Forecasting Grocery Sales with Past-Future Fusion LSTM

Feichi Yang
Department of Data Sciences and Operations
University of Southern California
Los Angeles, CA 90089
feichi.yang@usc.edu

Dihan Li
Department of Computer Science
University of Southern California
Los Angeles, CA 90089
dihanli@usc.edu

Leran Ma
Department of Computer Science
University of Southern California
Los Angeles, CA 90089
leranma@usc.edu

Tzuching Lan
Department of Computer Science
University of Southern California
Los Angeles, CA 90089
lantzuch@usc.edu

Abstract

This document is the report for the project of the course CSCI 567 in Fall 2022 at the University of Southern California. Our task is to join the Kaggle competition [1] and predict store sales of a grocery retailer Corporación Favorita in Ecuador. Related sales data from 2013-01-01 to 2017-08-15 is given, and we need to predict the sales from 2017-08-16 to 2017-08-31. With a preliminary understanding of the structured data provided, our team build a neural network with long short-term memory (LSTM) layers according to [2]. This machine learning model achieves a Root Mean Squared Logarithmic Error (RMSLE) of 0.39312. The code is available at https://github.com/LeranMa/Time-Series-Forecasting.

1 Feature engineering

Before building the neural network, we analyze and preprocess the given data. (Most steps in this section are done in the Jupyter notebook data_prepare.ipynb, which can be found in the given Git repository or the Gradescope code submission. Please run this Jupyter notebook before main.py when testing the code.)

Examining the provided csv files, we notice that the oil price is missing for some dates. To fill these empty fields, we use the average of the most recent previously known value and the next known value, instead of a moving average.

For each date, we specifically take out its month and the day of the week as two separate features. These two features usually have periodic information that a machine-learning model should capture.

For regional and local holidays, we explicitly store their state and city since this location information corresponds to the store locations.

For the continuous values, like oil price, sales, transactions, etc., we perform $log(x+1)$ transformation to normalize them.

Preprint. Under review.

2 Model overview

2.1 Overall framework

We divide features into three parts: features that represent the past, stable features, and features that represent the future. (Stable features are features that do not change with time, including store locations, store numbers, product family names, etc.) Below we introduce different treatments for these three types of features.

First, since all stable features are discrete, we learn the embedding vectors corresponding to these feature values through the embedding layer. (The length of all embedding vectors is 64.) After that, the information contained in these features is passed to a linear layer.

Because our treatment of features that represent the past and the future are consistent, we only explain the process of handling features that represent the past. The features that represent the past are divided into discrete and continuous. For discrete features, we also learn embedding vectors through an embedding layer. Then, these embeddings enter a linear layer together with those continuous features. The output of this linear layer and the output of the linear layer for the stable features are passed to the LSTM layer as input. Next, we use the output of the LSTM layer to extract information corresponding to the features that represent the past. The specific operation is to calculate the projection of family, store_nbr and stable feature from the output of LSTM layer.

Finally, we send the output obtained by these three paths to the predictor to get the final result. The predictor is composed of two linear layers, two corresponding dropout layers, and an ELU activation layer between the linear layers. The overall structure is shown in Fig.1.

