Machine Learning Engineer Nanodegree Capstone Project

Botao Deng
Udacity
bdeng3@ur.rochester.edu

1. Definition

1.1. Project Overview

Sales forecasting is a key element in conducting business.[2] Prediction is made based on past sales performance and analysis of expected market conditions.[1] Good forecasting guarantees that sufficient products will be manufactured or delivered to customers on a timely basis, resulting in happier customers and fewer complaints. It also prepared the company with good management of inventory, avoiding both overstock and stock-out situations. If the company can predict demand, and manage production more efficiently, it will have better control over the supply chain. This affords the company opportunities to fully manage resources.

Time series forecasting is a known approach to solve the sales forecasting problem[8]. Different kinds of patterns such as seasonality pattern, sales trend and cyclical pattern are extracted from the past transactions. Classic statistical time-series forecasting techniques includes Exponential Smoothing[5] and Auto Regressive Integrated Moving Average (ARIMA)[4].

Recently, deep learning is becoming more and more popular at sale forecasting.[10] Recurrent neural network, which is known to be exceptionally good at processing sequential data, has already been shown to be good at handling sale forecasting problem.[12] Compared with traditional neural network, it maintains a mechanism that hold on to the memory of its previous input. During each time step, it selectively added information from current input. An decision can be made either at the end or during each time step, depending on the form of output we would like to get.

At this project, we are going to build various recurrent neural networks, based on LSTM and GRU cell, to solve the sale forecasting problem introduced by Kaggle, "Corporacin Favorita Grocery Sales Forecasting".

1.2. Problem Statement

The following description comes from the kaggle challenge home page[1].

"Brick-and-mortar grocery stores are always in a delicate dance with purchasing and sales forecasting. Predict a little over, and grocers are stuck with overstocked, perishable goods. Guess a little under, and popular items quickly sell out, leaving money on the table and customers fuming.

The problem becomes more complex as retailers add new locations with unique needs, new products, ever transitioning seasonal tastes, and unpredictable product marketing. Corporacin Favorita, a large Ecuadorian-based grocery retailer, knows this all too well. They operate hundreds of supermarkets, with over 200,000 different products on their shelves.

Corporacin Favorita has challenged the Kaggle community to build a model that more accurately forecasts product sales. They currently rely on subjective forecasting methods with very little data to back them up and very little automation to execute plans. They are excited to see how machine learning could better ensure they please customers by having just enough of the right products at the right time."

It is possible to think of this problem as a classification problem, for example by predicting one of several possible ranges of the output values. However in this final project, we will only focus on the algorithms for regression since these are the most relevant ones when it comes to time series forecasting. Specifically, we are going to use random forest regressor and various recurrent neural networks (RNN) to do the prediction.

We first employ some memory optimization strategy so that the dataset can be saved and loaded quicker without taking a large amount of memory. And we also visualize the training data and supplementary data via an interactive visualization tool, and do regular cleaning on the dataset. After manipulating the data into a time-series manner, it is fed into random forest regressor and RNN to get the prediction.

[1] https://www.kaggle.com/c/favorita-grocery-sales-forecasting

id	date	store_nbr	item_nbr	unit_sales	onpromotion
0	2013-01-01	25	103665	7.0	NaN
1	2013-01-01	25	105574	1.0	NaN
2	2013-01-01	25	105575	2.0	NaN
3	2013-01-01	25	108079	1.0	NaN
4	2013-01-01	25	108701	1.0	NaN

Table 1. Sample of training data

1.3. Metrics

The default evaluation metrics Kaggle uses in this problem is Normalized Weighted Root Mean Squared Logarithmic Error. (NWRMSLE)

$$E = \sqrt{\frac{\sum_i^n w_i \sum_{d=1}^{16} (log(y_{i,d}+1) - log(\tilde{y}_{i,d}+1))^2}{\sum_i w_i}} \tag{1}$$

Whereas n is the number of (store, item) pair, specifying the exact item being sold at a specific store. d referring to the date that the model will predict. There are 16 days in the test set, therefore d ranges from 0 to 16. The weight w_i is generated depending on whether the item is 'perishable' or not. This is a meta data information from the file 'items.csv', items marked as perishable have a score weight of 1.25; otherwise, the weight is 1.0. $\tilde{y}$ is the predicted sales of a (store, item) pair, and y is the ground truth sale of that pair.

This metrics is chosen mainly because it avoids penalizing large difference when both predicted value and ground true value are large. The target variable y represents unit sales of a product at a store, the magnitude of difference could be substantial, therefore it is suitable to use NWRMSLE here.

