CSCI567 - Predict Future Sales

Shreyash Annapureddy
Affiliation
University of Southern California
Los Angeles, CA, 90007
sannapur@usc.edu

Kushal Sharma
Affiliation
University of Southern California
Los Angeles, CA, 90007
kushals@usc.edu

Shayan Sheikhakbari
Affiliation
University of Southern California
Los Angeles, CA, 90007
sheikhak@usc.edu

Abstract

This project was part of the CSCI 567 Machine Learning course conducted at USC in Spring 2020. The objective of this Kaggle project was to analyze and study a time-series dataset consisting of daily sales starting from 2013 Jan to 2015 Oct of a large Russian online retail platform. The challenge was to predict the total sales for every product and store for Nov 2015. The daily historical sales data consists of a list of shops, items, item prices~~,~~ and item count per day. It is important to note that not all shops are selling items every day of every month. Our group tried multiple machine learning models such as Linear regression, Random Forest and Support Vector Machine but majority of our, and this paper's, focus is on XGBoost and LSTM. The best results our group obtained was using XGBoost with a root mean square error (RMSE) of approximately 0.91.

1 Introduction

The goal of this project is to study the sales data from Jan 2013 to Oct 2015 and forecast the total amount of products sold in every shop for the test set for Nov 2015.

The general approach our group followed to solve this problem was:

1. Analyze data for outliers
2. Study data for patterns
3. construct basic features based on initial analysis
4. Random Forest model for feature analysis and importance
5. Construct new features
6. Test Different Models
7. Repeat steps 5 and 6 based on results of 6

2 Previous Work

One of the most important and possibly hardest part~~s~~ of developing any machine learning model is determining what features to use. The features heavily dictate the performance of the final model. The question then becomes, how do we select good features? While researching this question, our group came across an article written by Eryk Lewinson called [1] ***Explaining Feature Importance by example of Random Forest***

The process involves selecting/creating features based on an educated guess and then training a random forest on these features. After training, scikit learn has the option to print out the feature importance of each feature. This gives an indication of which features describe the data the best, after which we exploit what aspects of the feature made it so important so that we can generate more potentially useful features.

LSTM:
The dataset contains categorical features, these work in favor of decision tree-based methods such as Random Forest or XGBoost but proved to be a challenge for LSTM. The challenge was overcome by reading and studying the paper [2] ***Entity Embeddings of Categorical Variables,*** which outlines a method of handling categorical features. Furthermore our group found an implementation of this algorithm applied to another Kaggle competition [3] ***Entity-embedding-rossmann.***
XGBoost:
Our group learnt about XGBoost algorithm and its potential to analyze various types of data after reading ***XGBoost Algorithm: Long May She Rein!*** article [4] by Vishal Morde. ***Predict Future Sales Top 11 Solution*** Kaggle notebook [5] inspired our group to try different features to reduce the RMSE of the model.

3 Data

The first stage of the data preprocessing was observing if any of the data contained any outliers. When our group studied the data with respect to the number of sales made for each day as shown in fig 1, it was evident that two data points were outliers.

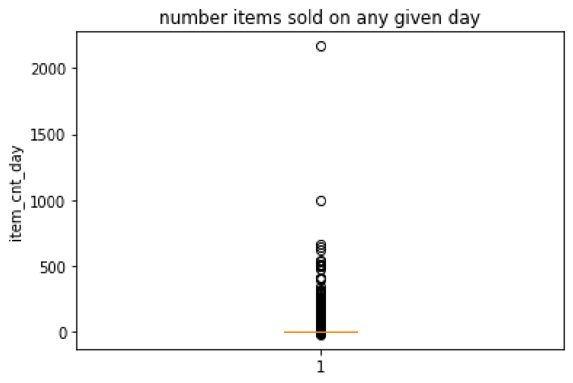

Fig.1 box plot of sales of item on daily basis.

Furthermore, the data with respect to item prices also highlighted some outliers, from very high item price and some items with negative prices. This is highlighted in fig 2.

