CAPSTONE PROJECT

Rossmann Store Sales Prediction

APRIL 4, 2018

Written by: Mu(Marvin) Yuan

Contents

Definition

The Rossmann Sales Prediction project of Kaggle competition was chosen for the final project of Udacity Machine Learning Engineer Nanodegree. Rossmann is a chain drug store which is the second largest drugstore chain with over 3600 retailer stores in Europe. The headquarter was founded in 1972[(Jean-Pierre, 2017)]. This competition is to predict the drug store's 6 weeks ahead daily sales for 1115 Germany stores.

Problem Overview and Statement

The competition is to use the given features to predict sale performance 6 weeks ahead. The machine learning algorithm will be used for the sales prediction. There are several types of learning methods such as supervised learning, unsupervised learning, reinforcement learning and so on. Supervised learning is the data munging task of inferring the output from labeled training data. The training data consist of a set of label input object and a desired output value pairs. A supervised learning algorithm analyzes the training data and produces an inferred method, which can be used for mapping new examples. An unsupervised learning is that of trying to find hidden structure in unlabeled inputs. Since the examples given to the learner are unlabeled, there is no error or reward signal to determine the optimal solution. A reinforcement learning allows software agents or machines to automatically determine the ideal behavior within a specific context (based on states, actions, and rewards), to maximize its performance. Most of the algorithms are based on statistical models with mathematic functions.

Throughout the prediction methods, the learning methods mainly deal with two types of problems or two types of outputs: classification problems and regression problems. A classification problem characterizes the unlabeled inputs into different given classes, while a regression problem generates unlabeled inputs to numerical values coordinating with functions.

The Rossmann sales prediction problem is a regression problem with 14 different features to predict sales. From the data given, it is a supervised learning problem since the output (sales) is given. The inputs are in different types, such as numeric variables and categorical variables. Numeric variables included continuous variables such as data and discrete variable such as sales. Categorical variable included ordinal variables such as assortment and nominal variable such as promo to indicate if a specific store running a promo that day.

The given data are formed from 13 different features as shown in Table 1. The project is to use 12 features out of 13 to predict the last one feature, sales.

Table 1: Drugstore sale prediction feature list [(Rossmann(Company), 2015)]

Feature Names	Description
Store	a unique Id for each store
Sales	the turnover for any given day
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
DayOfWeek	From 1 to 7 correlating with Monday to Sunday, respectively

Metrics

The metric for success is via Kaggle's leaderboard which is based on accuracy compared with the hidden labels. There are several terms involving the evaluation methods: accuracy, precision, recall, etcetera. Accuracy measures how often the classifier makes the correct prediction. It's the ratio of the number of correct predictions to the total number of predictions (the number of test data points).

Equation 1: Accuracy demonstration function

$$\frac{\text{True Positives} + \text{False Negatives}}{(\text{True Positives} + \text{False Positives})}$$

Precision tells us what proportion of messages we classified as spam, were spam. It is a ratio of true positives (words classified as spam, and which are spam) to all positives (all words classified as spam, irrespective of whether that was the correct classification), in other words, it is the ratio of

Equation 2: Precision demonstration function

$$\frac{\text{True Positives}}{(\text{True Positives} + \text{False Positives})}$$

Recall(sensitivity) tells us what proportion of messages that were spam were classified by us as spam. It is a ratio of true positives (words classified as spam, and which are spam) to all the words that were spam, in other words, it is the ratio of

Equation 3: Recall demonstration function

$$\frac{\text{True Positives}}{(\text{True Positives} + \text{False Negatives})}$$

For classification problems, the F-beta score or receiver operating characteristic(ROC) curve can be used to evaluate the training score due to different circumstances:

Equation 4: F-beta score equation

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

where the β parameter determines the weight of precision in the combined score.

For regression problems, coefficient of determination(R^2) is a well-known mathematics option:

Equation 5: coefficient of determination(R^2) equation

$$R^2 = 1 - \frac{\sum_i (y_1 - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

where $\hat{y}_i$ is the prediction value, y_i is the actual value and n is the amount of predictions, $\bar{y}$ is the mean value from all observed outputs. The evaluation metric is R2 score, which is for evaluating correlation. The range of R2 score is from 0 to 1, a closer to 1 score indicates a better metric hyperparameter set.

Decision trees will be the main statistical model, which is based on decision rules. The decision tree algorithms will decide the best decision rules. In this project, random forest and extreme gradient boost decision trees(XGBoost) will be used for training. Another common of random forest and XGBoost is that they all are ensemble learning algorithms. Decision trees is also known as classification tree, and it has been used in different fields, such as classify spam. Decision trees is a robust, scalable classifier, able to estimate non-linear decision boundaries and it can prevent overfit due to its hierarchical structure. Decision trees are especially good at different circumstances. Individual trees are prone to overfitting due to its growing branches. However, this condition can be alleviated by ensemble methods such as random forest. Ensemble method combines different algorithm together with a common mathematic model. There are countless advantages through ensemble method. Invariant to scaling of inputs to eliminate heavy features normalization, higher order interaction between features, scalable and used in Industry. Putting into context, a ensembled decision trees can be written as:

Equation 6: Algorithms mathematic model

$$Model: \hat{y}_i = \sum_{k=1}^{K} f_k(x_i) \ , f_k \in F$$

Equation 7: Ensemble object mathematic function

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

Assume the number of trees is k. l means the training loss of the model and it controls the accuracy of the model, Ω means the regulation of the model and it controls the complexity of the trees as well as prevent overfitting.

Analysis

Kaggle provides the dataset online directly. For instant responses, Jupyter notebook is the main platform of analysis tool throughout this report. The report will follow a standard workflow:

1. Data mining: understand the data by viewing basic raw data information
2. Exploratory data analysis and visualization: data preprocessing, munging
3. Benchmark or naïve predictor setup
4. Model selection
5. Tuning hyperparameters
6. Output the results