We also use r2 score and explained variance (EV) to quantify the predictive power of our model. They are defined as follows:

$$r^2 = 1 - \frac{\sum_i (y_i - f_i)^2}{\sum_i (y_i - \bar{y_i})^2} \tag{2}$$

$$EV = 1 - \frac{Var(y - f)}{Var(y)} \tag{3}$$

In the equations above, y is the true value of y, whereas f refers to the predicted value. $\bar{y}$ is the mean value of all y.

r2 score is also known as coefficient of determination, it ranges from 0 to 1. A model with an r2 of 0 is no better than a model that always predicts the mean of the target variable, whereas a model with an r2 of 1 perfectly predicts the target variable. Any value between 0 and 1 indicates what percentage of the target variable, using this model, can be explained by the features.

The best possible value of explained variance score is 1, which means the variance of trues value and predicted values are exactly the same. The lower the worse.

2. Analysis

2.1. Data Exploration

The data is available on the competition website, and it has been split into training and testing set. The training set, contains transactions data from 2013.01.01 to 2017.8.15. Overall, there are about one hundred and twenty five million transaction records. And the testing set has the sales records from 2017.8.15 to 2017.8.31. Our ultimate goal, is to learn from the history, and to predict the sales data of the testing set. But during training, we need to further split the training set into train, validation and test set. Because the original test set provided does not include target label, therefore it could not be used during the training phase.

The following file description comes from the Kaggle competition website.[2]:

- **train.csv:** Training data, which includes the target unit_sales by date, store_nbr, and item_nbr and a unique id to label rows. The onpromotion column tells whether that item_nbr was on promotion for a specified date and store_nbr.
- **test.csv:** Test data, with the date, store_nbr, item_nbr combinations that are to be predicted, along with the onpromotion information.
- **stores.csv:** Store metadata, including city, state, type, and cluster. Cluster is a grouping of similar stores.
- **items.csv:** Item metadata, including family, class, and perishable. Items marked as perishable have a score weight of 1.25; otherwise, the weight is 1.0.
- **transactions.csv:** The count of sales transactions for each date, store_nbr combination. Only included for the training data timeframe.

[2] https://www.kaggle.com/c/favorita-grocery-sales-forecasting/data

- **oil.csv:** Daily oil price. Includes values during both the train and test data timeframe. (Ecuador is an oil-dependent country and it's economical health is highly vulnerable to shocks in oil prices.)
- **holidays_events.csv:** Holidays and Events, with metadata.

A piece of training data sample is provided at Table 1. The column unit_sales is what our model is going to predict. The store_nbr and item_nbr together defines a unique transaction. Column onpromotion means if the specific item is on promotion that day, but lots of data in this column is missing. We will replace those missing values with zero in the data preprocessing section. Column id is unique for each transaction, and we will be using id to identify each row in the later analysis.

2.2. Exploratory Visualization

As we mentioned above, each row in the training set is a unique transaction. In this section, we will check out the distribution of transaction of selected store over years.

If we look at Fig. 1 and Fig. 2, we can see there are two types of characteristics we need to encode in our features:

- Seasonality. The most apparent seasonality pattern of store 25 is its high sales peak at around Feb. and Mar., while the sales around Jun. and Jul. are often the lowest. For the store 26, only small fluctuation exists except around Dec., we can infer that the items sold in this store has a relatively steady demand across all seasons.
- Trend. The overall sales for store 26 clearly show a downtrend, whereas the sales of store 25 over years are more steady.

More visualization of each store can be found at here[3]. Some observations are listed:

- There are several stores that shows an uptrend over years at transaction. They are store 44, 47, 45, 46, 3. Among them, 44, 45, 46, 47 are stores that located at Quito, and all of them belongs to type A. And I think it's safe to say that type A store probably sells grocery product. Store 3 located at Guayaquil. Therefore, I think large amount of profits comes from the Capital, Quito, and Guayaquial.
- Christmas has transaction spikes every year.

The takeaway message from these observation is we should definitely take into account categorical features such as location, store type during feature engineering phase.

[3] https://cdn.rawgit.com/Bato803/Sharpest-Mind/2d54a759/index.html

2.3. Algorithms and Techniques

There are mainly two quantitative approaches to tackle sales forecasting problem, the statistical time series methods and machine learning algorithms. Classic time series methods includes exponential smoothing and ARIMA (Auto Regressive Integrated Moving Average). The experiments reported in this paper exclusively use machine learning, especially deep learning approaches.

2.3.1 Random Forest

Random forest[3] is an ensemble of decision trees[11], and each tree is constructed using a random subset of data. Decision tree is supposed to predict the correct value by learning simple rules from the training data.

During training, the tree is incrementally built by breaking the data into smaller and smaller subset. It split the data in a way that maximize the information gain after splitting. In other words, after each split, the subset of data would be easier to distinguished compared with its parents.

In our case, a regression model is built for each feature and it is used to predict the output values. The error between output value and actual value is computed, and the decision node that has lowest error is chosen to be the next split point.