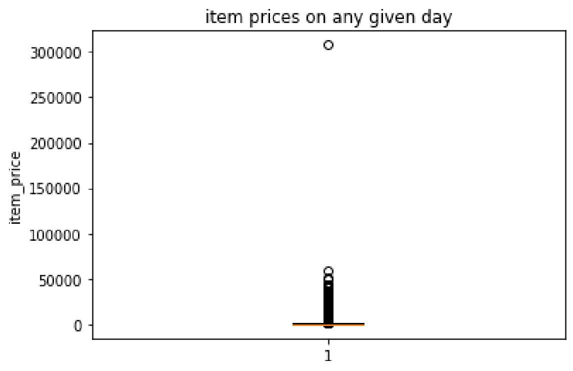

Fig.2 box plot of price of every item on daily basis.

Upon investigating the *shops.csv* data which contains the label encoding of the shops ('shop_id') it was discovered that certain shops have names that are very similar to each other. Thus, they were replaced in the following manner: 0→57, 1→58, and 10→11.

XGBoost:
At the beginning, feature which had the most intuitive effect on the targets were chosen. Then, after considering the feature importance and the correlation of the features to the targets, we could decide which other additional features needed to be added. Some features were extracted from the store and item category names such as: subtype code from item category name and city code from the shop name. After much feature engineering fig 3 shows the final list of features that were used. Their F-score (importance) and feature correlations can be seen in fig 3 and fig 4 respectively. Full expansion of features names is given in Table 1.

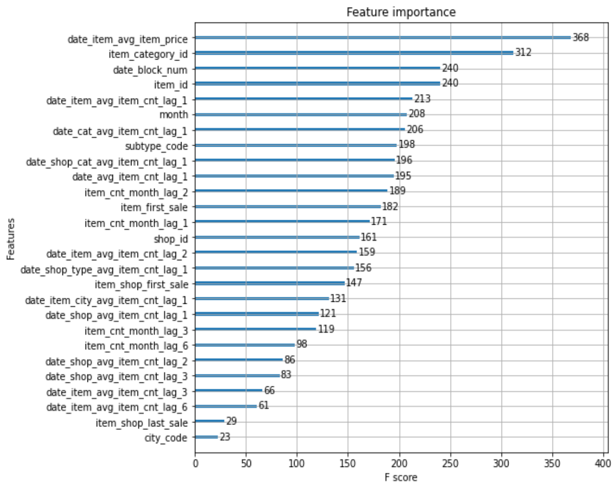

Fig.3 Feature importance, XGBoost.

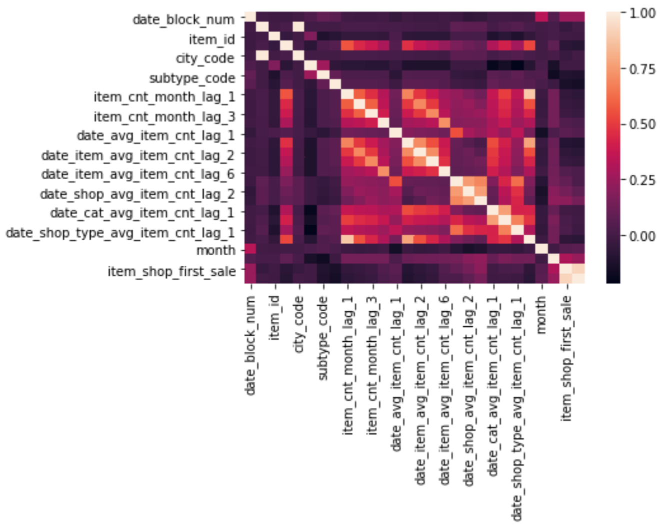

Fig.4 Correlation matrix graph

Feature	Importance
Avg item price/month (date block num/item id)	Date_item_avg_item_prce
Item category id	Item_category_id
Date block num	Date_block_num
Item id	Item_id
Avg Num of item sold/month (date block num/item id)	Date_item_avg_item_cnt
Month	Month
Avg Num of item sold/month (date block num/item cat)	Date_cat_avg_item_cnt

