + nlog(2\pi) + log\frac{RSS}{n} - \frac{n}{2} + 2p$$

The two main components of AIC are the likelihood (which measures goodness-of-fit, a function of RSS which will be constant across all models applied to the same data) and the penalty (which is a function of the size of the model). It will then repeatedly attempt to delete predictors until it reaches the best metric (lowest AIC), or reaches the intercept-only model. After the procedure, the full linear model ends up with 208 predictors while the log-transformed linear model ends up with 211 predictors. The full linear model after backward selection can predict slightly better than before. However, the log-transformed linear model after backward selection got the same score as the log-transformed linear model. The reason is that only 2 predictors were deleted which does not make a difference. If we change the penalty and do not use AIC as the metric, we may end up with models with less predictors. Nevertheless, if we look at the summary table of the linear model, almost all the predictors are significant at the level of 99.9%. Therefore, this also explains that there is no point consider dropping existing variables.

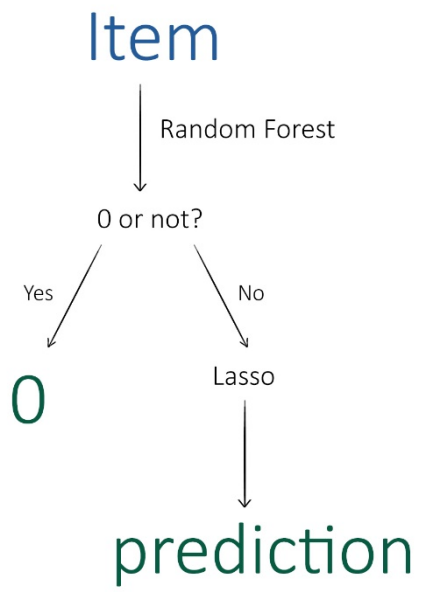
Name	Submitted	Wait time	Execution time	Score
lm_step_log_submission_zyq_all.csv	5 hours ago	167 seconds	3 seconds	0.13194

Adding interaction terms

As mentioned before, the predictors we used mainly include three types of information, which are date, weather, holiday, store and item number. We create interaction terms between date and store & item number, holiday and store & item number, weather and store & item number to make a more complex model and see if the complex model can predict with a smaller RMSLE. However, the Kaggle score did not improve.

Name	Submitted	Wait time	Execution time	Score
lm_log_int_submission_zyq_all.csv	a few seconds ago	0 seconds	4 seconds	0.13192

Trick: Classification Plus Lasso



Now we add some non-linear tricks to our linear regression. We observed from the histogram of units that even after log transformation there is still a huge number of 0 entries. We guess that happens to the testing set, too. So, we are going to fill more 0 to our prediction. This time, instead of directly running regression on all testing set, we now use a random forest algorithm (introduction will be given in following parts) to classify observations into "zero" and "nonzero" classes, then we linear regression to predict units for "nonzero" class, and simply fill the "zero" classes with number 0.

The random forest is trained with cross validation for searching best n_estimatoes and the cv result suggested n_estimators=20, which gives cv accuracy equal to 80%. We then run a log transformed lasso regression on the non-zero class. The result will be combined with zero class together.

Name	Submitted	Wait time	Execution time	Score
2m_submission.csv	just now	0 seconds	4 seconds	0.11920

Great! An improvement in the score validates our guess. There actually do exist more 0 entries than normal distribution in testing set.

5. Extra Models

Random Forests

Random Forest is an ensemble technique, which means that it uses multiple learning algorithms to obtain a better predictive performance. As it took from the name, it will use decision trees as base learners. In the data, we create some bootstrap samples. Then for each individual bootstrap sample, we create a decision tree that results in a prediction.

These predictions are then aggregated choosing majority voting and result in a final probability. This might look just like a bagged tree, however, there is a major difference between bagged trees and random forests. In bagged trees, all the predictors are used at splits in the decision trees, while in random forests, only a random sample of the predictors are considered as split candidates at each split of the tree.

After performing one-hot encoding to our dataset, it becomes fairly wide (213 columns total). Luckily random forests are good at dealing with high dimensional data because each base learner is working with a bootstrap resample of the data. Besides, random forests effectively reduce the variance for its large number of individual trees. The covariance between each individual tree are also reasonably small because a random sample of predictors are used at each split for each tree. The base learner of random forests is decision tree, so the data does not need to be rescaled or transformed. We barely need any preprocessing work.

One rule of thumb for choosing the number of parameters is the square root of the number of predictors. Using scikit-learn package in Python, we set the number of trees as 100, the function to measure the quality of a split as mean squared error, the minimum number of samples required to split an internal node as 2, and the minimum number of samples required to be at a leaf node as 1. The training time is about 5 minutes, and the Kaggle score is 0.11175.

Name	Submitted	Wait time	Execution time	Score
rf_submission.csv	14 minutes ago	468 seconds	3 seconds	0.11175

Light GBM

Like random forests, a tree boosting method generates large numbers of trees and "aggregate" their results. While in random forests trees are learned parallel, in boosting methods a tree is built on the bases of results of previous trees. You can imagine that they are "learning on current residuals". Suppose we are predicting the age of Yuqi, a clever girl who is now 22 years old, and we fit a single decision tree which gives a prediction equal to 20. The residual now is 2, and we will fit another tree model based on this

information. When an observation has larger residuals (or loss) currently, we put more weights on it when fitting the next tree. The final prediction is a simple weighted sum of results at each iteration.

Gradient boosting models use gradients of loss function at each step, hence it takes less iterations to converge. Light Gradient Boosting (Light GBM) further accelerates the convergence of GBM, using two algorithms (GOSS and EFB) that do subsampling and bagging both within and outside a single tree. Instead of simple random subsampling, light gbm chooses observations that have largest gradients at each iteration and does a so-called "best-first" splitting at each step of a tree. To explain it more detailly the "best-first" splitting, normal trees grow in a fixed order at each level (for example, from left to right), and when all splitting at this level is finished, they move to the lower level. However, lightGBM grows leaf-wise, it first chooses a best splitting point and grows on this point. When a leaf reaches the maximum depth of a tree, it goes back and grows other leaves. Though at the end of a full tree the there's no difference between the lightGBM's tree and other trees, when we cut the tree in half by setting maximum tree depth, lightGBM grows quicker with smaller loss. Hence the total convergence rate of both tree and boosting are increased. By doing this it not only reduces computational cost, but also gains higher accuracy.