Data Mining

The 13 features come from two datasets: 'train.csv' and 'store.csv' separately, and the general information shown in Figure 1 through Figure 3:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1017209 entries, 0 to 1017208
Data columns (total 9 columns):
Store            1017209 non-null int64
DayOfWeek        1017209 non-null int64
Date             1017209 non-null object
Sales            1017209 non-null int64
Customers        1017209 non-null int64
Open             1017209 non-null int64
Promo            1017209 non-null int64
StateHoliday     1017209 non-null object
SchoolHoliday    1017209 non-null int64
dtypes: int64(7), object(2)
memory usage: 69.8+ MB
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1115 entries, 0 to 1114
Data columns (total 10 columns):
Store                        1115 non-null int64
StoreType                    1115 non-null object
Assortment                   1115 non-null object
CompetitionDistance          1112 non-null float64
CompetitionOpenSinceMonth    761 non-null float64
CompetitionOpenSinceYear     761 non-null float64
Promo2                       1115 non-null int64
Promo2SinceWeek              571 non-null float64
Promo2SinceYear              571 non-null float64
PromoInterval                571 non-null object
dtypes: float64(5), int64(2), object(3)
memory usage: 87.2+ KB
```

Figure 1: dataset descriptive information of train dataset(left) and store dataset(right)

	count	mean	std	min	25%	50%	75%	max
Store	1017209.0	558.429727	321.908651	1.0	280.0	558.0	838.0	1115.0
DayOfWeek	1017209.0	3.998341	1.997391	1.0	2.0	4.0	6.0	7.0
Sales	1017209.0	5773.818972	3849.926175	0.0	3727.0	5744.0	7856.0	41551.0
Customers	1017209.0	633.145946	464.411734	0.0	405.0	609.0	837.0	7388.0
Open	1017209.0	0.830107	0.375539	0.0	1.0	1.0	1.0	1.0
Promo	1017209.0	0.381515	0.485759	0.0	0.0	0.0	1.0	1.0
SchoolHoliday	1017209.0	0.178647	0.383056	0.0	0.0	0.0	0.0	1.0

Figure 2: 'train.csv' dataset statistical information 'train.describe().T' command

	count	mean	std	min	25%	50%	75%	max
Store	1115.0	558.000000	322.017080	1.0	279.5	558.0	836.5	1115.0
CompetitionDistance	1112.0	5404.901079	7663.174720	20.0	717.5	2325.0	6882.5	75860.0
CompetitionOpenSinceMonth	761.0	7.224704	3.212348	1.0	4.0	8.0	10.0	12.0
CompetitionOpenSinceYear	761.0	2008.668857	6.195983	1900.0	2006.0	2010.0	2013.0	2015.0
Promo2	1115.0	0.512108	0.500078	0.0	0.0	1.0	1.0	1.0
Promo2SinceWeek	571.0	23.595447	14.141984	1.0	13.0	22.0	37.0	50.0
Promo2SinceYear	571.0	2011.763573	1.674935	2009.0	2011.0	2012.0	2013.0	2015.0

Figure 3: 'store.csv' dataset statistical information with 'store..describe().T'' command

The figures show the values information from the labels showing in rows and different statistical information showing in columns, where count is the sum-up number for the corresponsive label value, mean is the average value through the corresponsive label value, std is the standard deviation, min is the minimum, 25% is the one quarter tile, 50% is the median value, 75% is the third quarter tile and max is the maximum value. From Figure 1, it indicates that train dataset doesn't have any null cells but store dataset does.

From train dataset, the comparison between mean and median is quite close, which means the data distribution falls within a normal distribution. On the other hand, it is skewed for label competition distance of store dataset.

Apparently, they are not in the same shape, train dataset is much more than store dataset. However, these two datasets share one same feature: Store, which is the unique ID of a specific store. The store and train dataset combined with a unique ID (feature name: Store) into another new dataset named 'data'.

Exploratory Data Analysis and visualization

This section will focus on the data exploration and visualization, the predictor data will be prepared for preprocessing after this section. Exploratory data analysis is a statistical analysis of data sets to summarize the dataset's main characteristics with visual methods. For a better presentation, train and store datasets will be combined into a new dataset for analysis.

