CS273P Project Final Report

Chandler Gutlay, Lucas Morgan, Adelynn Paik

Abstract

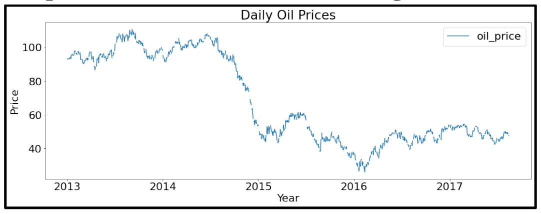
Our group explored grocery store sale predictions based on various time series data of oil prices, holiday celebrations, and information about the stores and their inventory. After appropriately merging and encoding all data, we implemented a temporal fusion transformer model and a recurrent neural network, given their adeptness with time series data. Both models demonstrated great success, particularly after hyperparameter tuning.

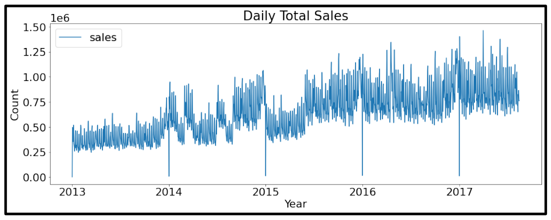
1. Introduction and Data Exploration

Our group elected to pursue the ninth Kaggle project option. In this project, we were tasked with forecasting sales for grocery stores in Ecuador based on time series data. The data ranged from January 1st, 2012 to August 14th, 2017. The various .csv files provided to us on the site include information such as:

- **Holidays:** All holidays within that five-year span were associated with a date, a label of whether it was a local/regional/national holiday, and logistical information about how that holiday was celebrated (ex.- Transfer days indicated a holiday that was celebrated on a different day than the actual date).
- **Oil prices:** Oil prices for the country of Ecuador were by given day.
- **Store data:** Each store was given a store number, city, state, one of five store types, and cluster number.
- **Product-specific information:** We were told the number of sales and the number of items on promotion for each product family (automotive, baby care, beauty, etc.).

To develop an intuition of the general trends present in the data, we plotted oil prices, total sales, and total items on promotion per day summed over all stores and all product families as seen in Figure 1 below.





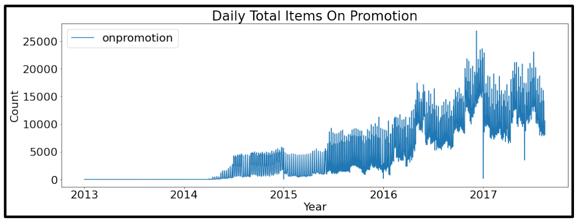
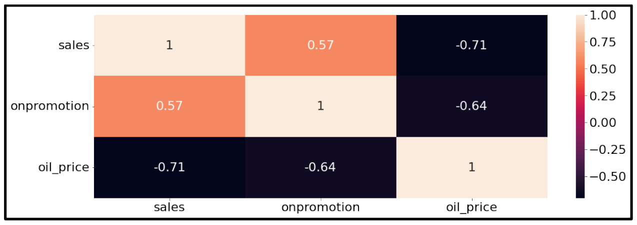


Figure 1. Plots of oil prices, sales, and total items on promotion over time.

The correlation between sales, oil prices, and items on promotion yielded the following results shown in Figure 2.



	sales	onpromotion	oil_price
sales	1	0.57	-0.71
onpromotion	0.57	1	-0.64
oil_price	-0.71	-0.64	1

Figure 2. Correlation matrix of oil prices, sales, and total items on promotion over time.

The shapes of the graphs and the correlation matrix suggest a relationship between rising oil prices and decreasing sales, but there are no definitive trends from this preliminary analysis. Likewise, plotting the sales of product families summed across stores remained inconclusive. The violin plots revealed the disparity in number of sales. Below in Figure 3 are the plots for the first product family, automotive.