We fit a lightGBM model on all features we extracted in part 2 in Python using lightgbm package. Number of leaves is kept as default = 31, and we tune n_estimator = 10, 20 and 30. We reaches a score of 0.116 finally at n_estimator = 10. Due to lack of time, we didn't tune the parameters carefully, and this is certainly not the best result lightGBM could reach. But anyway, it is still not bad.

Name	Submitted	Wait time	Execution time	Score
gcvgbm_submission.csv	just now	0 seconds	4 seconds	0.11616

Neural Networks --- Multi-Layer Perceptron

A perceptron, which is also called neuron or node, accepts multiple input numbers and provides an output number. Another component of this perceptron is called synaptic

weight, and each of the weights is associated with each of the incoming input values. These weights are meant to model the strength of the connections from one neuron to another. What happens inside the neuron is that we will sum up all the incoming values which can be calculated as the products of the inputs and their corresponding weights. This sum then gets processed through something called an activation function before we reach the output value. To sum it up, if we treat both input and weight as vectors, then the output is just the dot product of the input vector and the weight vector plus the bias going through an additional activation function. Figure 4 in the appendix shows an example of a perceptron.

A multi-layer perceptron is a fairly standard neural network model that takes the simple perceptron we discussed above and combines many of them together to make a larger and more powerful network. The image below (see figure 5) shows an example neural network, which is a fairly small multi-layer perceptron with an input layer, an output layer, and one hidden layer. There can be multiple hidden layers in a multi-layer perceptron, but this portion of a neural network is considered hidden for the sake of computation. This is because neural networks are normally thought of as black boxes meaning that we provide some vector of inputs and get some vector of outputs, and do not need to fully understand what is going on in the potentially many hidden layers in between. As a matter of fact, the whole multi-layer perceptron is not that hard to understand. Each neuron in the hidden layer has a value that can be obtained by the sum of the product of each input values and their corresponding weights going through an activation function of our choice. Similarly, we can calculate each output value from the sum of the product of each hidden layer values and their corresponding weights going through an activation function. We normally use the sigmoid function as the activation function for the output layer in classification problems so that all outputs add up to 1.

Using scikit-learn package in Python, we set 2 hidden layers with size 128 and 64 respectively. Multi-layer perceptron model can be used in very complex problems. In this problem we have 213 features total after one-hot encoding, which is not too much

compared to other neural network problems. Thus, 2 hidden layers is complicated enough for solving this problem. The computation time is fairly fast with such architecture and no extra data preprocessing work is required. We choose to use the rectified linear unit (ReLU) as the activation function for the hidden layer. ReLU function outputs zero if the input is negative and outputs identity if the input is positive. ReLU function can effectively prevent the gradient from being vanished, because it has a derivative of 1 for the positive gradients. We also used stochastic gradient descent (SGD) as the optimizer. Instead of using the whole training data to update the weights once, SGD only takes a batch of images, which will effectively use the computing power without sacrifice much accuracy to the result. The specific number of one such batch is called the batch size, which is 200 in our MLP model. We also set the learning rate as 0.001 for weight updates and the maximum number of iterations as 200. The training time is about 15 minutes, and the Kaggle score is 0.12128.

Name	Submitted	Wait time	Execution time	Score
mlp2_submission.csv	12 hours ago	0 seconds	4 seconds	0.12128

6. Conclusion

For the linear part, two models should be considered, the log-transformed linear regression and the lasso regression combined with random forest classification, with score equal to 0.1192. The previous runs pretty fast with a reasonable score, while the latter one gets lowest score among all linear models, but takes considerably long time to run.

For the extra model part, we pick the random forest model, which gives us lowest score we have, 0.1117. Light GBM runs much faster but earns a little bit less score, but there is still space for improvements if we can tune the parameters more carefully on better machines.

We still cannot get a score lower than 0.1. And I guess the problem lies on feature engineering. We are certainly using a column space larger than the best, but we are unable

to find redundant variables. Maybe we should consider delete variables manually, as many competitors said on Kaggle and what we observed, weather variables are useless.

References

[1] Deshpande, Mohit. “Perceptrons: The First Neural Networks.” Python Machine Learning, 25 Sept. 2019, pythonmachinelearning.pro/perceptrons-the-first-neural-networks/.

[2] Kho, Julia. “Why Random Forest Is My Favorite Machine Learning Model.” Medium, Towards Data Science, 12 Mar. 2019, towardsdatascience.com/why-random-forest-is-my-favorite-machine-learning-model-b97651fa3706.

[3] Mohanty, Awhan. “Multi Layer Perceptron (MLP) Models on Real World Banking Data.” Medium, Becoming Human: Artificial Intelligence Magazine, 15 May 2019, becominghuman.ai/multi-layer-perceptron-mlp-models-on-real-world-banking-data-f6dd3d7e998f.

[4] McCoy, Max (May 28, 2006). "Wal-Mart's data center remains mystery". The joplin globe. Community Newspaper Holdings. Retrieved May 31, 2019.

[5] Hayes, Thomas C. (February 28, 1990). "Company News; Wal-Mart Net Jumps By 31.8%". The New York Times. Archived from the original on July 23, 2015.

[6] Cheikhrouhou, N., Marmier, F., Ayadi, O., & Wieser, P., (2011).A collaborative demand forecasting process with event-based fuzzy judgments.Comput.Ind.Eng.61 (2), 409–421.

[7] Li, B., Li, J., Li, & W., Shirodkar, S.A., 2012.Demand forecasting for production planning decision-making based on the new optimized fuzzy short time-series clustering. Prod.Plan.Control23 (9), 663–673.