From data mining section, it is necessary to proceed both data concatenation. By the merge command, train and store dataset were merged together:

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
Store	1.01721e+06	NaN	NaN	NaN	558.43	321.909	1	280	558	838	1115
DayOfWeek	1.01721e+06	NaN	NaN	NaN	3.99834	1.99739	1	2	4	6	7
Date	1017209	942	2013-02-27	1115	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Sales	1.01721e+06	NaN	NaN	NaN	5773.82	3849.93	0	3727	5744	7856	41551
Customers	1.01721e+06	NaN	NaN	NaN	633.146	464.412	0	405	609	837	7388
Open	1.01721e+06	NaN	NaN	NaN	0.830107	0.375539	0	1	1	1	1
Promo	1.01721e+06	NaN	NaN	NaN	0.381515	0.485759	0	0	0	1	1
StateHoliday	1017209	4	0	986159	NaN	NaN	NaN	NaN	NaN	NaN	NaN
SchoolHoliday	1.01721e+06	NaN	NaN	NaN	0.178647	0.383056	0	0	0	0	1
StoreType	1017209	4	a	551627	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Assortment	1017209	3	a	537445	NaN	NaN	NaN	NaN	NaN	NaN	NaN
CompetitionDistance	1.01457e+06	NaN	NaN	NaN	5430.09	7715.32	20	710	2330	6890	75860
CompetitionOpenSinceMonth	693861	NaN	NaN	NaN	7.22287	3.21183	1	4	8	10	12
CompetitionOpenSinceYear	693861	NaN	NaN	NaN	2008.69	5.99264	1900	2006	2010	2013	2015
Promo2	1.01721e+06	NaN	NaN	NaN	0.500564	0.5	0	0	1	1	1
Promo2SinceWeek	509178	NaN	NaN	NaN	23.2691	14.096	1	13	22	37	50
Promo2SinceYear	509178	NaN	NaN	NaN	2011.75	1.66287	2009	2011	2012	2013	2015
PromoInterval	509178	3	Jan,Apr,Jul,Oct	293122	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Year	1.01721e+06	NaN	NaN	NaN	2013.83	0.777396	2013	2013	2014	2014	2015
Month	1.01721e+06	NaN	NaN	NaN	5.84676	3.3261	1	3	6	8	12
Day	1.01721e+06	NaN	NaN	NaN	15.7028	8.78764	1	8	16	23	31
WeekOfYear	1.01721e+06	NaN	NaN	NaN	23.6155	14.4334	1	11	22	35	52

Figure 4: train and store dataset merge and null values summary

In Figure 4, four features from Date created into the merged dataset as Year, Month, Day, WeekOfYear. Since all the data has a date, the index will be reset to date with a set_index command[1] for a convenience sort. From the figure 4, there are 4 none-date categorical valuables: StateHoliday, StoreType, Assortment, and PromoInterval, and the representing violin plots are shown in figure 5 and figure 6 below. Violin plot is a type of plot representing categorical values horizontally and numerical value vertically. The violin plot is similar to box plot with a rotated kernel density plot on each side. Recall that 'a b, c,0' indicate a public holiday, Easter holiday, Christmas and None for StateHoliday feature, respectively. Apparently, a drugstore won't open when it comes to a State Holiday, leading to a zero sale for that day. 'a,b,c,d' store types present four different store models for StoreType feature. 'a,b,c' in assortment represent an assortment level, basic, extra and extended, respectively. The sales of store type b have a larger range than the other three and also has a larger median value. The months showing in PromoInterval describes the consecutive intervals Promo2 feature is started. From the violin plot, there isn't a significant difference between each interval, which basically means PromoInterval generate less information than other features.

[1] For detailed jupyter notebook code, please refer the Appendix

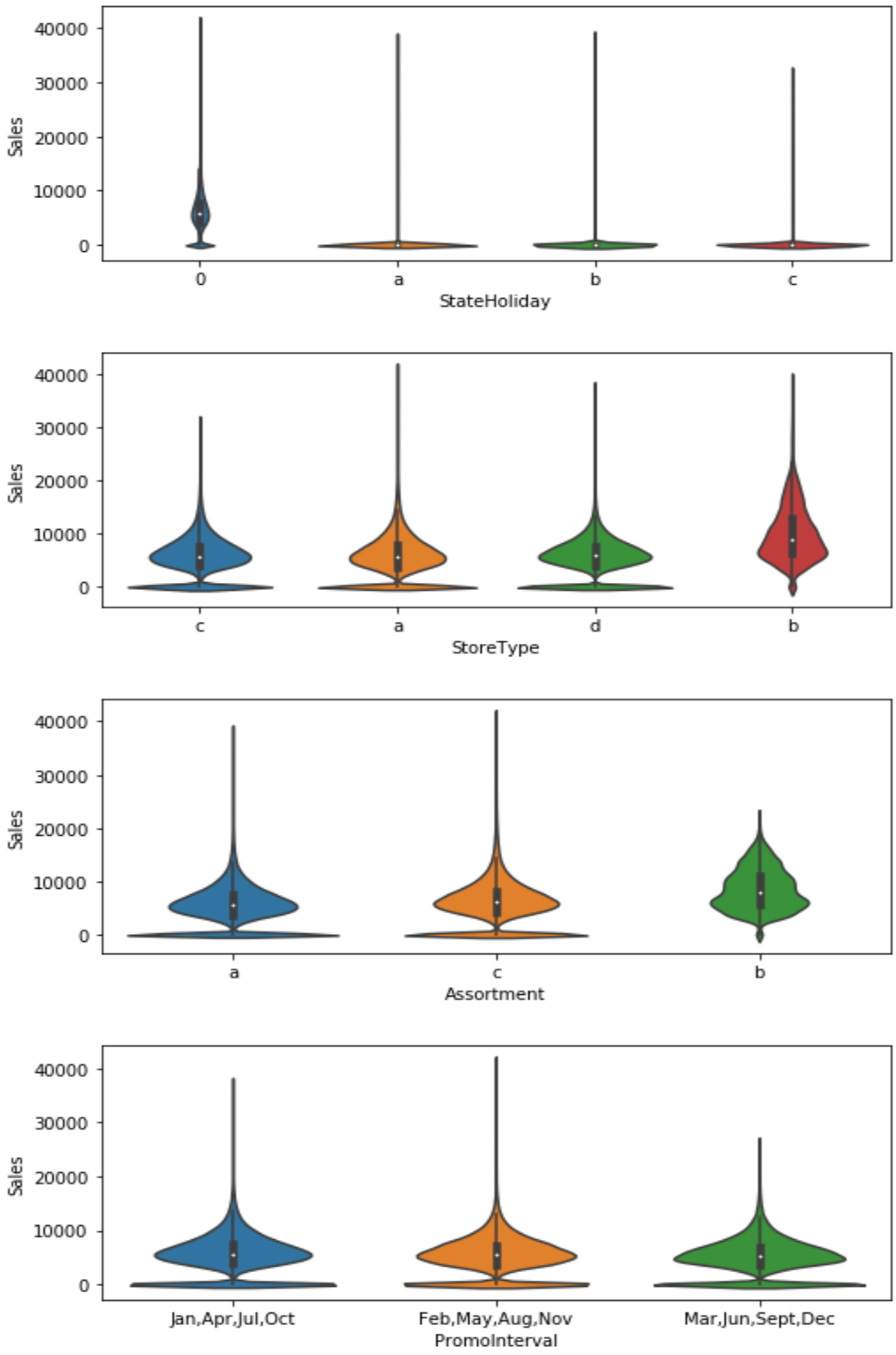

Figure 5: Violin plots showing State Holiday, Store Type, Assortment and Promo interval features

With the heat map plotted, a feature-wise correlation map displays in Figure 6. For strongly correlated features, discard those highly related labels will not only bring a similar result but less expensive computation for the servers.

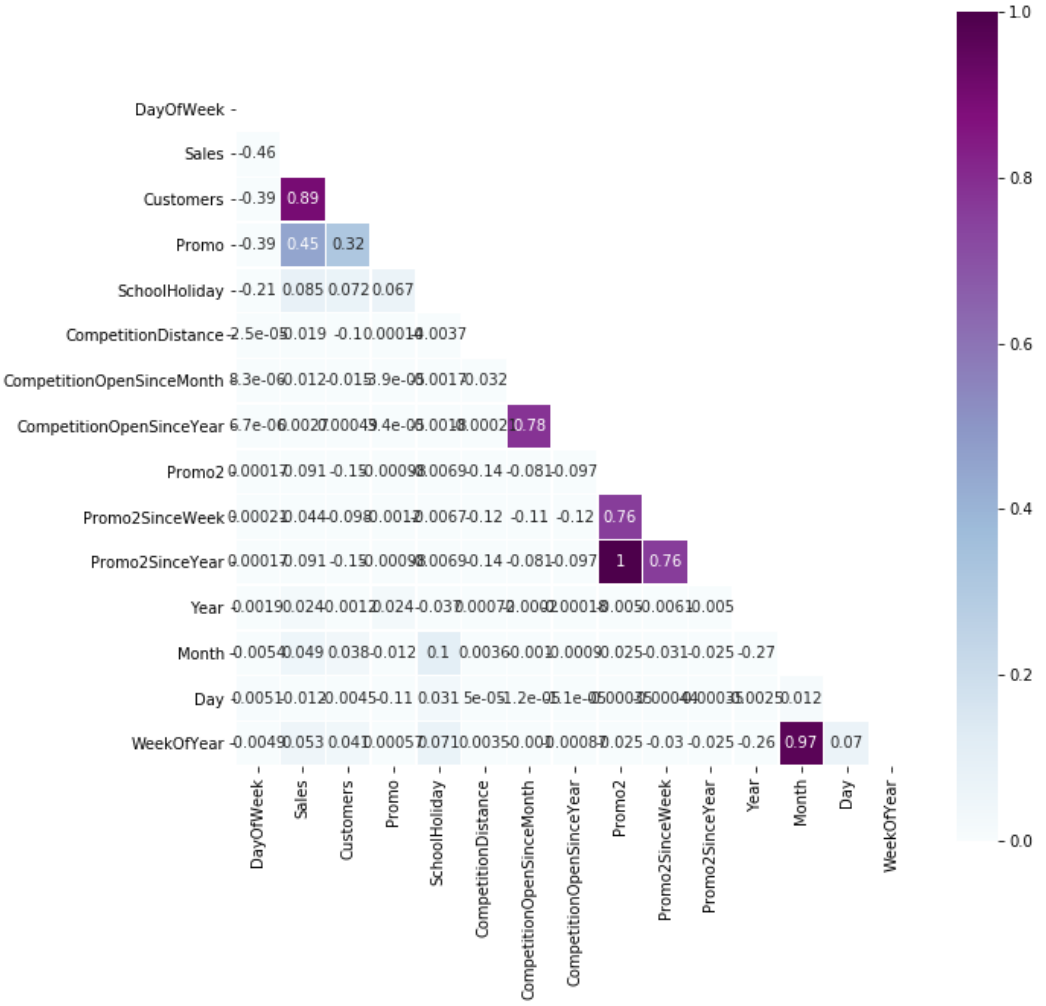

Figure 6: Heatmap for selected features' correlations

Gathering insights from figure 5 and figure 6, a factor plot with 'Assortment, StoreType and Promo' three features display in figure 7. The row-wise comparison shows an approximately 25% growth, while over 100% growth showing in column-wisely. For Store type b, extended assortment (assortment: c) is about 50% more than another type assortment. In addition, the trend rises while time runs in each graph. After gets a general idea about the graph, it's time to start choosing a proper algorithm to train the data.

