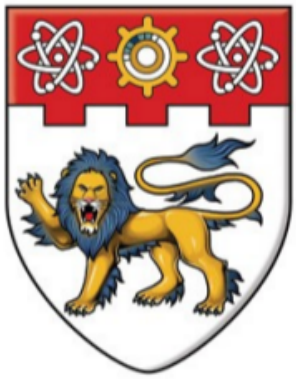

NANYANG TECHNOLOGICAL UNIVERSITY

CZ4041 – MACHINE LEARNING

PETER
ZILLION GOVIN
STEFAN ARTAPUTRA INDRIAWAN

SCHOOL OF COMPUTER ENGINEERING
2016/17

Contents

Contents ii

List of Tables iii

List of Figures iv

1 Introduction 1
- 1.1 Team Members . . . 1
- 1.2 Problem Statement . . . 1
  - 1.2.1 Challenge . . . 1
  - 1.2.2 Data . . . 1

2 Preprocessing 2

3 Experiments 4
- 3.1 LSTM . . . 4
- 3.2 Neural Network . . . 6
- 3.3 Lasso Regression . . . 7
  - 3.3.1 Background . . . 7
  - 3.3.2 Implementation . . . 7
  - 3.3.3 Preprocessing . . . 7
  - 3.3.4 Model Training & Testing . . . 7
- 3.4 Ridge Regression . . . 9
  - 3.4.1 Background . . . 9
  - 3.4.2 Implementation . . . 9
  - 3.4.3 Preprocessing . . . 9
  - 3.4.4 Model Training & Testing . . . 9
- 3.5 XGBoost . . . 12
- 3.6 Random Forest Regressor . . . 13
  - 3.6.1 Background . . . 13

3.6.2 Implementation 13
3.6.3 Preprocessing 13
3.6.4 Model Training & Testing 13

4 Results **15**

5 Conclusion **16**

List of Tables

3.1 LSTM per store result 5
3.2 Neural network with embedding layer's result 6
3.3 Normalized Lasso Result 8
3.4 Lasso Result per Store 8
3.5 5-Fold Validation Lasso Result 8
3.6 Normalized Ridge Result 10
3.7 Store Normalized Ridge Result 10
3.8 Ridge Result per Store 10
3.9 3-Fold Validation Ridge Result 10
3.10 5-Fold Validation Ridge Result 11
3.11 10-Fold Validation Ridge Result 11
3.12 Ridge Result with 26 Features 11
3.13 Ridge Result with 32 Features 11
3.14 XGBoost result 13
3.15 Normalized Random Forest Result 14
3.16 Random Forest Result with Logarithm Plus One 14

4.1 Model Results 15

List of Figures

3.1 An LSTM unit 4

1. Introduction

CZ4041 - Machine Learning project aims to expose students to a real life application of machine learning techniques. The main scope of this project is to explore, analyze, and discuss the machine learning methods used by our team in Kaggle Data Science Competition - Rossmann Store Sales.

1.1 Team Members

The team consists of four people:

- Benedita Tanabi
- Peter
- Stefan Artaputra Indriawan
- Zillion Govin

Each group member is in charge of exploring various machine learning techniques to be used in the competition.

1.2 Problem Statement

1.2.1 Challenge

The challenge for Rossmann Store Sales competition is to forecast 6 weeks of daily sales of 1,115 Rossmann stores located across Germany based on the historical data of the stores.

1.2.2 Data

To assist with the challenge, Rossmann has also provided 4 types of comma-separated-value (.csv) files, such as *train.csv, test.csv, store.csv, sample_submission.csv.* However, only *train.csv, test.csv, store.csv* are used in this project.

2. Preprocessing

The problem contains three files, *train.csv*, *test.csv*, and *store.csv*. The file *train.csv* contains historical data including sales for 1115 stores everyday from 1 Jan 2013 to 31 July 2015. While *store.csv* contains supplemental information about each store. Then, *test.csv* contains historical data excluding sales and number of customers everyday from 1 Aug 2015 to 17 Sept 2015.