Random forest is composed of an arbitrary number of decision trees, and all trees are going to participate in the prediction process. Compared with other machine learning algorithms, random forest require less effort in data preprocessing, and since it is a non-parametric model, therefore has less hyper-parameters to tuned.

2.3.2 Recurrent Neural Network (RNN)

A recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed cycle. This allows it to exhibit dynamic temporal behavior. Unlike feed-forward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs.[7] This feature enables detection of long-term temporal correlations in input data and makes the RNN suitable for processing sequential inputs.

The Recurrent Neural Network we use in this project has the following form, shown at Table 2.

There is a problem with RNNs called vanishing gradient problem. This problem arises when training RNN with backpropogation through time. During training, each of the weights receive an update proportional to the gradient of error function with respect to the current weight, computed using chain rule. The gradient becomes vanishingly small if the time step in RNN is too long. This results in inability to learn long-range dependencies and limits RNNs to looking back only a few time steps. One of the possible solutions

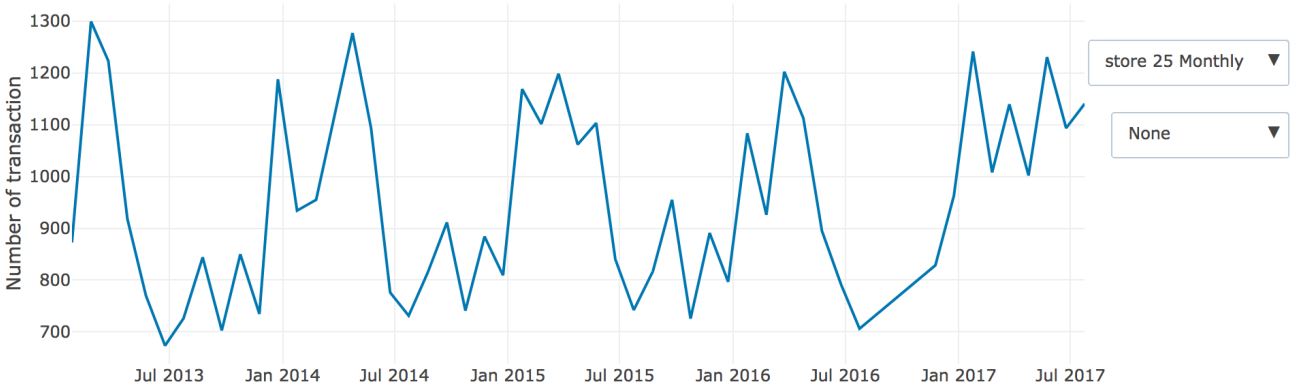

Figure 1. Monthly number of transaction of store 25

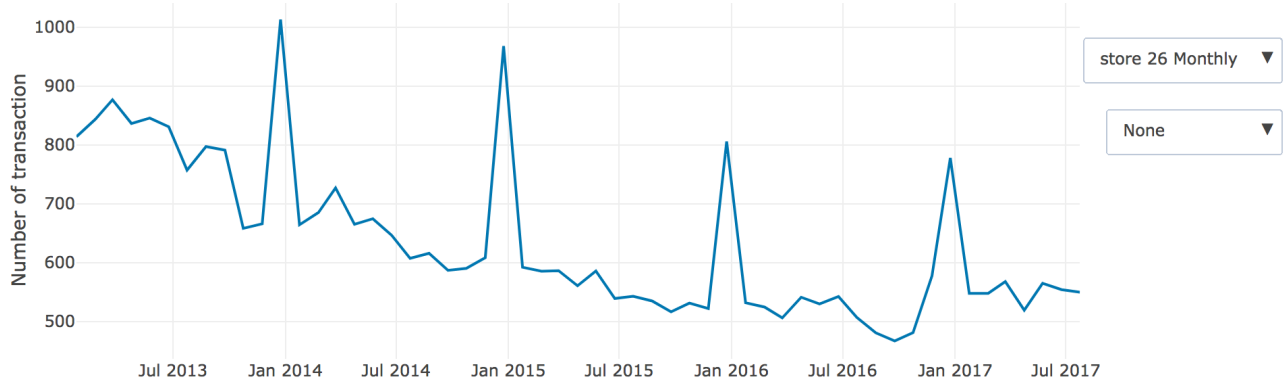

Figure 2. Monthly number of transaction of store 26

is a different neural network architecture called LSTM network that is covered in the next section. [12]

2.3.3 Long Short Term Memory (LSTM)

LSTM[9], is a special kind of recurrent neural network that is able to learn long-term dependencies. Unlike normal RNN, whose basic structure is a chain of simple module shown at Fig. 3, gating system is added into LSTM's unit. It's shown at Fig. 4.