Subtype code	Subtype_code
Avg Num of item sold/month (date block num/shop id/item)	Date_shop_cat_avg_item_cnt
Avg Num of item sold/month (date block num)	Date_avg_item cnt
Avg Num of item sold/month of each yr. (date block num)	Item_cnt_month
Month since the first sale (date block num/item id)	Item_first_sale
Avg Num of item sold/month of each yr. (date block num)	Item_cnt_month
Shop id	Shop_id
Avg Num of item sold/month (date block num/shop_id/type_code)	Date_shop_type_avg_item_cnt
Month since the first sale (date block num/shop id/item id)	Item_shop_first_sale
Avg Num of item sold/month (date block num/item_id/city code)	Date_item_city_avg_item_cnt
Avg Num of item sold/month (date block num/shop_id)	Date_shop_avg item_cnt
Month since the last sale (date block num/shop id/item id)	Item_shop_last_sale
City_code	City_code

Table.1 Details on XGBoost features with their name.

LSTM:
Deep Entity Embedding

Deep Entity Embedding is a technique to vectorize categorical variables using learning methods. Traditional methods such as one-hot-encoding are used for quick and easy method to vectorize categorical features. The method works well for categories with low cardinality, but categories with large cardinality suffer from an explosion of dimensions.

Entity embedding not only vectorizes the categorical variables into a smaller dimension compared to one-hot-encoding, it also clusters similar categories together. The size of the embedding vectors for each categorical feature now becomes another hyper-parameter to tune. A general rule of thumb used is $\min\left(50, \frac{n+1}{2}\right)$ [6] ***Deep Learning for Tabular Data,*** where **n** is the number of categories for a given categorical feature. Unfortunately, this rule could not be implemented as it increased the feature space such that the size of the data set became infeasibly large to train on local machines such as laptops. Thus, the following embeddings sizes shown in table 2 were chosen for each of the categorial variables. Fig.5 shows the architecture that was used to determine the embedding weights.