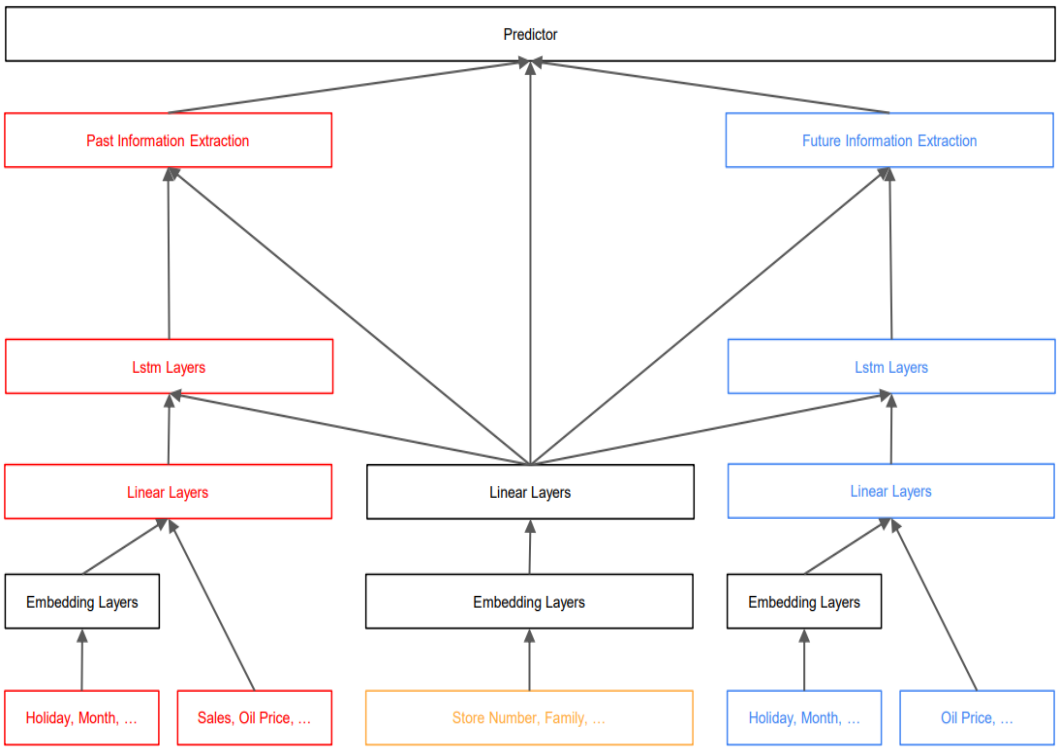

Figure 1: Overall structure of our method.

2.2 Implementation

The optimizer chosen is the RAdam optimizer, with a learning rate of 1e-3. The batch size is set to 1024 for training, and 512 for validation. The epoch is set to 10. The loss function is chosen to be RMSLE, which is the same as the evaluation metric used in the Kaggle competition.

3 Results

Fig.2 shows the changes of training loss and validation loss with respect to epoch during the training process. We saved the models of all epochs for testing and found that the model obtained by the fifth epoch had the best performance: a RMSLE of 0.39312. The models from later epochs had the problem of overfitting, so we finally chose the model obtained in fifth epoch.

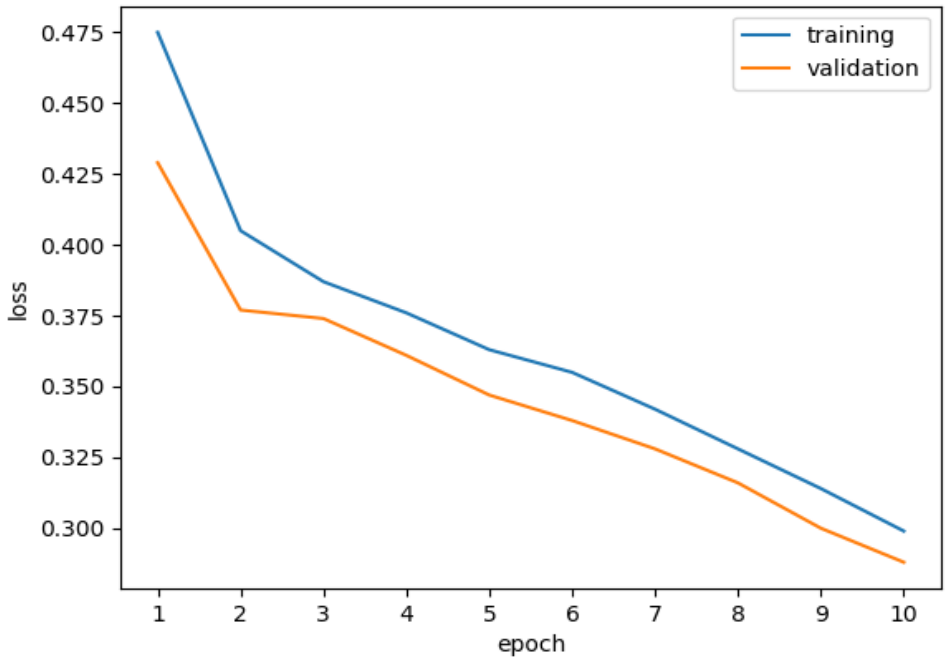

Figure 2: Training Loss and Validation Loss vs Epoch

4 Discussion

This section discusses potential improvements to our method and its strengths and weaknesses compared with some other methods that have achieved excellent performance on this topic.

4.1 Limitations compared with tree models

In order to predict target variables based on input data, people would choose supervised learning as their approach. Neural network and decision tree model are both popular methods of supervised learning. They both can deal with nonlinear relationships and correlations between variables. Although Recurrent Neural Network is the first thing we thought of when it comes to this time series problem, we found that many groups, who gave a presentation in class, chose decision tree model as their base model structure. For the purposes of exploring whether decision tree is better than recurrent neural network for this Kaggle competition, we consulted some material.