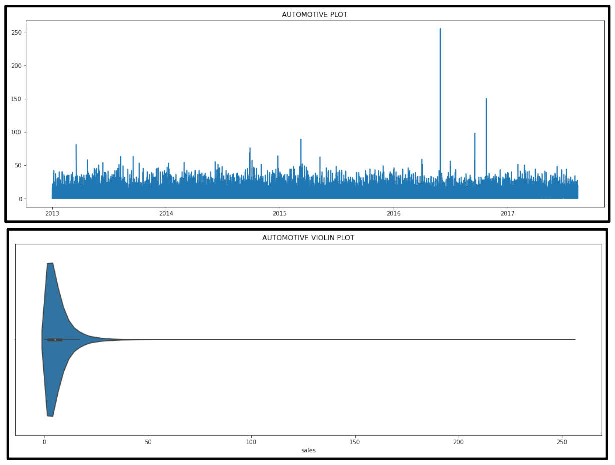


Figure 3. Time series plot and violin plot for automotive product family.

All of the above visuals warranted a closer look at the impact of holidays on grocery sales, as well as the possibility of exploring stores and product families separately in our models.

2. Data Preprocessing and Feature Design

Our most immediate challenge was to merge the various datasets into one for training. They were all inter-connected in various ways, whether by a store existing in a location that celebrated a specific holiday or oil prices corresponding to store sales by date. Additionally, some of the data such as store type (A, B, C, etc.) or holiday type (Transfer, Bridge, Additional, etc.) were categorical.

We began by merging the main training dataframe with the oil data on dates, then with the stores data on store number. The next task was to merge the holiday information with the corresponding store and date, depending on the location of where the holiday was celebrated. Because none of the holiday descriptors were numerical, we decided to use one-hot encoding. After merging the dataframes and converting all categorical labels, our training dataframe stored the date, store number, sales and items on promotion by product family, oil price by day, and one-hot encoded columns for: product type, local/regional/national holidays, store types A through E, and normal/additional/bridge/transfer holiday celebration types. The encoded columns stored values of 0 or 1, so as to not assign higher or lower values to any non-numerical category. Lastly, since each row described one product family, we removed one of the product families to avoid multicollinearity.

Because the oil dataset is based on market prices, and some days the market is not open for trade, there are missing dates and null values. To deal with this problem, we used an interpolation method to fill in the null values for oil price.

The following two sections will describe our model exploration and performance validation processes for each of our models.

3. Model 1: Temporal Fusion Transformer

3.1. Model Exploration

Originally, we had intended to explore the use of a Transformer model in order to solve this Kaggle problem. However, further research indicated that a Temporal Fusion Transformer (TFT) model would also be well-suited to a time series forecasting problem such as this. A TFT is a hybrid model, combining the multi-head attention module of a transformer, which is used to identify and weigh important patterns between input, and LSTM encoders and decoders, which are used to find patterns between time steps and surrounding values. Additionally, TFT's also use gated residual network that allows the model to skip over unimportant inputs (Onnen).

Before feeding the training data into a network, we added a time index column to keep track of the time steps across the dataset. Then the data was converted into a time series dataset, where *sales* was the target, and the columns identifying the time series were *family* and *store_nbr*. After doing this, we attempted to achieve the best model possible through parameter tuning. Because learning rate would most likely be one of the more important hyperparameters for the model, our initial model consisted of relatively neutral values for other parameters such as *hidden_size*, *attention_head_size*, and *hidden_continuous_size*, while also tuning the learning rate based on the given training and validation data. Below is the figure illustrating the process of finding an optimal learning rate.

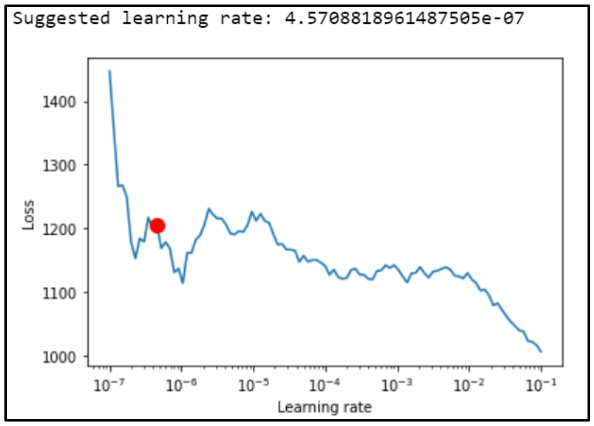


Figure 4. Finding the optimal learning rate for the TFT model.