Categorical Variable	Embedding Size
shop_id	3
item_category_id	3
year	2
month	4

Table.2 Embedding size of different categorical variables

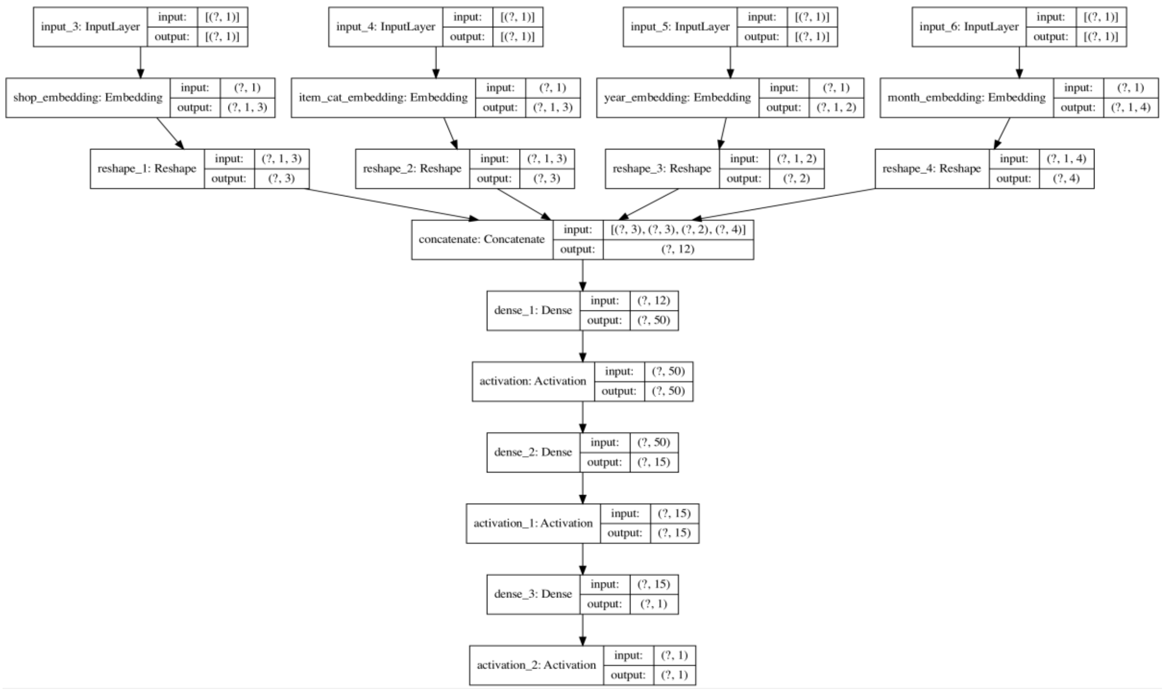

Fig.5 Embedding Model Architecture

The PCA analysis of the embeddings outlines the clustering of the categories. The clustering aids in better understanding of how the categorical features are related and thus help in further feature engineering. (Unfortunately, due to time constraints, feature engineering was not explored further for LSTM).
For instance, the year embeddings as shown in fig 6 shows that the sales in any particular year are not related to each other, thus Year as a feature may not provide any additional performance gain.

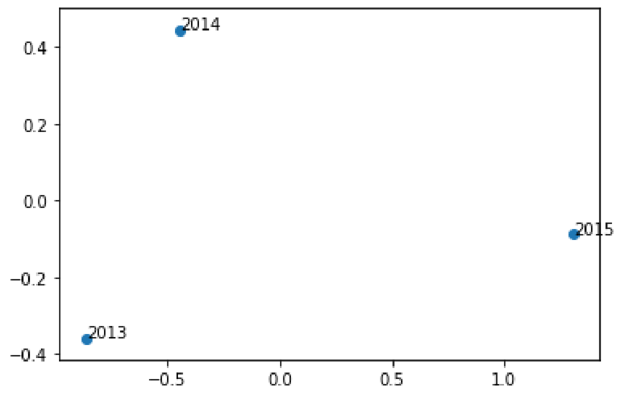

Fig.6 PCA Year embeddings

The month embeddings in Fig 7. shows that certain months are closely related to each-other e.g. Oct, Sept, Nov. This suggested that when training the model to predict sales for Nov, only training on Oct and Sep sales would be enough.

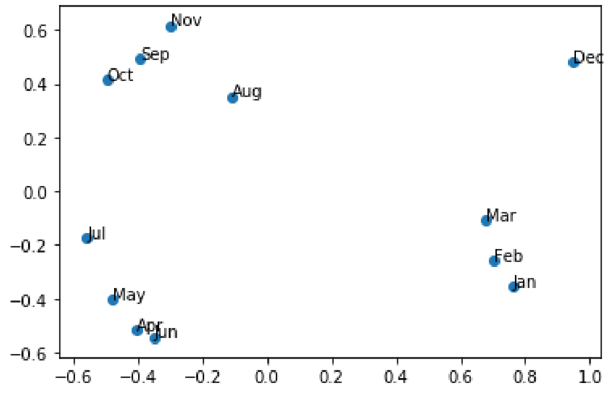

Fig.7 PCA Month embeddings

Mean Entity Embedding
Deep entity embedding method was not used on item_id due to the large number of categories, a simpler technique called mean embedding was used. An explanation of its implementation can be found in the following reference. [7] ***Encoding,***

4 Experiments

4.1 Models

XGBoost:
XGBoost produces a prediction model in the form of an ensemble of weak prediction decision trees and builds the model in a stage-wise fashion. It generalizes them by allowing optimization of an arbitrary differentiable loss function.
In order to use XGBoost, XGBRegressor library was imported. To further improve the results some hyperparameter tuning was conducted using RandomizedSearchCV, below are the final tuned parameters used for the XGBoost model:

XGBoost Model Parameter	Value
Max_depth	8
Min_Child_Weight	300
ColSample_byTree	0.8
Subsample	0.8
Eta	0.3

Table.3 XGBoost parameters with values

LSTM:
LSTM is designed to learn and store long term dependencies within the data. Given the fact that the Kaggle challenge is to forecast the sales it was only natural to attempt the challenge using an LSTM model. Going through the Keras documentation and reading the following article [8] ***LSTM Optimizer Choice,*** the Adam optimizer was selected due to its speed in convergence and its adaptive learning rate.