Appendix

Data dictionary

	variable	explanation	category
	units	the quantity sold of an item on a given day	numerical
Date	year	years from 2012 to 2014	2012, 2013, 2014
	month	months from Jan. to Dec.	Jan., Feb., …, Dec.
	day	Days from 1st to 31st	1st, 2nd, …, 31st
	is_weekend	if a certain day is weekend or weekday	True or False (1 or 0)
	is_holiday	if a certain day is holiday	True or False (1 or 0)
	is_holiday_ and_weekend	if a certain day is both holiday and weekend	True or False (1 or 0)
	is_holiday_ and_weekday	if a certain day is both holiday and weekday	True or False (1 or 0)
	black_Friday _info	if a certain day is around Black Friday and how many days	The number of days from a certain date to black Friday, only consider +- 3 days
Number	store number	an ID representing one of the 45 stores	1, 2, …, 45
	item number	an ID representing one of the 111 products	1, 2, …, 111
Weather	tavg	the average temperature for that day	numerical
	is_temp _surge	if the temperature of a certain day increases 8F than the former day	True or False (1 or 0)
	is_temp _drop	if the temperature of a certain day decreases 8F than the former day	True or False (1 or 0)
	is_heavy _precip	if a certain day have precipitation larger than 2 inches	True or False (1 or 0)