Each row in *train.csv* contains store ID, day of week, date, number of customers, sales, whether the store is open, whether the store is doing a promo, state holiday, and whether that day is a school holiday. Columns in *test.csv* is almost identical to those of *train.csv*, except that sales and number of customers are unknown in *test.csv*. Each row in *test.csv* contains a submission ID for the purpose of evaluation on prediction result. On the other hand, each row in *store.csv* contains the details of a store, such as store type, assortment type, whether the store has competition, since when the competition exists, competition distance, whether the store is doing *promo2*, and *promo2* period.

Initially, our preprocessing method merges *train.csv* and *store.csv* by store ID. The columns *DayOfWeek* and *StateHoliday* in *train.csv* are transformed into one-hot vector respectively. The columns *StoreType* and *Assortment* in *store.csv* are transformed into one-hot vector respectively as well. The columns *CompetitionOpenSinceMonth* and *CompetitionOpenSinceYear* are substituted into a single column *HasCompetition* which depends on its respective *Date* column. The same thing also applies to the columns *Promo2SinceWeek*, *Promo2SinceYear*, and *PromoInterval*, they are substituted into a single column *IsDoingPromo2* which depends in its respective *Date* column. *CompetitionDistance* is set to the maximum value in the training set if a store does not have a competition in any given date. All store data are retained even when a store is closed on a particular date.

After *train.csv* and *store.csv* has been merged, we proceed to merge *test.csv* and

store.csv by store ID as well. The preprocessing method in this step is identical in the previous paragraph. The difference is that sales and the number of customers are unknown in the test dataset.

3. Experiments

Lorem ipsum dolor sit amet.

3.1 LSTM

LSTM (abbreviation for Long-short Term Memory) is a variant of recurrent neural network. LSTM seeks to solve vanishing gradient and exploding gradient problem in vanilla recurrent neural network. These problems happen when a training sequence is too long. LSTM introduces gating concept in recurrent neural network. An LSTM contains three gates, an input gate, an output gate, and a forget gate. A forget gate determines when to reset the content of the cell memory. An input and output gate control the flows of input and output of the LSTM respectively.

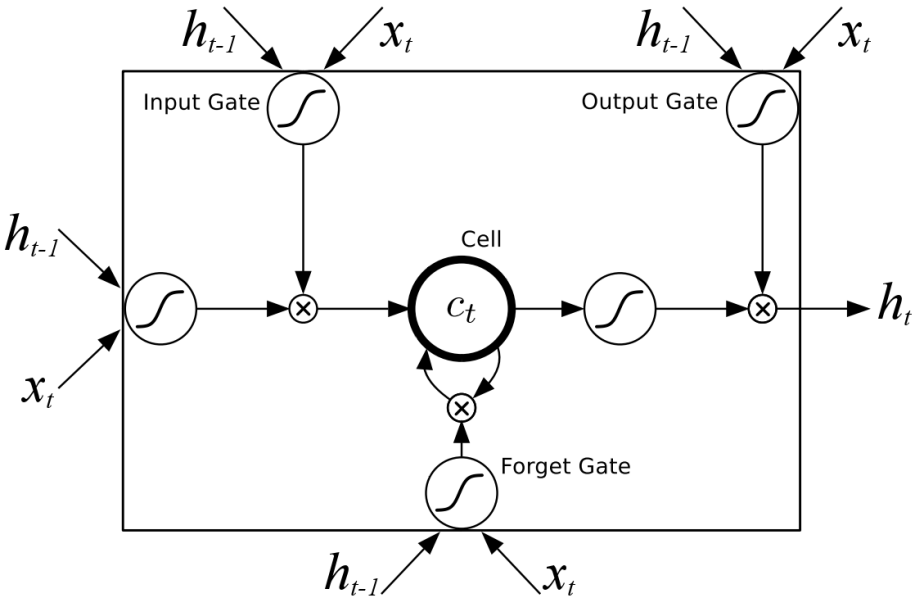

Figure 3.1: An LSTM unit

We use LSTM unit provided by Caffe[1] framework. It requires CUDA 8 and

[1]Caffe is developed by Berkeley Vision and Learning Center (BVLC)