3.2. Performance Validation

After fitting the basic model with an optimized learning rate, the model performance against the validation set was poor. In an attempt to improve the model's performance, we opted to optimize the remaining hyperparameters using the *optimize_hyperparameters* function and the Optuna test. Additionally, we implemented early stopping and emphasized use of dropout in order to prevent potential overfitting from over-tuning the hyperparameters. Optuna runs multiple different trials using a range for each parameter and either an Adam, RMSprop, or Stochastic Gradient Descent optimizer. We are then able to extract the best parameters from those trials and apply them to our model. The case that our model found to be optimal set *hidden_size* to 40, *dropout* rate to .19, *hidden_continuous_size* to 25, *attention_head_size* to 3, and *learning_rate* to .022. These settings for the hyperparameters are set at reasonable levels. The multi-head attention size being set to 3 allows for more connections and preservation of long range relationships. Similarly, a larger hidden size and hidden continuous size, corresponding to the LSTM encoder and decoder blocks, help to better identify patterns between time steps and their nearby values.

One value of note is the learning rate. The learning rate selected by the Optuna-optimized model is significantly higher than that of the learning-rate tuned model. This may be due to the increased model size, which in turn allows for a larger step size. The difference between the Optuna-optimized model and the learning-rate tuned model exemplifies the importance of taking a specific model's architecture into account and selecting parameters to tune based upon the given application. In this case, the Optuna-optimized model had a reasonable improvement in loss, decreasing by about 4%. Below is a figure that shows an example of a prediction made by the model.

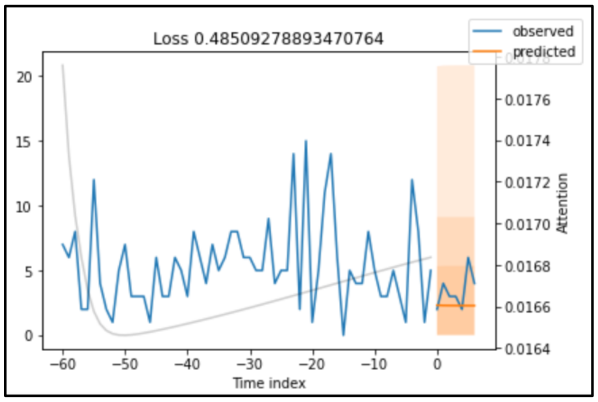


Figure 5. Final performance of the TFT model, where we can observe the prediction (orange line) versus the actual observation (blue line).

Observing the shape of the model's prediction, it has become clear that, though the hyperparameters were now optimized, the model's ability to generalize can certainly be improved. Our previous concern with overfitting led to too conservative of a training approach, not allowing the model to correctly learn and predict on the data as expected. By increasing the limit training batch number in the trainer's parameters, we should get a model that is able to better generalize. For example, after increasing this parameter from 30 to 1000 on the learning-rate tuned, the loss decreased from 0.43 to 0.37.

4. Model 2: Recurrent Neural Network with Long Short-Term Memory Gate

4.1. Model Exploration

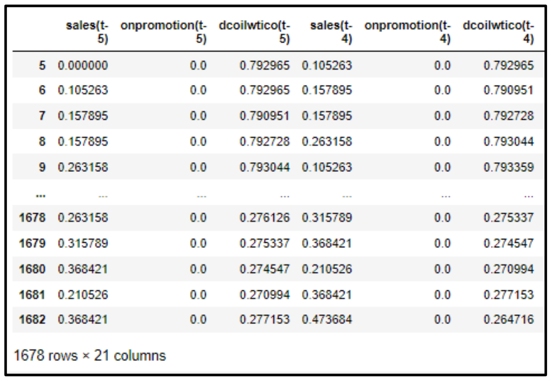
Beyond the conventional approach using ARIMA models, we decided to attempt the problem using recurrent neural networks. LSTM networks are well-suited to making predictions on time series data, so we attempt to use one here. RNNs already keep track of long-term dependencies, but they computationally suffer from the vanishing gradient problem where the long-term gradients can tend to zero. LSTMs differ in that they introduce feedback connections through gates.