The gating system regulate the information in LSTM cell in the following manner:

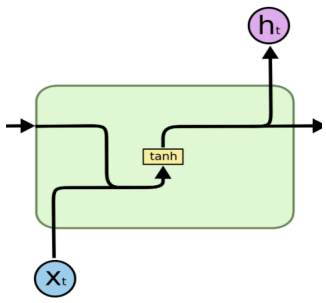

Figure 3. Basic RNN structure. Image is taken from colah's blog

- Input: the input to the current LSTM cell consists of input from current time step and the memory state from previous time step.
- Forget gate layer: $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$. This gate is used to control which part of previous memory state the model needs to discard.
- Input gate layer: $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$, $\tilde{C}_t = tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$ This gate is to control the degree to update the model, and the value candidates to update.
- Update memory: $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$
- Output gate layer: $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$, $h_t = o_t * tanh(C_t)$. This gate is to decide which values and to what degree this cell would output.

In our case, our transaction records ranges from

Layer	parameter
RNN	256 cells
BatchNormalization	-
Dropout	0.5
Dense	128
Exponential Linear Unit (ELU)	-
BatchNormalization	-
Dropout	0.5
Dense	64
Exponential Linear Unit (ELU)	-
BatchNormalization	-
Dropout	0.3
Dense	32
Exponential Linear Unit (ELU)	-
BatchNormalization	-
Dropout	0.3
Dense	16
Exponential Linear Unit (ELU)	-
BatchNormalization	-
Dropout	0.1
Dense	1

Table 2. Neural Network with RNN cell

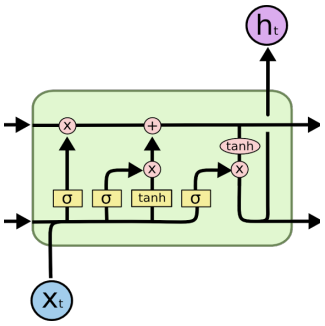

Figure 4. LSTM structure. Image is taken from colah's blog

2013.01.01 to 2017.08.15. Although we decide to use only use part of the dataset to train our model, there is still data across several months to about a year. It is important for our model to remember sales data not only weeks ago, but months ago. Because from our exploratory analysis we see that there is a strong seasonality pattern exist in our dataset. Therefore, we expect LSTM generate better result compared with simple RNN.

2.4. Gated Recurrent Unit (GRU)

GRU is an important variant of LSTM, it has a simpler model by merging forget gate and input gate into one update gate. And at the same time merge the hidden and memory state into one state.

We can see from Fig. 5 that, r_t decides how to combine

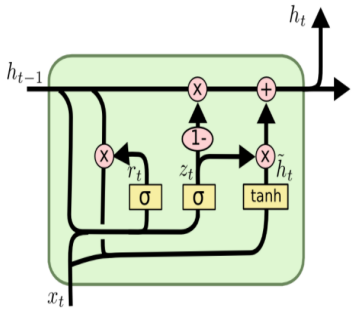

Figure 5. GRU structure. Image is taken from colah's blog

the new input with the previous memory, and z_t decides how much of previous memory can be kept around. If r_i is all one, and z_i to be all zero, then GRU becomes a plain RNN model.

The reason we try GRU here is, according to [12], GRU tends perform much better in terms of sales forecasting. Therefore let's see how well it does in this context.

2.5. Benchmark

The benchmark model comes from the public starter code[4]. There is two reasons I wanted to use this model:

First, in this model, basic feature engineering is performed on the raw dataset, it creates features such as "mean sales of previous 3/7/14/... days", "number of promotion in the last 14/60/140 days", "same weekday's sales of previous/next week'. After training for 5 epochs, it reached 0.518 Normalized Weighted Root Mean Squared Logarithmic Error, and that ranked around 697 out of 1632 teams at Jan 2. We intend to add more intuitive and complexity features in our final model.

Second, this model only has a shallow 3-layer LSTM network. We hope to explored more complex network architecture in our experiments.

3. Methodology

3.1. Data Preprocessing

3.1.1 Memory Optimization

A quick look into the training set we found two problems:

The training file has about 4.7G and a hundred twenty five million rows. Although the RAM of our AWS ec2 instance is about 60G, we still need to do some experiments on local computer or Kaggle's kernel, whose RAM are 8G and 16G respectively. Loading the dataset will take forever, and 4.7G is definitely not going to be a manageable size of local computer.

Before we feed data into our model, we need to manipulate and transform the original training set. Given that

[4] https://www.kaggle.com/senkin13/lstm-starter

we might need to do lots of parameter tuning of our deep learning model, we don't want to manipulate on the original dataset every time we run the model. Therefore, we need to think of a way that enable us to save and retrieve the transformed dataset quickly so that we don't waste too much time in data preprocessing.

In order to solve these two problems, we first analyze a subset of training data, and do the following:

- Check memory consumption of each column.
- Check the range of values in each column.
- Check if we can convert datatype and data structure for each column.