cuDNN v5.1 from Nvidia, which enable Caffe framework to utilize Nvidia GPU to train a neural network. We used Nvidia GTX 850m to train our LSTM model. This GPU is pretty decent to train a medium-sized network.

The main idea of this approach is to train each store separately. Training an LSTM model for the entire training set is infeasible as the network model required would be large and our GPU is not powerful enough for it. Since there are 1115 stores, we are going to train 1115 small LSTM models for each store.

There are few preprocessing steps that have to be taken before training an LSTM model. First, we need to split the training dataset by store ID since we are training one model for one store. Then, we need to drop *Customers* column as this column is not available in the test dataset. Next, we have to drop *StoreType*, *Assortment*, *HasCompetition*, and *CompetitionDistance* as we are going to train an LSTM model for each store. Finally, normalize the date to $[0, 1]$, where 0.0 indicates 1 Jan 2013, and 1.0 indicates 17 Sept 2015.

The network architecture consists of, in this order, an input layer with 15 features, an LSTM unit, a fully-connected layer, and an output layer. The LSTM unit contains 50 hidden units, and the fully-connected layer contains 50 neurons. The training algorithm used is RMSProp with decaying learning rate. The learning rate update equation is as follows.

$$\alpha_{t+1} = \frac{\alpha_t}{1 + \gamma t}, \text{ where } \gamma \text{ is the decay rate.} \tag{3.1}$$

Base learning rate α_0 is 0.001, learning rate decay γ is 10^{-6}, and RMS decay is 0.9. The LSTM model is trained for 200 epochs, and the last 50 days of training data of each store are used as validation. At the end of the training, we have 1115 LSTM models. The training duration is approximately four hours.

The performance result by this model according to Kaggle, is shown in the table below.

Table 3.1: LSTM per store result

Private score	0.15578
Public score	0.15904

3.2 Neural Network

Our neural network model requires Caffe framework as well, along with CUDA 8 and cuDNN v5.1. This model is also trained on the same Nvidia GTX 850m as before.

There are few preprocessing steps that are specific to this approach. First, we add four extra features, maximum sales, average sales, maximum customers, and average customers. Note that maximum sales are only specific to each store, i.e., maximum sales is not a global max, instead it is maximum sales of a store in the training dataset. The same thing applies for average sales, maximum customers, and average customers. Then, we drop *Customers* column as it is not available in the training dataset. Finally, normalize non-boolean features, such as maximum sales, average sales, maximum customers, average customers, and competition distance.

Our neural network model consists of, in this particular order, an input layer, an embedding layer, two fully-connected layers, and an output layer. The first fully connected layer has 128 neurons, and the second fully connected layer has 64 neurons. Not all features in the input layer are embedded. Features that are embedded include *StoreID*, *DayOfWeek*, *Day*, *Month*, *Year*, *StateHoliday*, *StoreType*, and *Assortment*.

The training algorithm used is Nesterov's accelerated gradient descent with decaying learning rate. The learning rate decay equation is identical to equation 3.1. The momentum is set to 0.9, and base learning rate is 0.01. The model is trained for 35 epochs with 10000 random training samples set aside for validation.

The performance result by this model according to Kaggle, is shown in the table below.

Table 3.2: Neural network with embedding layer's result

Private score	0.20356
Public score	0.20298

3.3 Lasso Regression

Besides neural network, we explored linear regression methods to solve the given challenge as well.