Figures and Charts

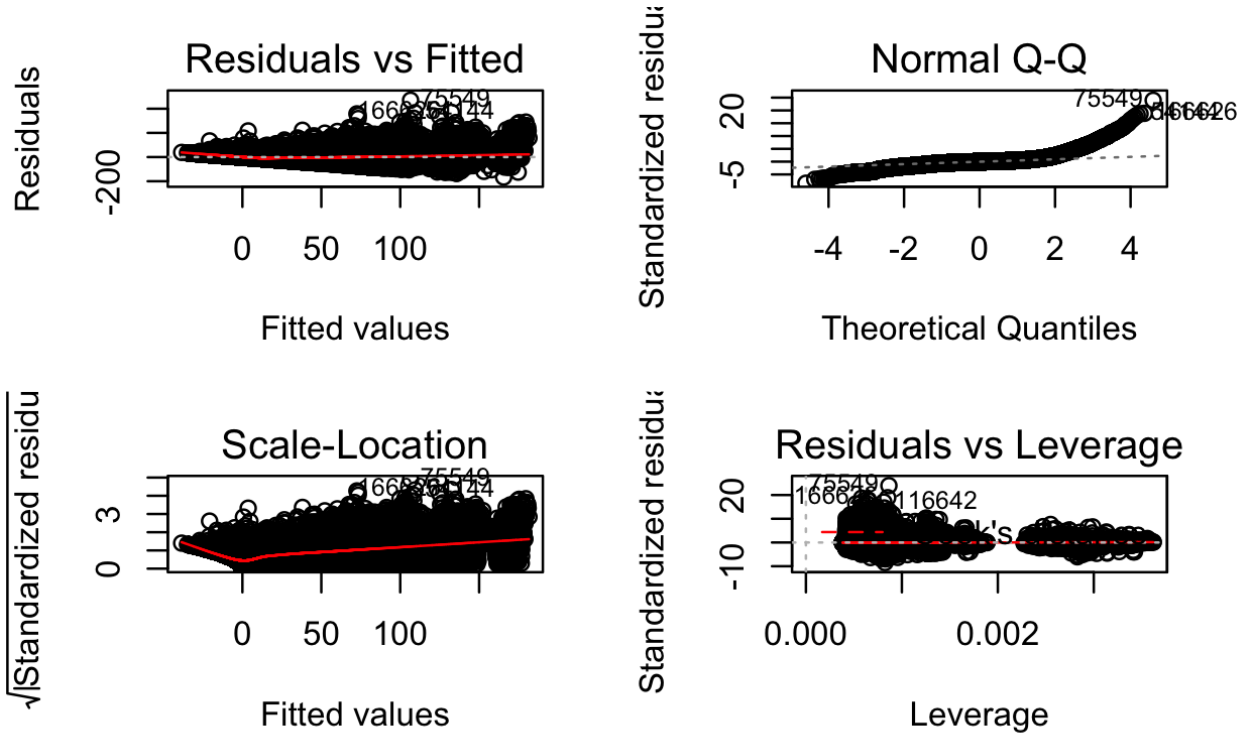


Figure 1: Diagnostics Plots for Full Linear Model

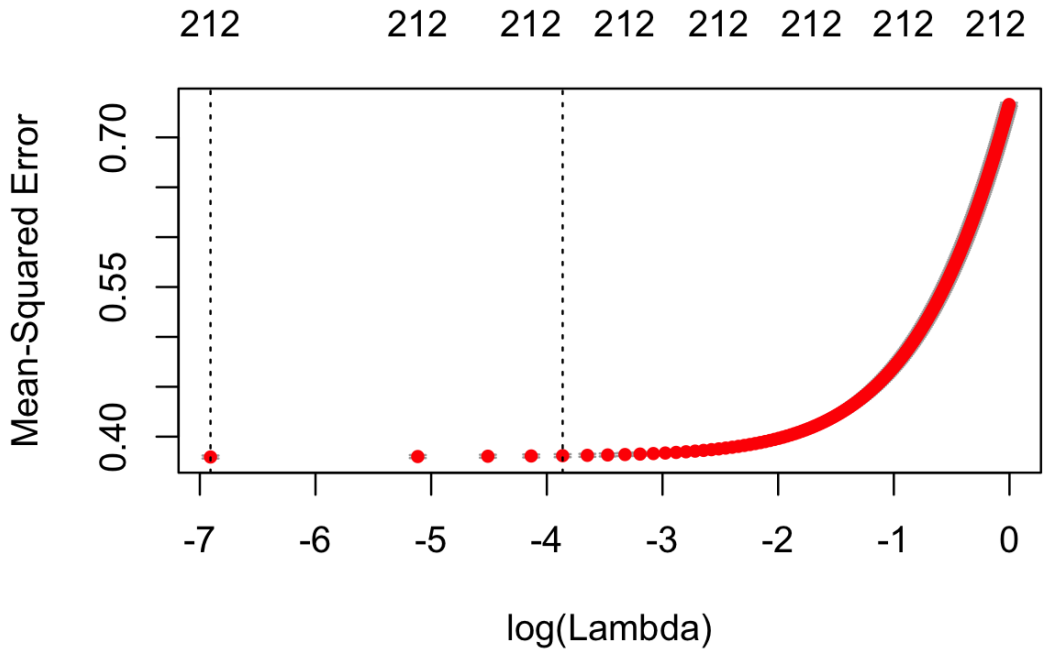


Figure 2: Lambda Plot for Ridge

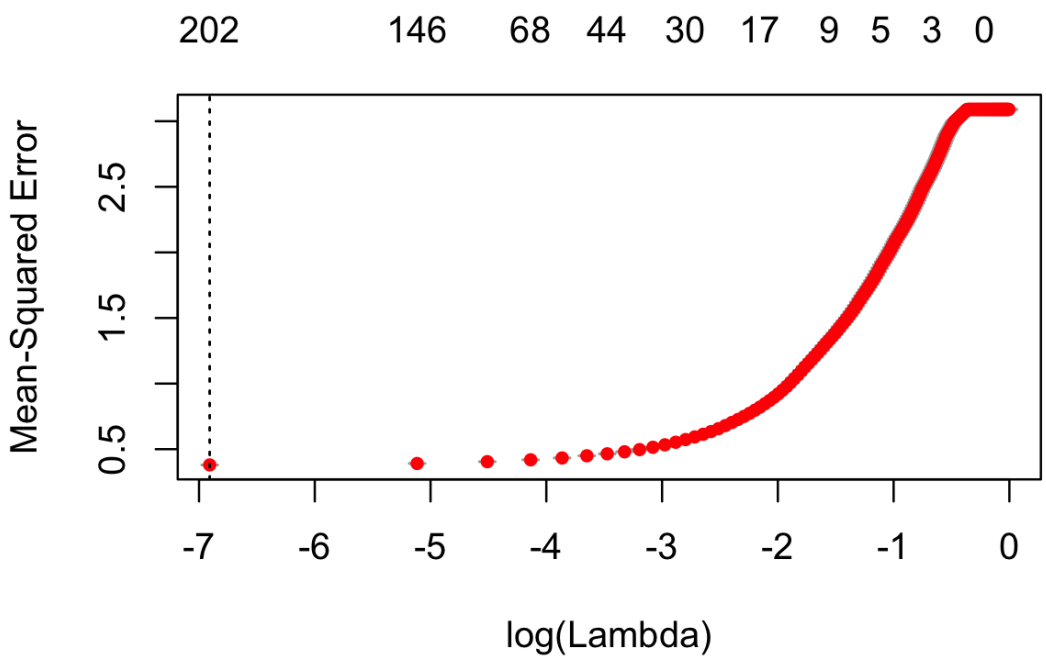


Figure 3: Lambda Plot for LASSO

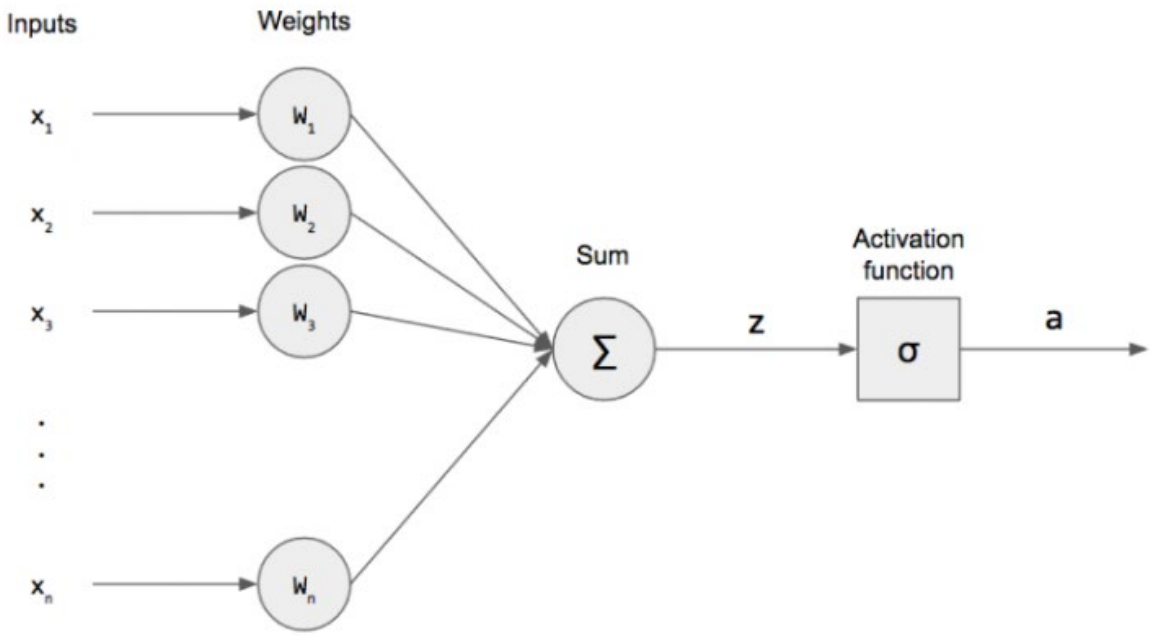


Figure 4: Graph of a single perceptron

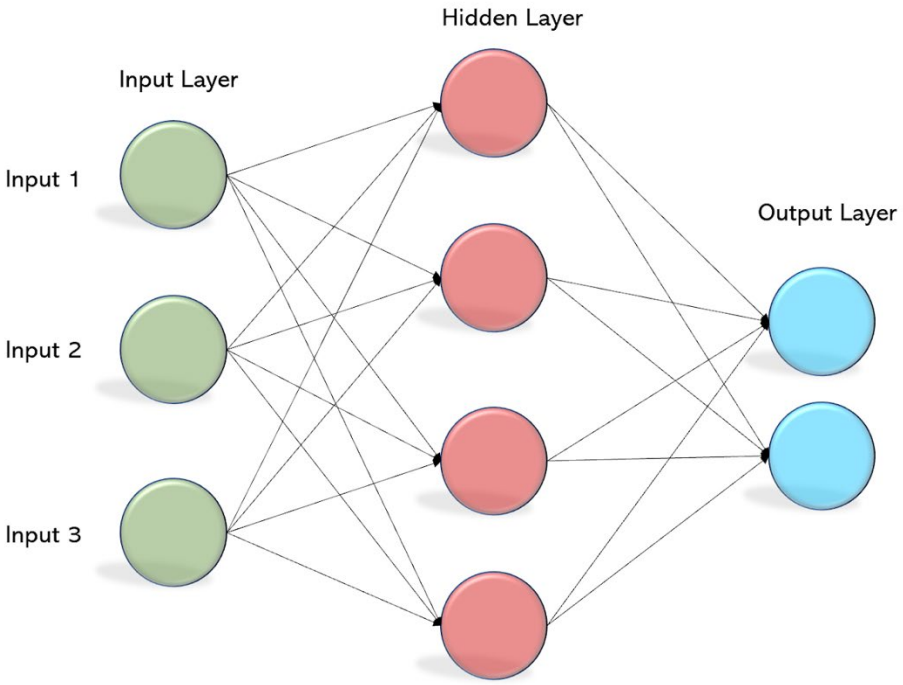


Figure 5: Graph of a multi-layer perceptron

Code

```
import numpy as np
import pandas as pd
import seaborn as sns
import statsmodels
from matplotlib import pyplot as plt
%matplotlib inline
# read weather data
# ----------------------------------
data = pd.read_csv('../input/weather/weather.csv')
data['date']=pd.to_datetime(data['date'])
# check non-numerical values in weather set
# ----------------------------------
def if_M(data):
    result = []
    for i in range(0,data.shape[1]):
        result.append('M' in data.iloc[:,i].unique())
    result = pd.DataFrame(result, index = data.columns, columns=['if M'])
    return result
def if_line(data):
    result = []
    for i in range(0,data.shape[1]):
        result.append('-' in data.iloc[:,i].unique())
    result = pd.DataFrame(result, index = data.columns, columns=['if -'])
    return result
def if_whitespace(data):
    result = []
    for i in range(0,data.shape[1]):
        result.append(' ' in data.iloc[:,i].unique())
```

```
    result = pd.DataFrame(result, index = data.columns, columns=['if whitespace'])
    return result
def if_T(data):
    result = []
    for i in range(0,data.shape[1]):
        result.append('  T' in data.iloc[:,i].unique())
    result = pd.DataFrame(result, index = data.columns, columns=['if T'])
    return result
def check_data(data):
    r1 = if_M(data)
    r2 = if_line(data)
    r3 = if_whitespace(data)
    r4 = if_T(data)
    return pd.concat([r1,r2,r3,r4], axis = 1)

check_data(data)
# clean non-numerical values
# ------------------------------------
M_list = pd.DataFrame(data.columns)[if_M(data).values][0].tolist()
def remove_M(column):
    column[column=='M'] = np.nan
    return column
for i in M_list:
    data[i] = remove_M(data[i])

line_list = pd.DataFrame(data.columns)[if_line(data).values][0].tolist()
def remove_line(column):
    column[column=='-'] = np.nan
    return column
for i in line_list:
    data[i] = remove_line(data[i])
```

```
T_list = pd.DataFrame(data.columns)[if_T(data).values][0].tolist()
def remove_T(column):
    column[column=='  T'] = np.nan
    return column
for i in T_list:
    data[i] = remove_T(data[i])
# correlation table
# -------------------------------------
data_sub = data.loc[:, ['tmax', 'tmin', 'tavg', 'depart', 'dewpoint', 'wetbulb', 'heat', 'cool', 'sunrise',
'sunset',
                        'snowfall', 'preciptotal', 'stnpressure', 'sealevel', 'resultspeed', 'resultdir', 'avgspeed']]
data_sub = data_sub.apply(lambda x: x.astype(float), axis=0)
data_sub = data_sub.fillna(method='ffill').fillna(method='bfill')


sns.set()
corr = data_sub.corr()
ax = sns.heatmap(
     corr,
     vmin=-1, vmax=1, center=0,
     cmap=sns.diverging_palette(20, 220, n=200),
     square=True
)


ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=45,
    horizontalalignment='right'
);
# variable selection, feature engineering
# -------------------------------------
data = data.loc[:,['station_nbr','date','tavg','depart','preciptotal']]
data["tavg"] = data["tavg"].astype(float)
data["depart"] = data["depart"].astype(float)
```

```
data["preciptotal"] = data["preciptotal"].astype(float)
data['is_heavy_precip'] = data['preciptotal'].apply(lambda x: int(x > 0.2))
data['is_temp_surge'] = data['depart'].apply(lambda x: int(x > 8))
data['is_temp_drop'] = data['depart'].apply(lambda x: int(x < -8))
data = data.drop(['preciptotal', 'depart'], axis = 1)
data = data.fillna(method='ffill').fillna(method='bfill')
data['is_weekend'] = data['date'].dt.dayofweek.apply(lambda x: int(x in [5,6]))
data['year'] = data['date'].dt.year
data['month'] = data['date'].dt.month
data['day'] = data['date'].dt.day
# create a processing indtroduction dataframe
# ------------------------------------
prepro = [['tavg', 'convert to float'],