We found that each column consume equal amount of memory (about 194MB), whereas the range of each columns vary greatly. For example, the column $store\_nbr$ only ranges from 1 to 54, but its data type is int64. And it has an equal amount of memory consumption as other columns whose range of values is much larger. Therefore, we decide to do the following data type conversion:

- Column store_nbr: There are only 54 stores, but the original dataset is using int64 (2^{32} - 2^{32}) to store number. We need to convert it to uint8, who ranges from 0 to 255.
- Column Item_nbr: The value of this column ranging from about 96,995 to 1,909,755. The original data type int64 is about 2000 times larger than the maximum value of item_nbr. Therefore, int64 is too large for the values in this columns. Instead, uint32 should be a good choic(0 — 4,294,967,295)
- Column id: The values range from 125,000,000 to 128,000,000. So, again, uint32 should be a good choice for id as well.
- Column unit_sales: These values range from 1344 to 12021. Technically speaking, float16 would be enough. But just in case there exist days that might have a large amount of sales (such as Christmas), we use float32 instead.
- Column onpromotion: This column's values are boolean variable. Therefore, we need to fill in those nan first. Then we convert them into uint8, where 0 represents false and 1 represents true.

Splitting date into three columns - 'Year', 'Month', 'Day' could save lots of space. Because date column is consuming the same amount of memory as those whose data type is int64. But we decided not to do that given we would have more convenience by keeping the date column as a timestamp format.

The original training set size is 4.7G, by utilizing our strategy, it ends up at 2.1G. But we do not plan to use all the records back to 2013 to train our model, because our model's performance will be tested on the the sales record from 2017.08.15 to 2017.08.31. of 2017 by Kaggle. Therefore, we only include data starting at 2016-08-15 into our training set so that our model can catch the seasonality across the whole year, meanwhile maintaining a reasonable amount of data to train on, considering the time and memory of computation. The size of our final training set ends up at 1.1 GB

In order not to repeat the same process again and again during training and testing, we save the modified dataframe in feather format, a fast on-disk format for data frames for R and Python, powered by Apache Arrow. By utilizing this technique, loading in the dataset only takes 2.4 second, which is 300 times faster than saving it into csv file!

3.1.2 Preprocessing

There is only one numeric variable in training set - 'unit_sales', since most of its values is from 0-10, while some are ranging from 20000 to 80000. We decided to perform log transform so that they become more normalized.

For categorical variables, we use sklearn's LabelEncoder to transform them into numeric values. And all the NaN are filled with zeros. We do think of one-hot encoding at first, but we need to do a large amount of feature engineering in the next few steps, some of the operation requires merging and reindexing different dataframe. The dataframe we get after we one-hot encoded categorical features becomes three times as large, and the future operation becomes much more expensive. Therefore we decide to use LableEncoder at last.

3.2. Implementation

The workflow of our works is shown at Fig.6

3.2.1 Time series

The original training set is ordered by id, and items that are sold at the same date or by the same store are grouped together, shown at Table 1. We use pandas to manipulate the dataset so that after conversion, each row represent a store/item pair, whereas each column represents a date. By doing this, each row represents a time series of an certain item sold at a specific store. The sample after conversion is shown at Fig.7.

3.2.2 Feature Engineering

The original daily sales record is extremely noisy, Fig. 9. Given that sales data has a very strong weekly and seasonality patterns, we decided to perform feature engineering to

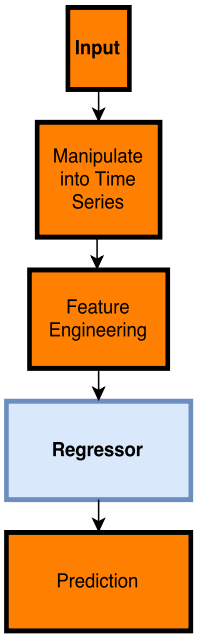

Figure 6. Workflow of implementation

our original dataset. The features are generated based on several intuition of the domain knowledge, such as "Sales of the same weekday of consecutive two weeks are related" or "If there is a promotion in the last few days, then the sale on today is more likely to increase", etc. At the end, the following features are added in the training set:

- The original unit sales of the last 16 days.
- The original onpromotion information of the last 16 days.
- The mean sale in last 3 days, a week, two weeks, a month, two months and half year.
- The number of onpromotion in the last two weeks, two months and half year.
- The mean Mon/Tue/Wed/... sales of last 4/8/16 weeks.
- min/max/median/std sales in the last 3/7/14/30/60/140 days.
- Same anaysis above for the same day a week ago.
- How much the next day have more/less sales than the previous day.
- Categorical features of store and item.

3.2.3 Random Forest

We used scikit-learn's Random Forest Regressor in this project. We first use grid search on 25% of our dataset to choose the best parameters including "n_estimators", "min_samples_split" and "max_depth.". After selecting the best parameters, we train the model on the whole dataset to predict the sales in the next 16 days.