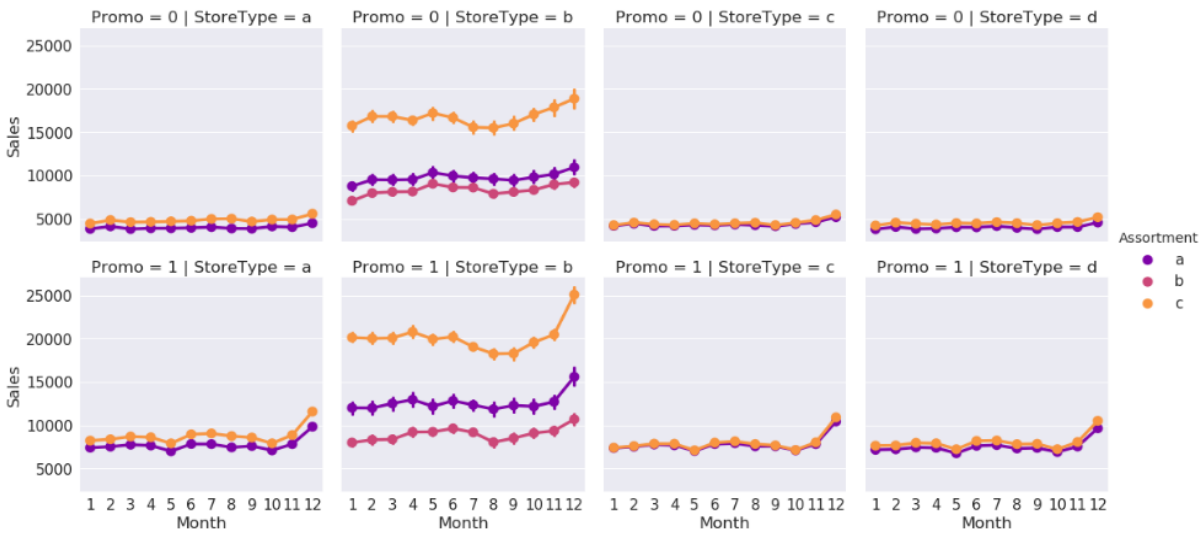

Figure 7: Factor plot with column figures showing Store type, row figures showing Promo

Algorithms and Techniques

Clearly, the Rossmann sales prediction is a structural supervised problem. The solution should be written as a python code file with a supervised learning algorithm. Due to research, one of the potential algorithms can be XGBoost, although the other algorithms will be used for training as well. The result will be shown in Kaggle leaderboard and accuracy.

The algorithms for this regression problem to evaluate are random forest and XGBoost. XGBoost is a kind of gradient boosting algorithm, while the random forest is a tree-like an algorithm.

Gradient boosting produces a prediction model in the form of an ensemble of weak prediction models, called weak learner, typically decision trees. It builds the model in a stage-wise and it generalizes weak learners by allowing optimization of an arbitrary differentiable loss function.

Decision trees are also known as classification tree, and it has been used in different fields, such as classify spam. Decision trees are a robust, scalable classifier, able to estimate non-linear decision boundaries and it can prevent overfit due to its hierarchical structure. 'Modern Machine Learning Algorithms: Strengths and Weaknesses' Decision trees are especially good at different circumstances. Individual trees are prone to overfitting due to its growing branches. However, this condition can be alleviated by ensemble methods such as a random forest. Random forest is one of the most famous algorithms using in Kaggle, which is a side point to prove this algorithm being a good candidate for the problem. (Hastie, 2008).

Both gradients boosting and random forest are based on decision trees, which share some common parameters. For random forest referring to the algorithm document, ‘n_estimators, max_features, min_sample_split, min_samples_leaf and max_leaf_nodes’ all contents

default value, which should be tuned for the optimal modeling process. For XGBoost, ‘use_buffer, num_round, eta, gamma, max_depth, max_leaves, max_bin’ are the main hyperparameters for tuning. After the determination of algorithms, it is time to decide what makes a model approved.

Benchmark

A benchmark model supposed to be fast and high performance. By measurable terms, the error score should be less than 0.117. After testing, the training time is proportional with model accuracy since each round of training takes time. A benchmark model will be evaluated from several different supervised learning algorithms.

Since it is a Kaggle competition, climbing to a higher place on the leaderboard is the ultimate goal. Behind the scenes, what the leaderboard drives is the accuracy of the model. After quantizing the optimal goal, it is easier to understand what winning is. The approved goal is to achieve first 10% of the leaderboard after submitting the prediction values from test dataset. The benchmark model built up from one of kernel in Kaggle shown in Appendix.

Methodology

In the methodology section, a jupyter notebook workflow will generate more insights by revealing the process from data preprocesses to predictive outputs, rather than descriptive statements.

Data Preprocessing

The data preprocessing consists of the following steps:

1. Feature selection
2. Fill missing values
3. Normalize numerical values
4. One-hot encoding
5. Shuffle and split data

Before data can be used as input for machine learning algorithms, it often must be cleaned, formatted, and restructured. The process of preparing the training data is also known as data preprocessing. There are some missing values and qualities about certain features that must be adjusted. The data preprocessing can help tremendously with the outputs and predictive power of nearly all learning algorithms.

Implementation