In the design of our model, we use two LSTM layers with hidden units of 30, and add two dropout layers in between each LSTM layer. For our output layer, our output is based on the number of time steps we need to predict, i.e. in this case, 16 days. For our optimizer, we are using the popular Adam gradient descent algorithm instead of the standard Stochastic Gradient Descent, and our loss function is the mean squared error loss function.

4.2 Additional Feature Pre-processing

In our test dataset, we are given the features *store_nbr*, *date*, *family*, and *onpromotion* and our target is *sales*. Because of the nature of time series data, we will create a model and fit it on different combinations of store_nbr and family. Thus, we will have 54 store x 33 family combinations of data, and our time series for each combination will consist of the features *onpromotion*, *dcoilwtico*, and *sales*.

For supervised learning, i.e. time series data regarding LSTM and RNN networks, we must arrange the data in a way that is readable by the model. Here, we reference Jason Brownlee's *series_to_supervised()* function, where we are converting our multivariate time series problem into a supervised learning problem readable by LSTMs ("How to Convert a Time Series"). We frame our input variables as *t-k* time steps, where *k* is the number of days we decide to use to predict the values at *t* to *t+j* time steps, where *j* is the number of days we need to predict. Here we use 8 as our *k* (as explained below in section 4.3), and 15 as our *j* as we have 16 days to forecast. Our features *onpromotion*, *dcoilwtico,* and *sales* are then formatted in such a way that each feature has columns dedicated to the number of previous *k* shifts and future *j* shifts.



	sales(t-5)	onpromotion(t-5)	dcoilwtico(t-5)	sales(t-4)	onpromotion(t-4)	dcoilwtico(t-4)
5	0.000000	0.0	0.792965	0.105263	0.0	0.792965
6	0.105263	0.0	0.792965	0.157895	0.0	0.790951
7	0.157895	0.0	0.790951	0.157895	0.0	0.792728
8	0.157895	0.0	0.792728	0.263158	0.0	0.793044
9	0.263158	0.0	0.793044	0.105263	0.0	0.793359
...	...	...	...	...	...	...
1678	0.263158	0.0	0.276126	0.315789	0.0	0.275337
1679	0.315789	0.0	0.275337	0.368421	0.0	0.274547
1680	0.368421	0.0	0.274547	0.210526	0.0	0.270994
1681	0.210526	0.0	0.270994	0.368421	0.0	0.277153
1682	0.368421	0.0	0.277153	0.473684	0.0	0.264716

1678 rows × 21 columns

Figure 6. A snippet of a sample dataframe for k=5.

4.3. Performance Validation

We employ transfer learning for this problem as we are only training one model based off of store 1 and family 'AUTOMOTIVE', and fitting the rest of our data, split up by store number and family combinations, to this model. We use this method because trying various combinations as the initial model had negligible change to our MSE. Below is the MSE loss for the initial combination.

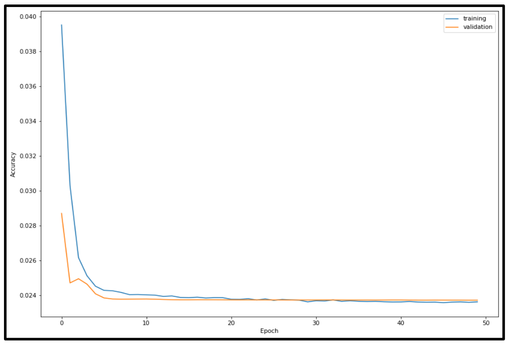


Figure 7. Initial MSE loss.

To assess model performance in multiple splits of our data, we would like to use some form of validation. Cross validation is unfeasible for time series data as if we randomly split along folds, it does not make sense to use future values to forecast past values. Thus, we performed *n*-step ahead cross validation where we gradually built our window of training data. We split the data initially with 20% train, 80% validation, and grew the window each time. This resulted in very similar MSE losses, showing that our model is performing quite well for different splits.