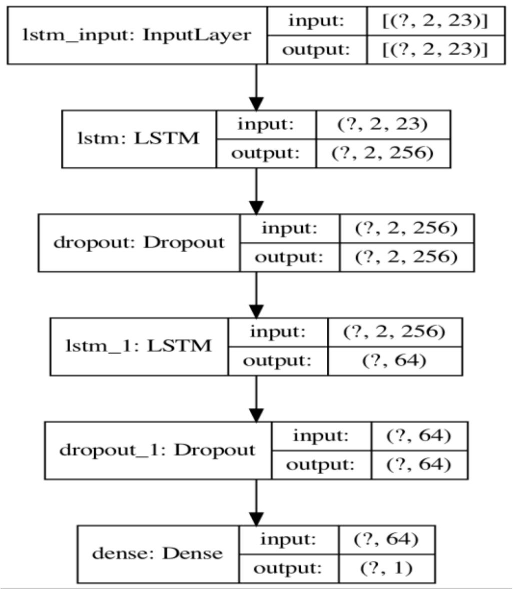
lstm_input: InputLayer | input: [(?, 2, 23)] | output: [(?, 2, 23)]
lstm: LSTM | input: (?, 2, 23) | output: (?, 2, 256)
dropout: Dropout | input: (?, 2, 256) | output: (?, 2, 256)
lstm_1: LSTM | input: (?, 2, 256) | output: (?, 64)
dropout_1: Dropout | input: (?, 64) | output: (?, 64)
dense: Dense | input: (?, 64) | output: (?, 1)

Fig.8 Model Architecture for training dataset

4.2 Evaluation

XGBoost:
XGBoost has a built-in cross validation method at each iteration that takes away the need to explicitly program the search and specify the exact number of boosting iterations required in a single run. The model has a great performance if given right combination of features. The model kept improving by adding more causal and correlated features to the target.

LSTM:
The loss function used with the model is the mean square error, because this is an error metric that is close to the root mean squared error which the Kaggle submission uses to evaluate our models.
The model was trained on logged normalized target variable.

$$y_{scaled} = \frac{\ln(1+y)}{\max(\ln(1+y))}$$

Final prediction is then inversed using the following equation:

$$y_{inversed} = e^{1+(y_{prediction} * max(\ln(1+y)))}$$

Furthermore, the data is clipped to between 0 and 40 before applying to normalization. Since the final prediction is between 0 and 20 (given by the Kaggle competition) the idea is that clipping with a large range would enable a more general model and then later we would clip the final prediction to be between 0 and 20.

4.3 Setup

XGBoost:
The model with XGBoost algorithm is coded in Python, Spyder IDE and the following libraries are imported in the model:
numpy, pandas, product form itertools, XGBRegressor, LabelEncoder from sklearn.preprocessing, seaborn

LSTM:
The LSTM model was trained using the Keras package in the Python programming language. The model was trained on the following hardware.

Hardware	Quantity
CPU	2.6 GHz Intel Core i7 6 cores, 12 threads
Ram	16GB 2400 MHz DDR4

Table.4 Configuration of hardware setup

5 Results

XGBoost:
As seen in Fig 9. no overfitting occurs. The relatively high difference between the test and training error can possibly be explained by unbalanced data or difficult data to predict on.
The RMSE values of the testing, training and final submission error in Table 5.