```
# input extra features
data=pd.merge(train,store,on='Store')
```

```
train_data=data.set_index(pd.DatetimeIndex(data['Date']),inplace=False)
# data extraction
train_data['Year'] = train_data.index.year
train_data['Month'] = train_data.index.month
train_data['Day'] = train_data.index.day
train_data['WeekOfYear'] = train_data.index.weekofyear
```

```
train_data['CompetitionDistance'].fillna(train_data['CompetitionDistance'].median(), inplace = True)
train_data.fillna(0,inplace=True)
train_data['CompetitionOpen'] = 12 * (train_data.Year - train_data.CompetitionOpenSinceYear) + \
(train_data.Month - train_data.CompetitionOpenSinceMonth)
train_data['PromoOpen'] = 12 * (train_data.Year - train_data.Promo2SinceYear) + \
(train_data.WeekOfYear - train_data.Promo2SinceWeek) / 4.0
```

```
# feature selection to drop repeating columns
label_data = train_data['Sales'] # Seperate label from training dataset
droplabels=['PromoInterval','Date','Sales','Customers','StateHoliday']
feature_data = train_data.drop(droplabels, axis = 1)
```

```
# Import sklearn.preprocessing.StandardScaler
from sklearn.preprocessing import MinMaxScaler

# Initialize a scaler, then apply it to the features
scaler = MinMaxScaler() # default=(0, 1)
numerical = ['WeekOfYear','CompetitionDistance']

features_data_transform = pd.DataFrame(data = feature_data)
features_data_transform[numerical] = scaler.fit_transform(features_data_transform[numerical])

# Show an example of a record with scaling applied
display(features_data_transform.head(n = 5))
```

Figure 8: the Implemental procedure of data preprocessing

In figure 9, ‘data’ data frame is the combined dataset from ‘train’ and ‘store’ datasets. ‘test_data’ data frame is the combined dataset from ‘test’ and ‘store’ datasets. Feature selection is very beneficial when dealing with a large amount of data. StateHoliday, DayOfWeek contains duplicates information with Date. From exploratory data analysis section, PromoInterval didn’t contain much variance/information from violin plots. Sales are the output labels. Those labels in ‘data’ are exclusive to ‘test_data’ dataset should be removed, since the only useful features are in ‘test’ dataset. There are several labels including missing values, the goal is to fill missing value with least alternation of the data’s statistical result. For CompetionDistance label, median value filled into the blank spots, and the rest filling with zeros.

Typically, learning algorithms expect input to be numeric, which requires that non-numeric features (called categorical variables) be converted. One popular way to convert categorical variables is by using the one-hot encoding scheme. One-hot encoding creates a "dummy" variable for each possible categorical feature. For example, assume theFeature has three possible entries: A, B, or C. One-hot encoder generates theFeature into theFeature _A, theFeature _B and theFeature _C three columns. From exploratory data analysis section, categorical feature: StoreType and Assortment are the only un-binary non-numeric valuables, thus these three features are suitable for one-hot-encoding.

```
# One-hot encode the 'features_log_minmax_transform' data using pandas.get_dummies()
features_final = pd.get_dummies(features_data_transform,columns=['StoreType','Assortment'])


# Print the number of features after one-hot encoding
encoded = list(features_final.columns)
print ("{} total features after one-hot encoding.".format(len(encoded)))
```

Figure 9: One-hot-encoding for selected features

The encoder for the feature labels was once changed from one-hot encoder to label encoder. Comparing with one-hot encoder, label encoder will exam target series and convert them to natural numbers. For instance, assume theFeature has three possible entries: A, B, or C. Label encoder will encode A, B and C as 0,1 and 2, respectively, therefore theFeature will fill with numerical numbers instead of letters.

Now all categorical variables have been converted into numerical features, and all numerical features have been normalized. In machine learning, using test dataset is cheating. For training purposes, train and test sets will usually be sliced from the original dataset (both features and their labels). 80% of the data will be used for training and 20% for testing.

```
# Import train_test_split
from sklearn.cross_validation import train_test_split

# Split the 'features' and 'income' data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features_final,
                                                    label_data,
                                                    test_size = 0.2,
                                                    random_state = 77)

# Show the results of the split
print ("Training set has {} samples.".format(X_train.shape[0]))
print ("Testing set has {} samples.".format(X_test.shape[0]))
```

```
/home/nbuser/anaconda3_420/lib/python3.5/site-packages/sklearn/cross_validation.py:44: DeprecationWarning: This module was depr
ecated in version 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Al
so note that the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)

Training set has 813767 samples.
Testing set has 203442 samples.
```

Figure 10: Dataset shuffle and split

Now the datasets are ready to train a machine learning model.

Refinement

During the process of inserting test data, the number of test's features should be the same size as the model requirement. Therefore, if there isn't a way to fill out the features in 'store' dataset, the prediction won't a success. The origin model that trained from only 'train' dataset got a score of 4.39867. The Kaggle submission score is based on root-mean-squared percentage error, the goal is to get the score as small as possible. The process to evaluate a model is from elapsed time and the r^2 score with default settings, shown in figure 12:

```
# create a decorater to measure the model training and predicting elapsed time
import time
from sklearn.metrics import r2_score

def elapsed_time(f):
    def wrapper(*args,**kwargs):
        t1=time.time()
        f(*args,**kwargs)
        t2=time.time()
        print('Elapsed time:{0:.2f} ms,{1:.2f} mins'.format((t2-t1) * 1000,(t2 - t1) / 60))
    return wrapper

@elapsed_time
def pipeline(learner,X_train,y_train,X_test,y_test):
    regr = learner(random_state=77)
    regr.fit(X_train, y_train)
    r2=r2_score(y_test, regr.predict(X_test))
    return print("For this algorithm, the percentage error of R2 is {0:.2f}".format((1-r2)*100))
```

```
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
pipeline(RandomForestRegressor,X_train,y_train,X_test,y_test)
pipeline(XGBRegressor,X_train,y_train,X_test,y_test)
```

Figure 11: Evaluation model procedures

After merging test and store dataset together as shown in figure 13, the RMS error on Kaggle successfully decreased to 0.77012 without any hyperparameter tuning. For this moment, the data process complete, and tuning the parameters is the next step.

Prepare the test dataset

```
test_data=pd.merge(test,store,how = 'inner',on='Store')
```

```
test_data.set_index(pd.DatetimeIndex(test_data['Date']),inplace=True)
# test dataset extraction
test_data['Year'] = test_data.index.year
test_data['Month'] = test_data.index.month
test_data['Day'] = test_data.index.day
test_data['WeekOfYear'] = test_data.index.weekofyear
#test_data.isnull().sum()
```

```
test_data['CompetitionDistance'].fillna(test_data['CompetitionDistance'].median(), inplace = True)
test_data['Open'].fillna(1, inplace = True)# assume store open if missing
test_data.fillna(0,inplace=True)
test_data['CompetitionOpen'] = 12 * (test_data.Year - test_data.CompetitionOpenSinceYear) + \
(test_data.Month - test_data.CompetitionOpenSinceMonth)
test_data['PromoOpen'] = 12 * (test_data.Year - test_data.Promo2SinceYear) + \
(test_data.WeekOfYear - test_data.Promo2SinceWeek) / 4.0
```

```
# feature selection to drop repeating columns
droptest=['PromoInterval','Date','Id','StateHoliday']
test_data.drop(droptest, axis = 1,inplace=True)
```

```
test_data_transform = pd.DataFrame(data = test_data)
transferlabels = ['WeekOfYear','CompetitionDistance']
test_data_transform[transferlabels] = scaler.fit_transform(test_data_transform[transferlabels])
```

```
# One-hot encode the 'features_log_minmax_transform' data using pandas.get_dummies()
test_final = pd.get_dummies(test_data_transform,columns=['StoreType','Assortment'])
```

Predict test dataset

```
regr = RandomForestRegressor(random_state=77)
regr.fit(X_train, y_train)
y_pred=regr.predict(test_final)
```