Generally speaking, a decision tree model performs better and takes less time than neural network model with small dataset. Besides, economists tend to use the decision tree model because it is has better interpretability: It is very intuitive and simple to see the importance of each feature. Moreover, compared to neural network, decision tree model takes less effort for data preprocessing. Normalization and scaling are not necessary. Embeddings are not required for discrete values. Missing values do not need to be filled out by guessing the correct value. There are multiple ways of handling them, like taking a default branch or distributing to a random child. Most importantly, much

less hyperparameters are learned compared to neural network, and it is easy to ensemble multiple tree models together to improve the overall performance.

4.2 Fourier extrapolation

After consulting and discussing, our group determined to do more effort on feature engineering, hoping that Fourier transform could help LSTM compete with the decision tree model.

The Fourier transform can capture valuable periodic information. Therefore, in order to improve the performance of our neural network, we tried to incorporate a simple Fourier extrapolation. Although the result was not ideal, still we think this idea is worth further exploitation.

When analyzing the data, we found that sales commonly show a clear periodic change on a weekly basis. Hence, we tried a simple Fourier extrapolation: For each pair of a store and a product, we take its sales of the last 7 days and calculate the discrete Fourier transform of these 7 numbers. Then, we use this transform to predict sales in the next 16 days. For example, as shown in Fig. 3. The blue line is the actual 23-day automotive sales from 2013-01-01 to 2013-01-23. The yellow line is the Fourier extrapolation based on the first 7 days. We can find that the Fourier series indeed fairly accurately captures the periodic information within a week.

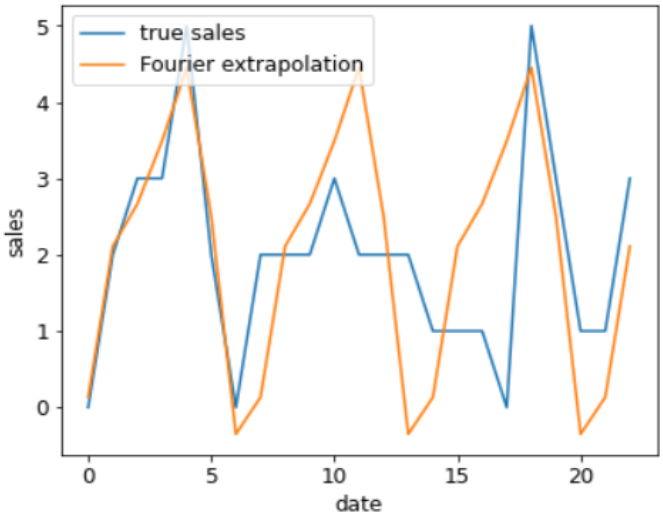

Figure 3: Forecasting 16-day automotive sales with 7-day sales using Fourier extrapolation

After simple parameter tuning, we predicted the sales from 2017-08-16 to 2017-08-31 based on the sales from 2017-08-09 to 2017-08-15 with this Fourier extrapolation. The resulting RMSLE was 0.52245, which is quite good for a simple model with only sales in the last 7 days. Hence, we believed that this Fourier extrapolation could help our current model. Even if the accuracy of the neural network would not increase, it still can help the model to converge faster. Therefore, on the basis of the original neural network, we integrated this Fourier extrapolation as part of the input. The resulting model RMSLE was 0.411. Although this number was higher than our original loss, we still think this was a worthwhile attempt, and we would probably continue investigating Fourier transforms if we had more time for the project.

5 Instructions to run the code

Please build and test the model according to the following order.

1. Run the Jupyter notebook data_prepare.ipynb. Remember to change the paths to the csv files.
2. Run main.py.
3. Run test.py.

Training may take 4 to 6 hours. (Depending on the machine used for training, this time may be shorter or longer.) The final results are stored in the file submission.csv.

References

[1] Alexis Cook, DanB, HCL-kanishkaa, inversion, Ryan Holbrook. (2021). Store Sales - Time Series Forecasting. Kaggle. `https://kaggle.com/competitions/store-sales-time-series-forecasting`

[2] Bryan Lim, Sercan Arık, Nicolas Loeff, Tomas Pfister. (2021). Temporal Fusion Transformers for interpretable multi-horizon time series forecasting. International Journal of Forecasting. 37. 10.1016/j.ijforecast.2021.03.012.