Dataset Name	RMSE
Train	0.82744
Test	0.91508
Eval	0.91570

Table.5 RMSE on training, testing and evaluation dataset

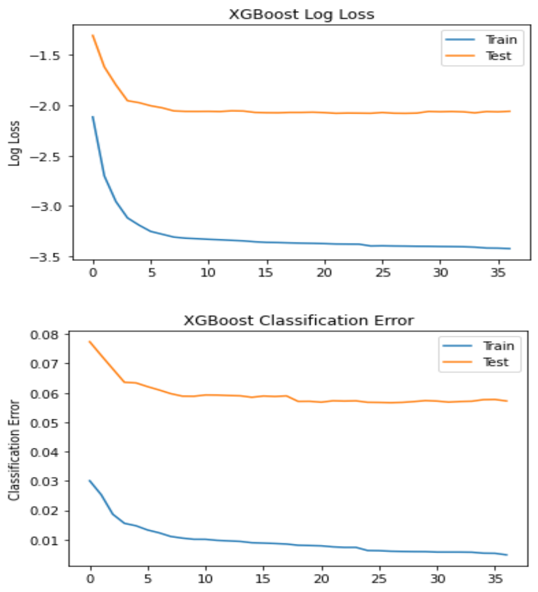

Fig.9 Graph of training and test error with epochs

LSTM:
It was observed that the model heavily overfitted after epoch 5 as shown in fig.10. To overcome this issue additional dropout layers were added to reduce the overfitting.

```
Val RMSE: 1.348
```

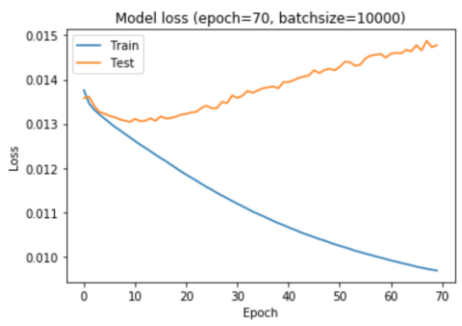

Fig.10. Graph of training and test error

As seen in fig.11 both overfitting and RMSE was reduced, with a final Kaggle RMSE of 1.051. From this experimentation it was clear that embedding significantly improved the model results.

```
Val RMSE: 1.239
```

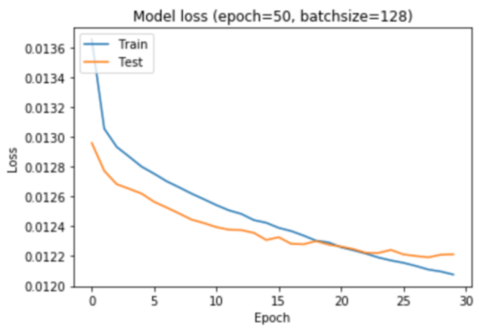

Fig.11 Graph of training and test error

6 Conclusion

XGBoost:
XGBoost has very robust properties such as regularization, weighted quantile sketch and built-in cross validation which result in better prediction performance and processing time compared to other more advanced algorithms.— Being simpler to implement, XGBoost enables rapid prototyping.
As for the future work, we are going to try different ensemble models such as combination of CatBoost, XGBoost and LightGBM to reduce RMSE and enhance our model.

LSTM:
Training time was a large bottle neck when it came to test different models, on average to get meaningful results the model needed to be trained for many epochs with small batch sizes. Without GPU acceleration training times exceeded 5 hours which severely hampered our ability to conduct rapid prototyping. Furthermore, the sparsity in the time series further introduced biasing that affected the results, a possible work around to this issue is to approach this problem from an ensemble perspective. The method involves training the LSTM on data that has good time density to ensure good model prediction and then train XGBoost for sparse time density data. Then the final predication is made based on the weighted average of the two models. The weight is defined by the sparsity of the data used for the prediction.

7 References:

[1] Explaining Feature Importance by example of Random Forest
https://towardsdatascience.com/explaining-feature-importance-by-example-of-a-random-forest-d9166011959e

[2] Entity Embeddings of Categorical Variables
https://arxiv.org/abs/1604.06737

[3] Entity-embedding-rossmann
https://github.com/entron/entity-embedding-rossmann/blob/master/models.py

[4] XGBoost Algorithm: Long May She Rein!
https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d

[5] Predict Future Sales Top 11 Solution
https://www.kaggle.com/szhou42/predict-future-sales-top-11-solution

[6] Deep Learning for Tabular Data
https://www.fast.ai/2018/04/29/categorical-embeddings/

[7] Mean Entity Embedding Encoding
https://www.saedsayad.com/encoding.htm

[8] LSTM Optimizer Choice
https://deepdatascience.wordpress.com/2016/11/18/which-lstm-optimizer-to-use/