```
result = pd.DataFrame({"Id": test["Id"], 'Sales': y_pred})
result.to_csv("output.csv", index=False)
```

Figure 12: Prepare and predict test dataset

The percentage errors of R2 scores from random forest and XGBoost are 5.26% and 33.48%, respectively. The results assert that the performance of XGBoost is highly dependent on hyperparameter tuning. By applying hyperparameter-tune evaluation method, the trends of each target hyperparameters clearly narrow down numerical parameters to a certain range, Figure 14 showing the process with random forest algorithm.

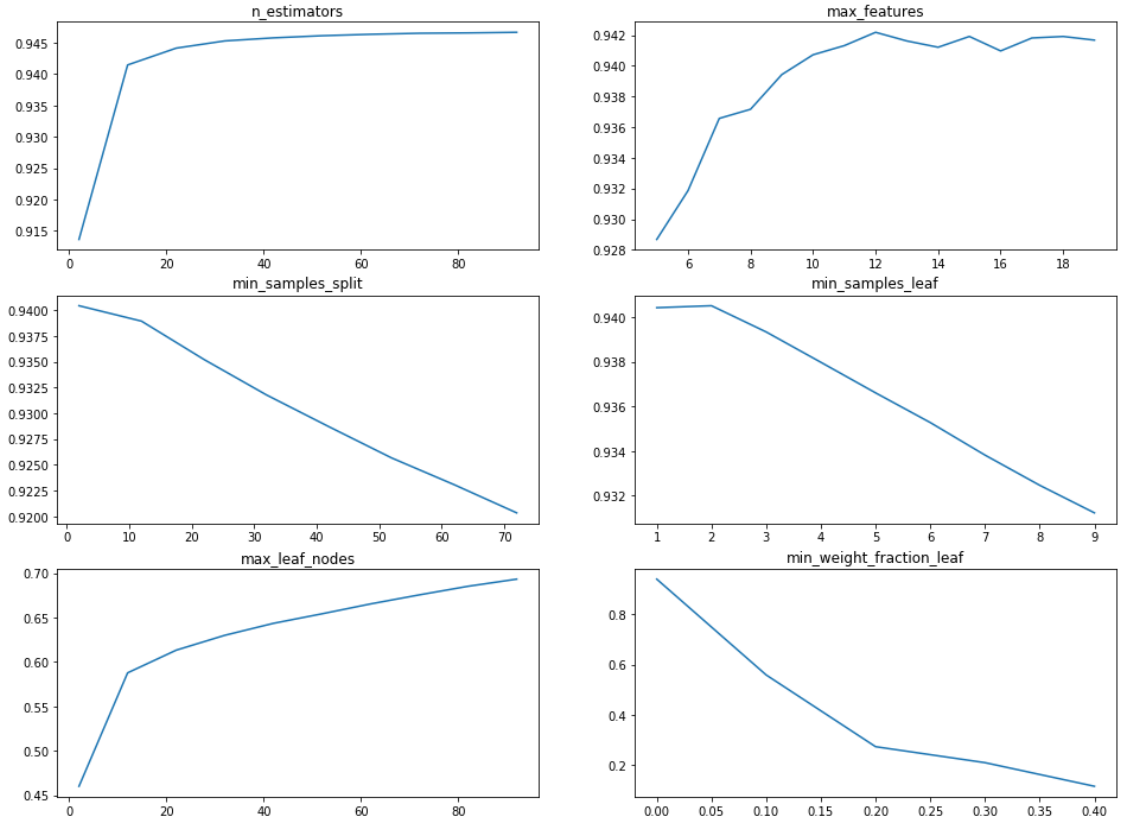

Figure 13: Parameter visualization of some key hyperparameters of Random Forest

The implemental code states in Appendix. Common parameters should apply the same trend to both algorithms, thus it is unnecessary to tune the same parameters again except the exclusive parameters for XGBoost. By optimal hyperparameters, the final results will lead to final Kaggle submission. By sklearn built-in package 'RandomizedSearchCV', the optimal parameters will compute. Due to consideration of the time spending on model optimal process, the amount of data is only sliced from year 2015 rather than total 2013 to 2015 three years. After the submission checks with Kaggle, there isn't significant improvement based on data amount proliferating from 2 to 3 years. On private score, training with full amount of data leads to a 0.7574 private score whereas 0.7578 for 2 years training data, and 0.7731 for only consuming data from 2015. Without other adjustments, a fill null method alternation increases the result from 0.7578 to 0.7449 on private leaderboard with only 2015 data for training.

```
params_sk = {
            'subsample':0.9,#add randomness to make training robust to noise
            'colsample_bytree':0.7,#add randomness to make training robust to noise
            'num_boost_round':300,
            'random_state':77
            }

params_grid = {
    'learning_rate': st.uniform(0.1, 0.9),
    'max_depth': np.arange(10, 20, 1),#control model complexity
    'gamma': st.uniform(0, 10),#control model complexity
    'reg_alpha': st.expon(0, 50),
    'min_child_weight':st.uniform(1, 10)
}

skrg = XGBRegressor(**params_sk)
skrg.fit(X_train, y_train)

search_sk = RandomizedSearchCV(skrg, params_grid,n_jobs=-1, cv = 5) # 5 fold cross validation
search_sk.fit(X_train, y_train)

# best parameters
print(search_sk.best_params_); print(search_sk.best_score_)
```

Figure 14: Grid search for some key hyperparameters tuning of XGBoost

Since the number of rounds is just a few hundreds, parameters tuning were using randomized search. However, the optimized parameters got a better public score result though a worse private score, which indicates worse overfitting than manually tuning. In this case, manually parameters tuning have better controls to the training and validation comparison.

Results

Model Evaluation and Validation

Due to limitation of vigor, the research of random forest will have to put off, research of XGBoost will be the main topic of this paper.

From model validation process of XGBoost algorithm, root mean squared errors from iterate over train and test datasets and create a larger and larger difference. There is a large gap between train error and test error indicating that overfitting occurs. In addition, the results assert an overfitting model with an over 400 RMSE difference between train and test datasets in 300 rounds (without log transformation). For controlling overfitting, inserting 'subsample' and 'colsample_bytree' reduces number of rounds and adds randomness to make training robust to noise. By decreasing 'learning rate' from 0.3 to 0.2 and increasing 'num_boost_round' from 100 to 300, the RMSE difference successfully dropped from 600 to 250(without log transformation). By controlling the overfitting, the Kaggle submission score dropped from 0.74 to 0.59. After the alternation of learning rate from 0.3 to 0.17, the score dropped from 0.59 to 0.55. However, the error is still quite high. Fortunately, a sudden breakthrough comes. After introduced another data merging process, the Kaggle score was steeply dropped from 0.55 to 0.15. The reason behind the error steeply falling is due to how the data are merging and information fills. The default data

frame merging process was inner joining, which will only keep the overlapping joined keys. By switch the merging process to train data, it will combine the information from store and train data set as well as assert each input with a label with it.