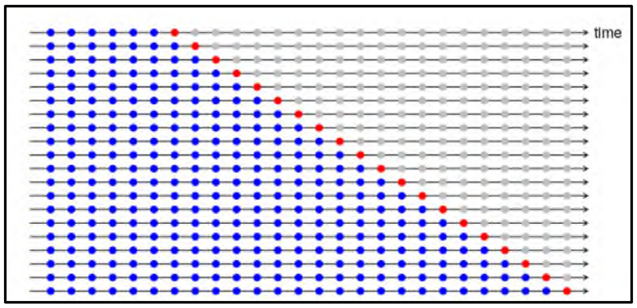


Figure 8. One-step ahead cross validation (StackExchange).

We also took a look at how the model's accuracy would change if we varied our *k* previous steps. Initially, we started with *k* = 50, and tested various *k*'s between 7 and 100. We found that the MSE values did not change significantly between different *k*'s, as *k* = 8 had a 0.0218 MSE validation loss after multiple epochs while *k* = 32 had a 0.0214 MSE validation loss. As choosing a lower *k* significantly reduces the number of trainable parameters, we chose *k* = 8. We also varied the number of *epochs* and *batch_size* and found that around *batch_size* = 150 and *epochs* = 50 performed the best.

5. Adaptation to Underfitting and Overfitting

When exploring the TFT model, our original fear of overfitting led us to place heavy emphasis on early stopping and dropout rate. Through the use of Optuna to find optimal hyperparameters, we arrived at a model with a higher learning rate than we anticipated. This eventually led us to achieve a loss rate of 0.37 with a final TFT model that both generalized well and fit the data.

With regards to the LSTM model, we added two dropout layers after each LSTM layer to account for overfitting. Additionally, we used early stopping to avoid overtraining our model. Because the training and validation losses are comparable, there is no evidence of overfitting. Our model's ability to converge and respond to various parameter tunings also does not suggest that we underfit the data.

6. Conclusion

Although the prospect of training a high-dimensional time series dataset was daunting at first, both the temporal fusion transformer model and the recurrent neural network with long short-term memory gate proved to be successful. Given that the current highest scoring model on Kaggle has a loss of 0.3818, our models are on par with some of the top contestants. The LSTM model eventually outperformed the TFT with an MSE loss of 0.02 versus 0.37, but either option would be suitable for future problems similar to this one. Our two greatest indicators of success are that both models converged on the training set as well as the validation set, and they responded well to hyperparameter tuning.

7. Individual Contributions

The group as a whole discussed how to group, merge, and encode the Kaggle data. Adelynn experimented with an MLP model (included in code submission but not in the report for brevity) and wrote sections 1, 2, and the discussion portions of this report. Chandler researched, created, and adapted a Temporal Fusion Transformer model and authored all of section 3 based on his own findings. Lucas researched, created, and adapted an LSTM model and authored all of section 4 based on his own findings.

Sources

Brownlee, Jason. “How to Convert a Time Series to a Supervised Learning Problem in Python.” *Machine Learning Mastery*, 8 May 2017. https://machinelearningmastery.com/convert-time-series

Brownlee, Jason. “How to Develop Multilayer Perceptron Models for Time Series Forecasting.” *Machine Learning Mastery*, 9 November 2018. https://machinelearningmastery.com/mlp-for-time-series

“Interpretable Deep Learning for Time Series Forecasting.” *Google AI Blog*, 13 December 2021. https://ai.googleblog.com/2021/12/interpretable-deep-learning-for-time.html

“Leave one out cross validation for LSTM.” *StackExchange*, 17 December 2018. https://stats.stackexchange.com/questions/350655/leave-one-out-cross-validation-for-lstm

Onnen, Heiko. “Temporal Fusion Transformer: A Primer on Deep Forecasting in Python.” *Towards Data Science*, 28 December 2021. https://towardsdatascience.com/temporal-fusion-transformer