We also try to train multiple random forests to predict sales one day at a time. And we found this approach is more accurate compared with predicting 16 day's sales all at once. And at the end, we use feature selection to select the best 300 features and train random forest again, only with the best features.

The features selection process is done using the metrics called mutual information, implemented by Scikit-learn. The function relies on nonparametric methods based on entropy estimation from k-nearest neighbors distances[6].

Our random forest implementation can be concluded in Fig. 8:

3.2.4 LSTM & GRU

The LSTM and GRU network we implemented have the same structure as shown at Table. 2

During training, we also implement early stopping in case that the validation loss stop to decrease at 10 iterations. Learning rate is originally set at 0.001, and if the validation loss stop improving for 5 iterations, we would decrease the learning rate by a factor of 0.1.

Both of these models are trained with Adam optimizer, which outperforms SGD and other adaptive learning rate optimizer in a wide range of applications.

3.3. Refinement

I made lots of parameters changes for every model, sometimes the performance get boosted for a bit and sometimes it really does not make a difference.

For random forest, I used Grid Search to try to following parameters:

- "n_estimators": [50,80]
- "min_samples_split": [10,15]
- "max_depth": [10,15]

And it turns out that a [80, 10, 15] is the best set of parameters. Notice that we are using grid search on the rows on our training set, and each row represents a time-series (after feature engineering) of a specific item sold at a store. Therefore, we are not splitting the time-series into N fold, but splitting all the time-series into N fold, each fold containing a certain amount of complete time series. The initial result without using grid search has 0.583 NWRMSLE, and

	date	2016-08-15 00:00:00	2016-08-16 00:00:00	2016-08-17 00:00:00	2016-08-18 00:00:00	2016-08-19 00:00:00	2016-08-20 00:00:00	2016-08-21 00:00:00	2016-08-22 00:00:00	2016-08-23 00:00:00	2016-08-24 00:00:00	...	2017-08-06 00:00:00	2017-08-07 00:00:00
store_nbr	item_nbr													
1	96995	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	1.098612	1.098612
	99197	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	1.098612
	103520	0.000000	0.000000	0.693147	0.693147	0.000000	0.000000	0.693147	0.000000	1.098612	0.000000	...	0.000000	0.000000
	103665	0.693147	0.000000	1.945910	0.693147	1.386294	2.197225	1.386294	2.197225	1.098612	0.000000	...	0.693147	1.098612
	105574	0.693147	1.386294	1.386294	0.000000	1.386294	0.000000	1.609438	2.564949	0.000000	1.386294	...	0.000000	1.791759

Figure 7. Sample of training set in time-series format

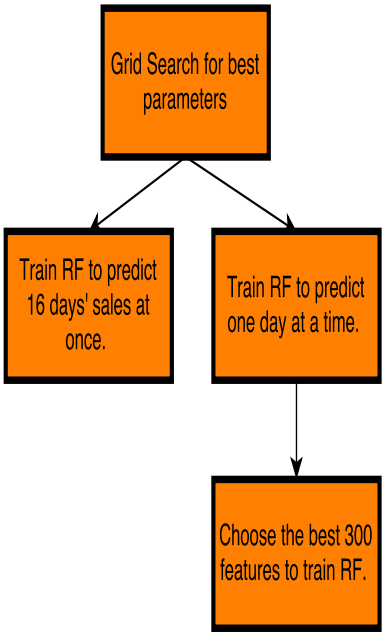

Figure 8. Workflow of Random Forest Experiment

the model after grid search has 0.570. These two results turn out to be not very impressive, but we can stay with them for now considered that it is only trained on 25% of our training set.

In the recurrent neural network approaches, most of the tuning is tweaking the parameters of the neural networks. I try to use another network architecture for LSTM, where we have a 4 layer neural network and the first LSTM layer has 128 neurons, the batch normalization and Dropout layer comes after. And the rest three layers are all fully connected layer, each with 64, 32 and 1 neurons respectively. All of them have batch normalization and Dropout layer. But the result turns out to be a little bit worse.

I also try to tune the Dropout rate in the LSTM network hoping that it would overfit the training set less. But the result is almost the same.

Two kinds of different activation functions, Parametric Rectified Linear Unit (PRelu) and Exponential Linear Unit (ELU) are tried out. The first activation allows the gradient to go through when the gradient is negative, where the control factor α is learned during training. The ELU takes the exponential of negative gradient before being multiplied by α. Both of these two choice shows similar performance.

4. Results

4.1. Model Evaluation and Validation

The model that gives the best performance is the recurrent neural network with GRU cell. This model architecture I use is a single GRU layer with 256 memory cells, 0.3 dropout rate, followed by 5 fully connected layer. Using this model I received a NWRMSLE rate of 0.511 from Kaggle. I think it is a good model to use for this problem, because during feature engineering phase, we design the features in a way that they can catch both the seasonality and weekly patterns. And the GRU cell would be able to find internal relationship between whose features.