3.3.1 Background

Lasso regression is an extension of Ordinary Least Square (OLS) regression. The most notable difference between Lasso and OLS is the existence of L1 regularization term in Lasso.

$$\lambda \sum_{i=0}^{N} |w_i| \tag{3.2}$$

The L1 regularization term is defined as the summation of the absolute of the weight for each features, multiplied by the penalty weight of λ (default value 0.1). The L1 regularization term is included to the error term in OLS regression. The objective of defining a regularization term is to constraint the value of w in order to avoid overfitting the model.

3.3.2 Implementation

The implementation of Lasso regression model is done using Scikit Learn Library in Python, along with supporting libraries such as Pandas and Numpy.

3.3.3 Preprocessing

The data used in this experiment is derived from preprocessed data in chapter 2 of this report, where *StateHoliday, Storetype, Assortment* are all converted into one hot encoding.
Additionally, *CompetitionDistance* feature is normalized using min-max normalization method. *Customers* data is dropped from training set as it does not exist in the testing data. Furthermore, *Date* feature is also dropped from training data as we want to ensure that all features are in the form of integer, boolean, or float.

3.3.4 Model Training & Testing

During the training and testing phase, the Lasso model was trained with different training methods, and the difference was observed based on the model's performance in the competition leaderboard. In addition, prediction is only done for

open stores (i.e. *Open* equals to true). For closed stores, the forecasted sales are defaulted to the value of 0.

Normalized Train Test

In this method, the *Sales* column is normalized (using global mean and standard deviation of *Sales*) during training and the testing result is denormalized. By using 20 features for training and testing, the performance of this training type, measured with RMSPE by Kaggle, is shown in table 3.3

Table 3.3: Normalized Lasso Result

Private score	0.24082
Public score	0.22302

Train Test per Store

In this experiment, the *Sales* data is normalized with regards to the mean and standard deviation of each store instead of using global mean and standard deviation. Furthermore, a unique predictive model is defined for each *StoreId*.
The result of this model can be observed in table 3.4.

Table 3.4: Lasso Result per Store

Private score	0.23920
Public score	0.23479

K-Fold Cross Validation

In addition to training and testing per store, this step performs a 5-fold validation on each predictive model during the training phase. Using the model with the best validation score, the model is then used to predict the sales of the respective store.

Table 3.5: 5-Fold Validation Lasso Result

Private score	0.25260
Public score	0.25426

Variations of Penalty Weight (λ)

The experiments above are repeated using different λ values, such as 10^{-15}, 10^{-10}, 10^{-8}, 10^{-4}, 10^{-3}, 0.01, 1, 5, 10, 20. However, all variation of λ produced similar or worse result compared with the default λ value of 0.1.

3.4 Ridge Regression

3.4.1 Background

Similarly with Lasso, Ridge regression is an extension of OLS with regularization term. The difference between Lasso and Ridge is the L-norm regularization. Instead of L1 norm, Ridge uses the L2 norm for its regularization term.

$$\lambda \sum_{i=0}^{N} (w_i)^2 \tag{3.3}$$

The L2 regularization term is defined as the summation of the square of the weight for each features, multiplied by the penalty weight of λ (default value 0.1). The L2 regularization term is included to the error term in OLS regression.

3.4.2 Implementation

The implementation of Ridge regression model is done using Scikit Learn Library in Python, along with supporting libraries such as Pandas and Numpy.

3.4.3 Preprocessing

The data used in this experiment is derived from preprocessed data in chapter 2 of this report, where *StateHoliday, Storetype, Assortment* are all converted into one hot encoding.
Additionally, *CompetitionDistance* feature is normalized using min-max normalization method. *Customers* data is dropped from training set as it does not exist in the testing data. Furthermore, *Date* feature is also dropped from training data as we want to ensure that all features are in the form of integer, boolean, or float.

3.4.4 Model Training & Testing

Sales forecast is only done for open stores (i.e. *Open* equals to true). For closed stores, the predicted sales value will be set to 0.