        ['is_heavy_precip', 'preciptotal > 0.2: 1'],
        ['', 'preciptotal <= 0.2: 0'],
        ['is_temp_surge', 'depart > 8: 1'],
        ['', 'depart <= 8: 0'],
        ['is_temp_drop', 'depart < -8: 1'],
        ['', 'depart >= -8: 0'],
        ['is_weekend', 'extracted from date'],
        ['is_holiday', 'extracted from date'],
        ['is_holiday_and_weekend', 'extracted from date'],
        ['year', 'extracted from date'],
        ['month', 'extracted from date'],
        ['day', 'extracted from date'],
        ['Black Friday Info', 'generate 7 extra columns from date']]
prepro = pd.DataFrame(prepro, columns = ['variable', 'process'])
prepro
# Add is_holiday
# ------------------------------------
from datetime import datetime
date_format = "%Y-%m-%d"
```

```
holiday_list = ["2012-01-01", "2012-01-16", "2012-02-14", "2012-02-20", "2012-04-08",
                "2012-05-13", "2012-05-28", "2012-06-17", "2012-07-04", "2012-09-03",
                "2012-10-08", "2012-10-31", "2012-11-11", "2012-11-20", "2012-11-21",
                "2012-11-22", "2012-11-23", "2012-11-24", "2012-11-25", "2012-11-26",
                "2012-12-24", "2012-12-25", "2012-12-31", "2013-01-01", "2013-01-21",
                "2013-02-14", "2013-02-18", "2013-05-31", "2013-05-12", "2013-05-27",
                "2013-06-16", "2013-07-04", "2013-09-02", "2013-10-14", "2013-10-31",
                "2013-11-11", "2013-11-26", "2013-11-27", "2013-11-28", "2013-11-29",
                "2013-11-30", "2013-12-01", "2013-12-02", "2013-12-24", "2013-12-25",
                "2013-12-31", "2014-01-01", "2014-01-20", "2014-02-14", "2014-02-17",
                "2014-04-20", "2014-05-11", "2014-05-26", "2014-06-15", "2014-07-04",
                "2014-09-01", "2014-10-13", "2014-10-31"]

holidays = [datetime.strptime(holiday, date_format) for holiday in holiday_list]

data['is_holiday'] = 0

for i in range(0, data.shape[0]):
    if data["date"][i] in holidays:
        data.is_holiday.iloc[i] = 1

data['is_holiday_and_weekend'] = data['is_holiday'] & data['is_weekend']
# add black-friday related variables
# -------------------------------------
data['Black_Friday_info'] = '0'

data.loc[data['date'] == datetime.strptime("2012-11-20", date_format), ['Black_Friday_info']] = '-3days'
data.loc[data['date'] == datetime.strptime("2012-11-21", date_format), ['Black_Friday_info']] = '-2days'
data.loc[data['date'] == datetime.strptime("2012-11-22", date_format), ['Black_Friday_info']] = '-1day'
data.loc[data['date'] == datetime.strptime("2012-11-23", date_format), ['Black_Friday_info']] = 'BF'
```

```
data.loc[data['date'] == datetime.strptime("2012-11-24", date_format), ['Black_Friday_info']] = '+1day'

data.loc[data['date'] == datetime.strptime("2012-11-25", date_format), ['Black_Friday_info']] = '+2days'

data.loc[data['date'] == datetime.strptime("2012-11-26", date_format), ['Black_Friday_info']] = '+3days'


data.loc[data['date'] == datetime.strptime("2013-11-26", date_format), ['Black_Friday_info']] = '-3days'

data.loc[data['date'] == datetime.strptime("2013-11-27", date_format), ['Black_Friday_info']] = '-2days'

data.loc[data['date'] == datetime.strptime("2013-11-28", date_format), ['Black_Friday_info']] = '-1day'

data.loc[data['date'] == datetime.strptime("2013-11-29", date_format), ['Black_Friday_info']] = 'BF'

data.loc[data['date'] == datetime.strptime("2013-11-30", date_format), ['Black_Friday_info']] = '+1day'

data.loc[data['date'] == datetime.strptime("2013-12-01", date_format), ['Black_Friday_info']] = '+2days'

data.loc[data['date'] == datetime.strptime("2013-12-02", date_format), ['Black_Friday_info']] = '+3days'

# read train and key

# ------------------------------------

train = pd.read_csv('../input/trainkey/train.csv')

train.date = pd.to_datetime(train.date)

key = pd.read_csv('../input/trainkey/key.csv')


# merge data

data = pd.merge(data, key, how = 'left', on = 'station_nbr')

training = pd.merge(data,train,how = 'right', on = ['date', 'store_nbr'])

# find items that are never sold in a store

#########################################################################################