```
def build_features(features, data):
    # remove NaNs
    data.fillna(0, inplace=True)
    data.loc[data.Open.isnull(), 'Open'] = 1
    # Use some properties directly
    features.extend(['Store', 'CompetitionDistance', 'Promo', 'Promo2', 'SchoolHoliday'])

    # Label encode some features
    features.extend(['StoreType', 'Assortment', 'StateHoliday'])
    mappings = {'0':0, 'a':1, 'b':2, 'c':3, 'd':4}
    data.StoreType.replace(mappings, inplace=True)
    data.Assortment.replace(mappings, inplace=True)
    data.StateHoliday.replace(mappings, inplace=True)

    features.extend(['DayOfWeek', 'Month', 'Day', 'Year', 'WeekOfYear'])
    data['Year'] = data.Date.dt.year
    data['Month'] = data.Date.dt.month
    data['Day'] = data.Date.dt.day
    data['DayOfWeek'] = data.Date.dt.dayofweek
    data['WeekOfYear'] = data.Date.dt.weekofyear
    # CompetionOpen en PromoOpen from https://www.kaggle.com/ananya77041/rossmann-store-sales/randomforestpython/code
    # Calculate time competition open time in months
    features.append('CompetitionOpen')
    data['CompetitionOpen'] = 12 * (data.Year - data.CompetitionOpenSinceYear) + \
        (data.Month - data.CompetitionOpenSinceMonth)
    # Promo open time in months
    features.append('PromoOpen')
    data['PromoOpen'] = 12 * (data.Year - data.Promo2SinceYear) + \
        (data.WeekOfYear - data.Promo2SinceWeek) / 4.0
    data['PromoOpen'] = data.PromoOpen.apply(lambda x: x if x > 0 else 0)
    data.loc[data.Promo2SinceYear == 0, 'PromoOpen'] = 0

    # Indicate that sales on that day are in promo interval
    features.append('IsPromoMonth')
    month2str = {1:'Jan', 2:'Feb', 3:'Mar', 4:'Apr', 5:'May', 6:'Jun', \
             7:'Jul', 8:'Aug', 9:'Sept', 10:'Oct', 11:'Nov', 12:'Dec'}
    data['monthStr'] = data.Month.map(month2str)
    data.loc[data.PromoInterval == 0, 'PromoInterval'] = ''
    data['IsPromoMonth'] = 0
    for interval in data.PromoInterval.unique():
        if interval != '':
            for month in interval.split(','):
                data.loc[(data.monthStr == month) & (data.PromoInterval == interval), 'IsPromoMonth'] = 1
    # Insert a feature to differ different zones from state holidays
    features.append('zone')
    for storeid in data.Store.unique():
        data.loc[(data.Store == storeid) & (data.StateHoliday == 0),'zone'] = 0
        zonenum=data.loc[(data.Store == storeid),'zone'].isnull().sum()
        # store the number to holidaycount feature
        data.loc[(data.Store == storeid)& (data.zone.isnull()),'zone'] = zonenum
    # alter the data type to a proper one
    altintfeatures=['SchoolHoliday','CompetitionOpenSinceMonth','CompetitionOpenSinceYear','Promo2SinceWeek','Promo2SinceYear','zone']
    data[altintfeatures]=data[altintfeatures].astype(np.int64)
    return data
```

Screenshot(Alt + A)

Figure 15 Improved feature engineering section, refer to Kaggle kerne from user cast42 [https://www.kaggle.com/cast42/xgboost-in-python-with-rmspe-v2/code]

In addition, the evaluation metric could be the miracle puzzle to train the metric down to acceptable region. After the `max_depth` increases, unsurprisingly, the public score drops where private score relatively increases. This attempt shows a strong relationship between overfitting and decision tree max depth. Apparently, the Kaggle error is still quite high comparing from the top 10% level, which lays in 0.11. From previous process, the actual values keep a strongly positive skewed distribution where might influent the training process as well as enhance non-accumulative additive random error. In this case, logarithm transformation is a well fit method to solve this problem. Since there are zeros in the sales values, np.log1p is a great way to deal with zero inputs by adding one for each entry. Of course, the output should use np.expm1 to subtract the addition one from each entries. By doing logarithm transformation, one side advantage is to narrow down the RMSE range

from hundreds to several decimal points less than one, which will provide a more intuitive way to show the performance of the model comparing with Kaggle's evaluation metric.

Justification

Since the error is still quite high reflecting with the submission goal, the data cleaning and feature engineering stages were reintroduced. During data cleaning, there is a stage somehow missing in the previous sections, which is outlier detection with interquartile range, the corresponding equation showing below in equation 8:

Equation 8 Interquartile range method for outlier detection

$$\text{Outlier range} = [\text{Q1} - 1.5 * \text{IQR}, \text{Q3} + 1.5 * \text{IQR}]$$

$$IQR = Q3 - Q1$$

Where IQR is the abbreviation of interquartile range, and Q1 is the 25% percentile, Q3 is the 75 percentile. After the outlier detection, the error actually increased with a suspicious reason of dropping useful information. In that case, the outlier detection won't include in the final file.

```
# train_data.head()
numlabel=['Sales','Customers','CompetitionDistance']
data_det=train_data[numlabel]
```

```
# Create the indices for data points for outliers
outliers=[]
outliers_indices=[]
# For each feature find the data points with extreme high or low values
for feature in data_det.keys():

    # Calculate Q1 (25th percentile of the data) for the given feature
    Q1 = np.percentile(data_det[feature],25)

    # Calculate Q3 (75th percentile of the data) for the given feature
    Q3 = np.percentile(data_det[feature],75)

    # Use the interquartile range to calculate an outlier step (1.5 times the interquartile range)
    step = 1.5*(Q3 - Q1)
    # Display the outliers
#     print ("Data points considered outliers for the feature '{}':".format(feature))
#     display(data_det[~((data_det[feature] >= Q1 - step) & (data_det[feature] <= Q3 + step))])
    # Create a list contains non repeated outliers indices to remove
    outliers_indices=outliers_indices+(list(data_det[~((data_det[feature] >= Q1 - step) & (data_det[feature] <= Q3 + step))][feature].keys()))