Normalized Train Test

In this method, *Sales* data is normalized using global mean and standard deviation. The resulting predictions are denormalized using the same mean and standard deviation. By using a total of 20 features for training and testing, the result from Kaggle is summarized in table 3.6.

Table 3.6: Normalized Ridge Result

Private score	0.21149
Public score	0.20215

Following the experiment, further normalization is done by normalizing *Sales* with regards to the mean and standard deviation of each store.

Table 3.7: Store Normalized Ridge Result

Private score	0.20079
Public score	0.18565

Train Test per Store

Using the store-normalized *Sales* data, we defined a specific Ridge model for each *StoreId*. The result of this method can be observed in table 3.8.

Table 3.8: Ridge Result per Store

Private score	0.17369
Public score	0.15602

K-Fold Cross Validation

To further improve the performance of Ridge regression, K-fold cross validation is used in the training phase.
Using a 3-fold cross validation method, the model with best validation score is selected to predict the *Sales* for each *StoreId*. The result can be observed in table 3.9.

Table 3.9: 3-Fold Validation Ridge Result

Private score	0.17637
Public score	0.15988

Increasing the number of fold to 5, the improvement of the prediction result can be seen in table 3.10.

Table 3.10: 5-Fold Validation Ridge Result

Private score	0.17318
Public score	0.15527

Using 10 fold validation, the resulting score is shown in table 3.11.

Table 3.11: 10-Fold Validation Ridge Result

Private score	0.17387
Public score	0.15628

Extra Feature

This method performs a 5-fold validation on the previous dataset with additional features.
Using a one hot encoding for *DayOfWeek*, a total of 26 features are used for the predictive model.

Table 3.12: Ridge Result with 26 Features

Private score	0.16308
Public score	0.14370

The historical data for *Customers* and *Sales* is used to derive additional features such as *Customers_Max, Customers_Min, Customers_Avg, Sales_Max, Sales_Min, Sales_Avg*, where the additional features are relative to *StoreId*. The result of using 32 features can be observed in table 3.13.

Table 3.13: Ridge Result with 32 Features

Private score	0.16309
Public score	0.14367

Variations of Penalty Weight (λ)

The experiments above are repeated using different λ values, such as 10^{-15}, 10^{-10}, 10^{-8}, 10^{-4}, 10^{-3}, 0.01, 1, 5, 10, 20. However, all variation of λ produced similar or worse result compared with the default λ value of 0.1.

3.5 XGBoost

XGBoost[2] is a short name for Extreme Gradient Boosting, which is a library designed and optimized for the gradient boosting algorithms. It is an extension for classic gradient boosting algorithm. XGBoost utilizes multi-threads and regularization, making it able to do more computational power and get more accurate prediction. XGBoost is used for supervised learning problems. Since XGBoost uses tree ensembles as its model, XGBoost is pretty similar to random forests. The difference is on how they train their data. Python interface for XGBoost is used to do the Rossmann store sales.

The general idea for this approach is just to do a comparison between our selected machine learning models and XGBoost. That is, how would general machine learning models, such as Lasso, Ridge, LSTM, and Random Forests fare against a boosting model, which in this case is XGBoost. There is no intention on using XGBoost as our final results.

There are additional preprossessing steps for the training data before it is trained on XGBoost model. In the beginning, *Stores* rows that are closed, and *Stores* rows that are open, but not generating any sales, are excluded.Next, *Date* column is splitted into *Day*, *Week*, *Month*, *Year*, and *DayOfYear* columns. Any N/A data from those columns will be replaced by 0. Then,the *CompetitionOpenSinceYear* and *CompetitionOpenSinceMonth* columns are converted into integer instead of boolean. Similarly for *Promo2SinceYear* and *Promo2SinceWeek*, they are converted into float. Finally, any NAN data is converted into -1. In addition, there are several new features implemented, such as *store data sales per customer per day*, *store data customers per day*, and *store data sales per customer per day.*

For validation,the training data is splitted with 7:3 ratio. Linear regression is chosen as the objective for xgboost model training. For the other parameters applied, it is using xgtree as the booster option. Base eta (learning rate for xgboost model) is 0.1. The maximum depth of the tree generated is 10. Subsamples for each tree is 0.85 , and the fraction of columns to be randomly samples for each tree is 0.4. Minimum sum of weight required in a child is 6. Number of rounds for training the data is 1200, and the evaluation metrics used is RMSPE. Early stopping value is 150. The training duration is approximately an hour.