# warning!!!!!!!!!!!!!!!!!!don't run!!!!!!!!!!!!computer will boom!!!!!!!!!!!!!!!!! #

#########################################################################################

for store_nbr in range(1, 46):

    print(store_nbr, "/ 45")
```

```
    for item_nbr in range(1, 112):
        # extract one specific comb of store nbr and item nbr
        temp_data = training[(training["store_nbr"] == store_nbr) & (training["item_nbr"] == item_nbr)]

        # check if all units in the temp data are zeros
        if np.mean(temp_data["units"] == 0) == 1.0:

            # delete all indices in the temp data from the whole data
            training = training.drop(axis = 0, index = temp_data.index)

# save the results for future usage
training.to_csv("training_data.csv", index = False)

deleted_list = []
saved_list = []
for store_nbr in range(1, 46):
    for item_nbr in range(1, 112):
        # extract one specific comb of store nbr and item nbr
        temp_data = training[(training["store_nbr"] == store_nbr) & (training["item_nbr"] == item_nbr)]

        if temp_data.shape[0] == 0:
            deleted_list.append((store_nbr, item_nbr))
        else:
            saved_list.append((store_nbr, item_nbr))

import pickle
# save deleted_list as a file
with open("deleted_list.txt", "wb") as dl:
    pickle.dump(deleted_list, dl)

# save saved_list as a file
```

```
with open("saved_list.txt", "wb") as sl:
    pickle.dump(saved_list, sl)

# Run the following code to load lists
with open("deleted_list.txt", "rb") as dl:
    deleted_list = pickle.load(dl)

with open("saved_list.txt", "rb") as sl:
    saved_list = pickle.load(sl)

training_reduced = pd.read_csv("training_data_1216.csv")
training_reduced = training_reduced.fillna("0")

training_reduced.to_csv("training_reduced.csv", index = False)
data.to_csv("weather_data.csv", index = False)

# remove zero units in testing data
training = pd.read_csv('training_data.csv')
testing = pd.read_csv('test.csv')

for i in range(0, testing.shape[0]):
    if (i % 527) == 0:
        print(i/527, "/1000 completed")
    if (testing.store_nbr[i], testing.item_nbr[i]) in deleted_list:
        testing = testing.drop(axis = 0, index = i)

testing.to_csv("testing_reduced.csv", index = False)
testing_idx = testing.index.tolist()

import pickle
# save testing_idx as a file
with open("testing_idx.txt", "wb") as ti:
```

```
    pickle.dump(testing_idx, ti)

testing = pd.read_csv("testing_reduced.csv")
testing.date = pd.to_datetime(testing.date)
testing = pd.merge(data,testing,how = 'right', on = ['date', 'store_nbr'])
testing.to_csv("testing_reduced.csv", index = False)
'''
# We have alternative code for above process
# but needs a little bit revise
train = pd.read_csv('../input/trainkey/train.csv')
train.date = pd.to_datetime(train.date)
matrix = train.pivot_table(values='units',index = 'store_nbr', columns='item_nbr',aggfunc='sum')
nonzero_index = matrix.to_numpy().nonzero()
train_ind = train.apply(lambda x: str(x.store_nbr) + '-' + str(x.item_nbr),axis = 1)
train = train[train_ind.isin(nonzeros)]

# then merge the data
'''
# Now we can read reduced files!
# ------------------------------------
training = pd.read_csv('training_reduced.csv')
testing = pd.read_csv('testing_reduced.csv')
# data processing
# ------------------------------------
# drop abnormal data
training = training.drop(training[training['units'] > 1000].index, axis=0)

# get y
ytrain = training['units']
training = training.drop('units', axis = 1)

# split data
```

```
len_train = len(training)
train_test = pd.concat([training, testing])

'''
train_test = pd.get_dummies(train_test, columns=['year','month','store_nbr','item_nbr','Black_Friday_info'],
                            drop_first=True)
'''

train_test = train_test.drop(['station_nbr', 'is_holiday_and_weekday', 'date',
                              'is_heavy_precip', 'is_temp_surge', 'is_temp_drop'], axis=1)
train_test = pd.get_dummies(train_test, columns=['year','month','day','store_nbr','item_nbr','Black_Friday_info'],
                            drop_first=True)

#dates = train_test['date'].apply(lambda x: x.replace("-", "")).astype('int')