The validation set is used to monitor the training process of our model. Whenever the validation loss stop decrease for a number of iterations, we either reduce the learning rate or stop the training to avoid overfitting.

4.2. Justification

After prediction is made, I calculated the r-2 score and explained variance for different model's predictions. And at last I need to create a CSV file and write all the probabilities on each row, where the row id is the transaction id from the original testing set. Below is the submission file I prepared for Kaggle, and it is shown in Fig.

My first attempt has a NWRMSLE of 0.556 and placed me in the 1134^{th} place out of 1632 teams on Jan 1^{st}. This

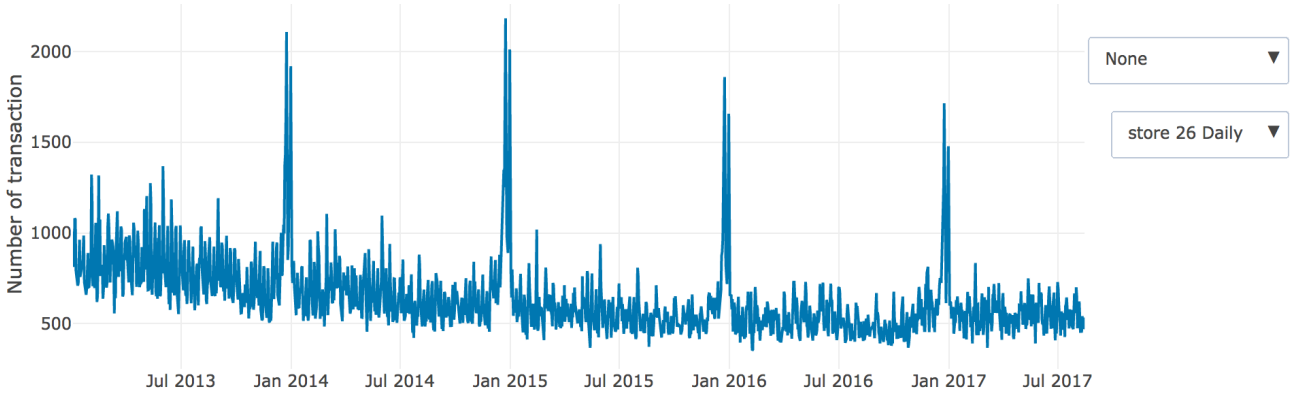

Figure 9. Daily sales of store 26

id	unit_sales
125497040	0.2269
125497041	0.2133
125497042	0
125497043	1.1101
125497044	2.0178
125497045	6.5272
125497046	11.032
125497047	0
125497048	0.8872
125497049	0.1885
125497050	1.3975
125497051	4.302
125497052	2.9591
125497053	0.6738
125497054	0.0077
125497055	1.3583
125497056	1.1092
125497057	1.7759

Figure 10. Submission file format

was the RandomForest model which predicts 16 days all at once. And My second attempt is the Random Forest Model that predicts one day at a time and this gives me 1058^{th} out of 1632, having a NWRMSLE of 0.532. To have a more thorough understanding, the r-2 score and explained variance of these two models are 0.72, 0.74 and 0.71, 0.74 respectively. The result shows consistency across all metrics. I made a series of attempts afterward, and below are the result measured by Kaggle and sklearn's metrics.

1. Random Forest (16 days at once)
    - NWRMSLE(Kaggle): 0.556
    - r2-score(sklearn): 0.72
    - explain variance(sklearn): 0.71
2. Random Forest (one day at a time)
    - NWRMSLE(Kaggle): 0.532
    - r2-score(sklearn): 0.74
    - explain variance(sklearn): 0.74
3. Random Forest with Feature Selection:
    - NWRMSLE(Kaggle): 0.536
    - r2-score(sklearn): 0.73
    - explain variance(sklearn): 0.74
4. Recurrent Neural Network with Basic Recurrent unit:
    - NWRMSLE(Kaggle): 0.516
    - r2-score(sklearn): 0.91
    - explain variance(sklearn): 0.92
5. Recurrent Neural Network with LSTM cell:
    - NWRMSLE(Kaggle): 0.515
    - r2-score(sklearn): 0.91
    - explain variance(sklearn): 0.91
6. Recurrent Neural Network with GRU cell:
    - NWRMSLE(Kaggle): 0.511
    - r2-score(sklearn): 0.93
    - explain variance(sklearn): 0.94

In our benchmark model, whose NWRMSLE is 0.518. By the end of the competition, my random forest predictor can not to a better performance than that. But after more exploration in LSTM and GRU, my model place me on 153^{rd} out of 1675 competitors on the public leaderboard. As for the private leaderboard, my model is in 147^{th}. Public leaderboard is calculated with approximately 31% of the test data. And the private leaderboard is based on the other 69%, so the final standings could be different.