[2]XGBoost is developed by DMLC (Distributed Machine Learning Community)

Table 3.14 shows the performance of this xgboost model according to Kaggle evaluation.

Table 3.14: XGBoost result

Private score	0.11677
Public score	0.10437

3.6 Random Forest Regressor

The last algorithm that we explored was Random Forest Regressor.

3.6.1 Background

Random Forest Regressor is a machine learning algorithm that constructs a number of decision trees and generates the mean prediction of each tree. The result of each tree will then be averaged over the number of trees. This averaged result is the final prediction of the regressor.

3.6.2 Implementation

Random Forest Regressor was implemented by using Python scikit-learn library along with other libraries such as numpy and pandas.

3.6.3 Preprocessing

The data used in this experiment is derived from preprocessed data in chapter 2 of this report, where *StateHoliday, Storetype, Assortment, DaysOfWeek* are all converted into one hot encoding.
We experimented on two different kind of normalization. The first one is by normalizing all features using the standard deviation and mean of each store. The second one is by converting *Sales* column to its logarithmic value.

3.6.4 Model Training & Testing

Each store was trained and tested independently from each other. We split the training data into training data and validation data with ratio of 50:50 and 80:20 to see the difference in their performance.

Normalized Train Test with Standard Deviation

In this method, we normalized all features using the standard deviation and mean of its corresponding store. All preprocessed features, which were 26 in total, were used. The results are summarized in table 3.15.

Table 3.15: Normalized Random Forest Result

Train:Validation	**80:20**	**50:50**
Private score	0.15812	0.15701
Public score	0.14692	0.14632

Converted Sales to Its Logarithmic Value

In this experiment, we use logarithm formula to reduce the sensitivity of *Sales* to change. Plus one was added so that the logarithmic value of *Sales* would be 0 when the real value was 0.

$$Sales = \ln(Sales + 1) \tag{3.4}$$

The equation above is implemented using numpy *np.log1p* function. New features were also added which were *Month* and *Year*.The result of this model can be observed in table and 3.16.

Table 3.16: Random Forest Result with Logarithm Plus One

Train:Validation	**80:20**	**50:50**
Private score	0.14692	0.14849
Public score	0.13378	0.13514

4. Results

In this chapter, we will explain more about the results of all of the models that are used to train Rossmann store sales data.

First, LSTM per store model yields 0.15578 for private score, and 0.15904 for public score. Next, Neural network model produces 0.20356 for private score, and 0.20298 for public score. Then, Lasso model gives the best RMSPE result when it is trained per store, instead globally trained. It yield 0.23920 as private score, and 0.23479 as public score. Ridge produces the best result when using 5-fold cross validation with 32 features. It yield 0.16309 for private score and 0.14367 for public score. Random forests has the best RMSPE result with 5:5 ratio for training and validation data. It produces 0.14692 for private score, and 0.13378 for public score. As a comparison, XGBoost model gives 0.11677 as private score, and 0.10437 for public score. Full result scores can be seen in table 4.1.

Table 4.1: Model Results

Model	Private score	Public score
LSTM	0.15578	0.15904
NN	0.20356	0.23479
Lasso	0.23920	0.15904
Ridge	0.16309	0.14367
Random forest	0.14692	0.13378
XGboost	0.11677	0.10437

5. Conclusion

Lorem ipsum dolor sit amet.