#train_test['date'] = dates - dates.iloc[0]

xtrain, xtest = train_test.iloc[:len_train, :], train_test.iloc[len_train:,:]

# convert to sparse matrix
from scipy.sparse import csr_matrix
xtrain = csr_matrix(xtrain, shape = xtrain.shape)
xtest = csr_matrix(xtest, shape = xtest.shape)
# descriptive statistics
# ------------------------------------
# distribution of y
yhist = sns.distplot(ytrain)
plt.title("Histogram of Training Units")
plt.show()

yhist2 = sns.distplot(ytrain[ytrain > 0])
plt.title("Histogram of Training Units(Zero Excluded)")
```

```
plt.show()

# after log transformation
sns.distplot(np.log(ytrain[ytrain > 0] + 1))
plt.title("Histogram of log-units(Zero Excluded)")
# functions for saving results
# ------------------------------------
import pickle
test = pd.read_csv('../input/testing/test.csv')
len_test = test.shape[0]
with open("../input/testingidx/testing_idx.txt", "rb") as ti:
    testing_idx = pickle.load(ti)
ss = pd.read_csv("../input/samplesubmission/sampleSubmission(1).csv")

# save ols results
def get_result(prediction, output_name):
    result = np.zeros(len_test)
    for i in range(0, len(prediction)):
        result[testing_idx[i]] = prediction[i]
    result = pd.DataFrame(result)
    result[result<1]=0
    ss["units"] = result
    ss.to_csv(output_name+"_submission.csv", index = False)
# fit linear regression
# ------------------------------------
from sklearn.linear_model import LinearRegression
ols = LinearRegression().fit(xtrain, ytrain)

# R^2
ols.score(xtrain, ytrain)

# predict
```

```
pred_ols = ols.predict(xtest)

# save linear results
get_result(pred_ols, 'ols')
# fit log-transformed linear regression
# ------------------------------------
log_ols = LinearRegression().fit(xtrain, np.log(ytrain + 1))

# predict
pred_log = log_ols.predict(xtest)
pred_log = np.exp(pred_log) - 1
pred_log[pred_log < 0] = 0

# save results
get_result(pred_log, 'logols4')
# lasso
# ------------------------------------
# cross-validationn

from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Lasso
gcv_ls = GridSearchCV(Lasso(), {'alpha':np.linspace(.0001,.001,30).tolist()},scoring =
'neg_mean_squared_error')
gcv_ls.fit(xtrain, np.log(ytrain + 1))
gcv_ls.best_params_

# alpha: 0.0005
# predict lasso cv
pred_lasso = gcv_ls.predict(xtest)
pred_lasso = np.exp(pred_lasso) - 1
pred_lasso[pred_lasso < 0] = 0

get_result(pred_lasso, 'lasso3')
```

```
'''
# logistic + lm
from sklearn.linear_model.logistic import LogisticRegressionCV
yclass = ytrain.apply(lambda x: int(x>0))
logcv = LogisticRegressionCV(scoring='accuracy',cv=5)
logcv.fit(xtrain, yclass)
preds = logcv.predict(xtest)
'''
# random forest classification + lasso regression
# ------------------------------------
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier

yclass = ytrain.apply(lambda x: int(x>0))
param_grid = {'n_estimators':[5,10,20]}
gcv = GridSearchCV(RandomForestClassifier(), param_grid, scoring='accuracy',cv = 5)
gcv.fit(xtrain, yclass)
preds = gcv.predict(xtest)

# get positive data
posind = np.where(preds==1)
res1 = np.zeros(xtest.shape[0])
xtrain_pos = csr_matrix(xtrain.toarray()[ytrain>0])
xtest_pos = csr_matrix(xtest[preds==1])
ytrain_pos = ytrain[ytrain>0]
ytrain_pos_log = np.log(ytrain_pos+1)

# lasso
lscv = LassoCV(cv=5, alphas = [.0001,.0005,.001,.005,.01,.05,.1,.5,1])
lscv.fit(xtrain_pos, ytrain_pos_log)
```

```
# save results
preds2 = np.exp(lscv.predict(xtest_pos))-1
res1[posind] = preds2
get_result(res1, '2m')
# lgbm
from lightgbm import LGBMRegressor
gbm = LGBMRegressor()
gbm.fit(xtrain_pos, ytrain_pos_log)
preds3 = np.exp(gbm.predict(xtest_pos))-1
res2 = np.zeros(xtest.shape[0])
res2[posind] = preds3
get_result(res2, '2gbm')
# random forest
# 0.11175
rf_mod = RandomForestRegressor(n_estimators = 100, n_jobs = -1, random_state = 42).fit(xtrain, ytrain)
preds = rf_mod.predict(xtest)
# neural networks
# 0.12128
mlp = MLPRegressor(hidden_layer_sizes = (128, 64), max_iter = 500).fit(xtrain, ytrain)
preds = mlp.predict(xtest)


# 0.16855
# gb = GradientBoostingRegressor(n_estimators = 500).fit(xtrain, ytrain)
# preds = gb.predict(xtest)
```