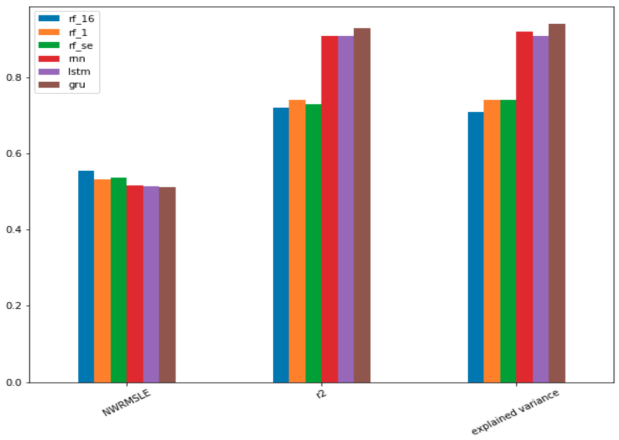

Figure 11. Results of various models.

5. Conclusion

5.1. Free-Form Visualization

We can see from Fig.11 that basically recurrent neural network methods outperform random forest by a large margin. Although some methods might outperform another in r2-score, while having lower value in explain variance, GRU tends to be the most stable model for solving problems of this kind.

5.2. Reflection

Before I get into this final project, my previous research and project experience are more focused on computer vision and image processing. I did read some literature with regard to time-series data, but I never spend a large amount of time processing a real-world dataset. This Kaggle competition really gets me into this field. I read about different technique with regard to time-series data, including Jason Brownlee Machine Learning blog, papers cited at the end of this final report and other documents from Keras and Sci-kit Learn. They are very useful. I also learned a lot about data visualization and processing with plotly and Geojson, and how to create an intuitive understanding of business data. It was also helpful to read discussions on the Kaggle's competition page, and the kernels that are provided by experienced expert in related field. Their opinions helps a lot and sometimes even tiny changes in learning rate might lead to improvement in the final result.

I do get a little disappointed at the random forest's result, considered the time we spent using grid search to find the good set of hyper parameters. Unlike deep learning model, random forest is run on CPU, therefore training a deep learning model in this project is not necessarily ended up using more time than a traditional machine learning model. But if we wanted to tweak the parameters of an RNN network heavily, things become tricky. First, there are so many parameters we could tuned, such as the number of neurons, number of layers or different type of activation functions. And it's not always clear what could work best. We have a rough guide line on how each feature would affect overfitting or underfitting, but it is difficult to quantitatively measure its effect.

5.3. Improvement

Something interesting I notice is that I use the features starting from 2017.7.26 to 2017.8.15 as our validation set, I wonder if it helps if we try a rolling window of validation set instead of simply using a fixed time range. Since the date from that fixed time range has its own patterns, and training our model toward that time range will possibly bias our model. But doing all those experiments are quite expensive, my local computer often crashes when processing large dataset like this, and using AWS all the time is going to cost us greatly.

Besides, by reading Kaggle discussion I noticed that the features we crafted play a vital role in the model's final performance. I think it might be helpful if we include more classic features into our training set, such as the exponential smoothing of weekly mean sales or moving average across the last several weeks. Actually adding the categorical features really boosted my model's performance, and maybe more feature engineering can be performed over the categorical features.

References

[1] J. S. Armstrong. Combining forecasts. In *Principles of forecasting*, pages 417–439. Springer, 2001.

[2] S. Beheshti-Kashi, H. R. Karimi, K.-D. Thoben, M. Lütjen, and M. Teucke. A survey on retail sales forecasting and prediction in fashion markets. *Systems Science & Control Engineering*, 3(1):154–161, 2015.

[3] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[4] W. contributors. Autoregressive integrated moving average — wikipedia, the free encyclopedia, 2017. [Online; accessed 31-January-2018].

[5] W. contributors. Exponential smoothing — wikipedia, the free encyclopedia, 2017. [Online; accessed 31-January-2018].

[6] W. contributors. Mutual information — wikipedia, the free encyclopedia, 2018. [Online; accessed 31-January-2018].

[7] W. contributors. Recurrent neural network — wikipedia, the free encyclopedia, 2018. [Online; accessed 30-January-2018].

[8] M. Gahirwal. Inter time series sales forecasting. *arXiv preprint arXiv:1303.0117*, 2013.

[9] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[10] Y. Kaneko and K. Yada. A deep learning approach for the prediction of retail store sales. In *Data Mining Workshops (ICDMW), 2016 IEEE 16th International Conference on*, pages 531–537. IEEE, 2016.

[11] J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[12] N. Tyrpáková. Deep neural networks for sales forecasting.