# Only keep indices occured more than once
outlier_indice=[indices for indices in outliers_indices if outliers_indices.count(indices)>=2]
# Count the unique elements in the all_outliers array
outliers = np.unique(outlier_indice);
```

Figure 16 Outlier detection using IQR on jupyter notebook

With the extra "zone" features showing in figure 15 comparing with figure 12, the local RMSE score was at 0.0116, and the Kaggle score was 0.125, and the difference between private score and public score from 0.016 to 0.011 thanks to the regulation parameter alpha. However, the result is highly overfitting due to limit validation data setting. After increase the validation data ratio from 0.012 to 0.1, the score from Kaggle is about 95% accuracy comparing with the final evaluation score in the training stage. The score private leaderboard score was about 0.127. The best score of the model is 0.11393 on public leaderboard where 0.12856 on private leaderboard. The public leaderboard score is higher than the benchmark score 0.117.

Conclusion

Free-Form Visualization

Figure 17 shows a clear relationship between model and features. There are several parts dividing the feature important map: time, store information and competition.

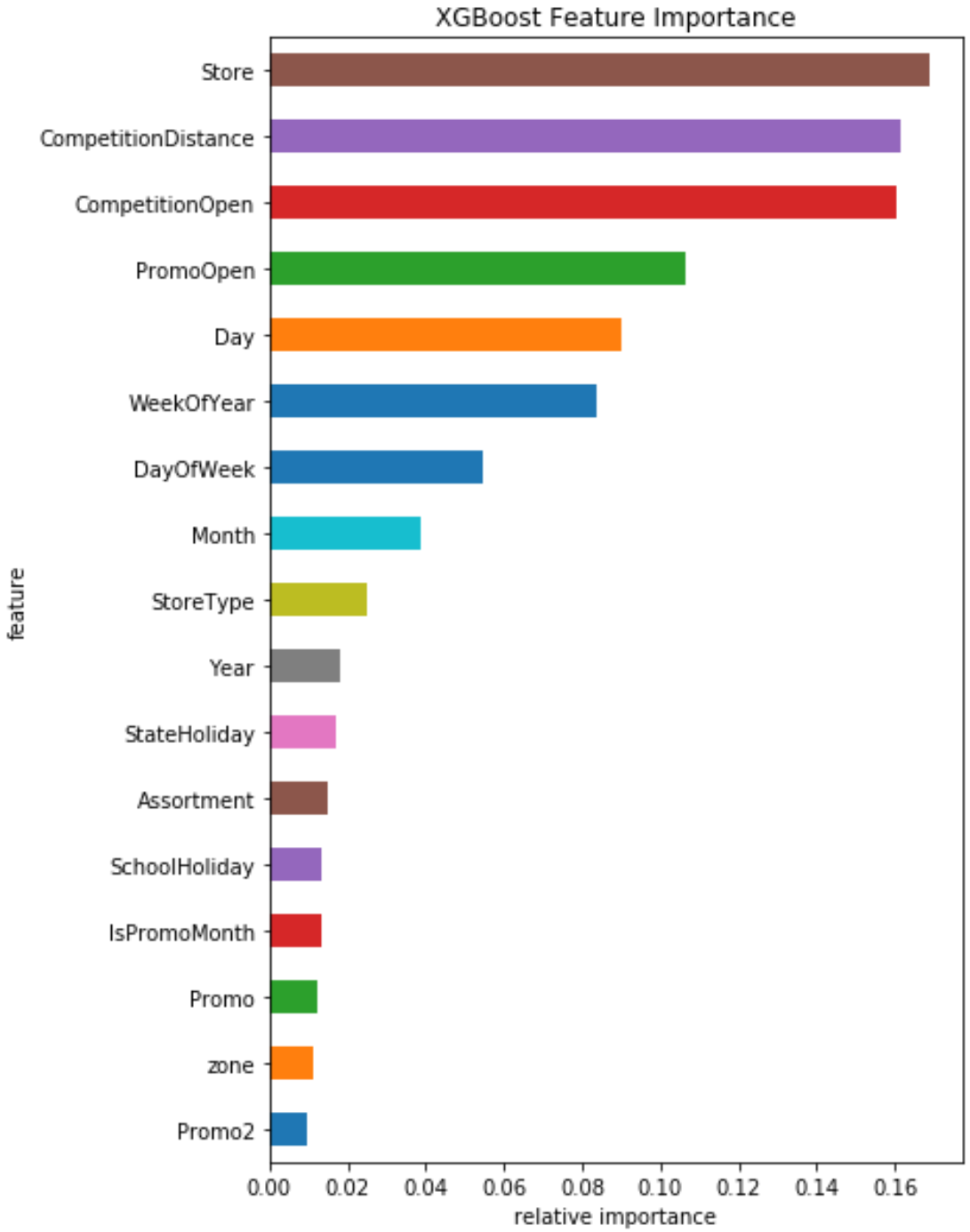

Figure 17 Feature importance

For this model from XGBoost, Store information plays the most important role in the model, such as Store id showing on the top of the list, whereas store id can stand for several different factors, such as store location. The second important feature is about competition details, competition distance and competition open. Surprisingly, the timestamps factors show less important than the other two parts.

Reflection

Rossmann Sales project demonstrated a typical sales problem through the whole process. The first step called data munging, the goal is to get familiar with the data. There are different categories in the data wrangling process such as data cleaning to take care of the meaningless values, exploratory data analysis to understand data's behaviors. Throughout the Rossmann project, it is important to take a glance at the raw data to get a first impression of the data. The impression better includes the data types of each features, what the label data type is. For instance, there are plenty of timestamp features displaying as float numbers, if these timestamp input as it was, the timestamp features will impact the result leading to a bad performance. Another example of data types could be the distinction between categorical variable and numerical variable. These different data types play tremendous roles in the feature engineer step afterwards. On the other hand, keeping an eye on the missing value and statistical summary of the data will provide insights for data preparation stage. During exploratory data analysis, domain knowledge mostly will play a key role to improve the understanding of data as well as build features. After understanding the data, a draft for the selection of features and model will be no fuzzy. For Rossmann sales prediction project, it is a supervised learning regression problem, which leads to a model selection of XGBoost and Random forest, since trees are good at structural problems.

After data munging, data preparation will take care of the null values or irregular values from the raw data. In Rossmann project, since the data has already been processed, only missing value need to perform in the data preparation stage. For convenience, the null values are filled with 0s or 1s depend on the circumstances. For categorical variable "Open", fill null with 1 to assume the store opened when missing. For other variables such as competition distance, fill null with 0 will simplify the question.

The data is ready to use after data preparation. It comes to build features for a model. A sales prediction clearly can be treated as a time series problem, which leads this problem to be a time series supervised learning problem. Several timestamp features extracted from exist feature "Date" such as day, month and year. Thanks to Kaggle public kernels, several time related features built by proper mathematic transformation from timestamp features, such as "CompetitionOpen". Different states supposed to have different state holidays, therefore feature "zone" comes. Feature engineering is the hardest and most important step throughout the whole process.

Step four is the parameter tuning stage, the improvement of model performance rather depending on model concept's understanding than parameter values. Through tuning parameter, a solid understanding of what parameters use for, how the parameters influent performance, and why the parameters will influent the certain part. An example of

XGBoost, the combination of number of rounds and learning rate will influent the gradient descent learning process, thus a better performance.

At last but not least, data visualization. A good model is much more useful when it can be communicated well through inspiring graphs.

Improvement

There are several directions this project to dive into: time series features building, models embedding, data munging process improvement. First, time series features building, the model can extract past sales data for future reference. Second, different models have different behaviors, such as XGBoost is not very powerful dealing with categorical variables. By embedding different models, the shortage of the final model theoretically, can improve. Third, the missing value can be filled by K-nearest-neighbors instead of 0.

References

Hastie, T. T. (2008). *The Elements of Statistical Learning.* Stanford Press.

Jean-Pierre. (2017, June). *Drogeriekette Rossmann – Historie, Zukunft und die Konkurrenz*. Retrieved from Die Wirtschaftsnews: https://en.wikipedia.org/wiki/Rossmann_(company)

Rossmann(Company). (2015, 12 7). *Rossmann store sales prediction.* Retrieved from Kaggle: https://www.kaggle.com/c/rossmann-store-sales/data

Appendix

Code please check GitHub link