Rcode

```
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

load packages
```{r}
library("dplyr")
library("glmnet")
```
```

load training dataset and testing dataset
```{r, load data}
training_reduced = read.csv("training_reduced.csv")
testing_reduced = read.csv("testing_reduced.csv")
```

data preprocessing
```{r data preprocessing - training}
# data preprocessing
training_data = training_reduced %>%
  select( - station_nbr, - date) %>%
  mutate_each(as.factor) %>%
  mutate(units = as.numeric(as.character(units)),
         tavg = as.numeric(as.character(tavg))) %>%
  filter(units < 1000)
```

```{r data preprocessing - testing}
testing_data = testing_reduced %>%
  select( - station_nbr, - date) %>%
  mutate_each(as.factor) %>%
  mutate(tavg = as.numeric(as.character(tavg)))
```

```{r}
testing_data_m = testing_reduced %>%
  select( - station_nbr, - date) %>%
  mutate_each(as.factor) %>%
  mutate(tavg = as.numeric(as.character(tavg))) %>%
  mutate(units = rep(0,nrow(testing_reduced)))
```

```{r}
index_trn = sample(nrow(training_data), nrow(training_data)*1)
set.seed(1)
index_tst = sample(nrow(training_data), nrow(training_data)*0.1)
training_trn = training_data[index_trn,]
training_tst = training_data[index_tst,]
```

create RMSLE function
```{r rmsle function}
cal_rmsle = function(act, pred) {
  sqrt(mean((log(pred + 1) - log(act + 1))^2))
}
```

linear model with all the predictors
```{r linear regression-full model}
```

```
training_lm = lm(units ~., data = training_trn)
summary(training_lm)
par(mfrow = c(2,2))
plot(training_lm)
```

Check the constant variance assumption for the errors
```{r}
plot(fitted(training_lm), resid(training_lm),
  col = "grey", pch = 20,
  xlab = "Fitted Values", ylab = "Residuals",
  main = "Residuals vs Fitted Values")
abline(h = 0, col = "darkorange", lwd = 2)
library(lmtest)
bptest(training_lm)
```
Check the normality assumption for the errors
```{r normality}
qqnorm(resid(training_lm), main = "Normal Q-Q Plot", col = "darkgrey")
qqline(resid(training_lm), col = "dodgerblue", lwd = 2)
```

Outliers
```{r outliers}
Bonferroni_Critical_Value = qt(0.05/nrow(training_trn), 23391)
BCV = Bonferroni_Critical_Value
sum(rstudent(training_lm)[abs(rstudent(training_lm)) > abs(BCV)] != 0)
```

Influential points
```{r influential points}
cd1<- cooks.distance(training_lm) > 1
cd1[cd1 == TRUE]
```

Write prediction into csv file
```{r}
lm_prediction = ifelse(round(predict(training_lm, testing_data)) < 0, 0, predict(training_lm,
testing_data))

prediction_lm_csv <- data.frame(lm_prediction)
write.csv(prediction_lm_csv, file = "prediction_lm_all.csv")
```

Linear model with log transformation
```{r log linear regression-full model}
training_lm_log = lm(log(1+units) ~., data = training_trn)
summary(training_lm_log)
#plot(training_lm)
```

Write prediction into csv file
```{r}
lm_prediction = ifelse(exp(predict(training_lm_log, testing_data))-1 < 0, 0,
exp(predict(training_lm_log, testing_data))-1)

prediction_lm_csv <- data.frame(lm_prediction)
write.csv(prediction_lm_csv, file = "prediction_lm_log_all.csv")
```

```{r ridge-full model}
training_lm_ridge = cv.glmnet(model.matrix(training_lm), log(1+training_trn$units), alpha = 0,
nfolds = 5, lambda = seq(0.001: 0.01, by = 0.005))
summary(training_lm_ridge)
#plot(training_lm)
training_lm_ridge$lambda.min
0.031
plot(training_lm_ridge)
```

```{r lasso-full model}
training_lm_lasso = cv.glmnet(model.matrix(training_lm)[,-1], log(1+training_trn$units), alpha =
1, nfolds = 5, lambda = seq(0.001: 0.01, by = 0.005))
summary(training_lm_lasso)
#plot(training_lm)
plot(training_lm_lasso)
coef(training_lm_lasso)
```

linear model using backward selection
```{r stepwise-backward witohut log trans}
training_lm_step = step(training_lm, trace = 0)
```

Write prediction into csv file
```{r}
lm_step_prediction = ifelse(predict(training_lm_step, testing_data) < 0, 0,
predict(training_lm_step, testing_data))

write.csv(data.frame(lm_step_prediction),
        file = "prediction_lm_step_all.csv")
```

linear model with log transformation using backward selection
```{r}
training_lm_log_step = step(training_lm_log, trace = 0)
```

Write prediction into csv file
```{r}
```

```
lm_step_log_prediction = ifelse(exp(predict(training_lm_log_step, testing_data))-1 < 0, 0,
exp(predict(training_lm_log_step, testing_data))-1)

write.csv(data.frame(lm_step_log_prediction),
        file = "prediction_lm_step_log_all.csv")
```

linear model with interaction terms using log transformation
```{r}
training_lm_log_int = lm(log(units+1) ~. + I(is_holiday_and_weekend:store_nbr) +
I(is_weekend:store_nbr) + I(is_holiday_and_weekend:item_nbr) + I(is_weekend:item_nbr) +
I(year:store_nbr) + I(year:item_nbr), data = training_trn)
```

Write prediction into csv file
```{r}
lm_step_log_int_prediction = ifelse(exp(predict(training_lm_log_int, testing_data))-1 < 0, 0,
exp(predict(training_lm_log_int, testing_data))-1)

write.csv(data.frame(lm_step_log_int_prediction),
        file = "prediction_lm_log_int_all.csv")
```