CSCI567: Project report for Group 46

Han Kyul Kim
Department of Industrial & Systems Engineering
University of Southern California
hankyulk@usc.edu

Kenny Low
Department of Computer Science
University of Southern California
kennylow@usc.edu

Yechan Seo
Department of Computer Science
University of Southern California
yechanse@usc.edu

Yuhang Xiao
Department of Computer Science
University of Southern California
yxiao776@usc.edu

Abstract

This final report contains a summary of different approaches that our group explored in predicting the future sales of various product families for different stores in Ecuador. During this process, we not only had the experience of applying different machine learning algorithms that we learned throughout this semester but also learned to appreciate the value of feature engineering in addressing machine learning problems in the real world. By utilizing various off-the-shelf machine learning packages such as `Darts` and `Optuna`, we were able to train and validate a LightGBM model that exhibited a test error of approximately 0.384.

1 Introduction

This project aims to predict future sales of multiple product families sold at different stores of Corporación Favorita, a large Ecuadorian grocery retailer. By applying various time-series forecasting methods and machine learning models covered throughout this semester, we generated data-driven forecasts of store sales based on historical sales data. By incorporating additional features such as holidays and oil prices, we explored and optimized a time-series predictive model that accurately predicts the unit sales for items sold at different stores. Based on careful model selection and hyperparameter tuning, we created an effective predictive model that resulted in the root mean squared logarithmic error (RMSLE) of 0.384 in the Kaggle competition.

2 Background: algorithms explored

We explored not only the machine learning algorithms that we had covered in class but also forecasting algorithms that have been traditionally used in predicting time-series data. For easy data interface and quick testing, we adopted the Python package `Darts`[1] for all of the data pre-processing and model training.

2.1 Forecasting algorithms

Forecasting algorithms refer to a set of models that use the patterns and trends in previous observations of time-series data to predict future data points. In this project, we tested two different forecasting methods: Holt-Winters' exponential smoothing (Holt [1960], Winters [1960]), and AutoARIMA

[1] `https://unit8co.github.io/darts/README.html`

Preprint. Under review.

(Hyndman and Khandakar [2008], Wang et al. [2006]). In Holt-Winters' exponential smoothing, a pre-defined seasonality or trend is estimated from the past data points, which is added to the local variations, often denoted as a level, at each time point.

On the other hand, autoregressive integrated moving average (ARIMA) (Box et al. [2015]) enforces stationarity in a time-series, thereby removing the impacts of seasonality or trends. To induce stationarity, ARIMA learns the coefficients for every differences and error residuals between the data point at a fixed time t and previous data points within $t - p$ for pre-defined p. Based on this formulation, AutoARIMA introduces a robust method of optimizing these coefficients based on comparing the estimated coefficients on various information criterion such as Akaike Information Criterion (Sakamoto et al. [1986]), and Bayesian Information Criterion (Schwarz [1978]).

2.2 Machine learning algorithms

As for machine learning algorithms, we tested algorithms that we had covered in class, such as linear regression, random forest (Breiman [2001]), gradient boosting regression (Friedman [2001]), and gated recurrent units (GRU) (Cho et al. [2014]). In the case of linear regression, random forest, and gradient boosting regression, we used the default package interface provided by `Darts` that allowed us to directly input data pre-processed in `Darts` to linear regression and random forest implemented in `scikit-learn`[2] and the gradient boosting regression provided in `LightGBM`[3]. As for GRU, we implemented a model with 128 hidden nodes using `PyTorch`[4].

3 Models from scratch

During our initial attempts, we tested various algorithms from the previous section with some feature engineering. As explored in our homework, we included the rolling average of oil prices over 7 days (`oil_price`) and the working days (`wd`) as potential features in our initial modeling attempts. Not only was a simple linear regression based on these covariates generate a decent baseline model, but we also speculated that the changes in the oil price would have a direct impact on the price of goods, thereby affecting the level of sales in the end. Furthermore, we also hypothesized that the sales from the previous n days (`sales_past_n`) would be crucial indicators for predicting future sales. For example, in order to predict the sales at day t, we would use the sales from previous $\{t - i : i \in [1, 2, ..., n]\}$ days, in which n serves as a hyperparameter that would need to be further fine-tuned.

One of the most critical modeling design choices that we faced was the scope of an individual prediction model. As the competition required us to predict sales of different families of products (`family`) in multiple stores (`store_nbr`), we could potentially generate a single global prediction model, `family`-level models or `family-store_nbr`-level models. As we assumed that future sales in a store would vary depending on the specific location of the store and the products, we decided to generate a prediction model for each `family` and `store_nbr`. Since there were 54 unique `store_nbr` and 33 unique `family`, we trained $54 \times 33 = 1782$ models for each algorithm we tested.

In order to prevent our models from overfitting to the test dataset used in the Kaggle competition, we generated a validation dataset (D_v) from the training dataset (D_{tr}), which contains the sales data from 2013-01-01 to 2017-08-15. Since $D_v \subset D_{tr}$, we initially trained each of our algorithms with $D_{tr} - D_v$ and compared their RMSLE in D_v. Subsequently, we selected the machine learning or forecasting algorithm with the lowest RMSLE and re-trained it using the entire D_{tr} to generate the final predictions for the submission. Although we tested various data split strategies for creating D_v, generating D_v to be all sales data from 2017-07-31 to 2017-08-15 was the most consistent indicator of the model performance in the test dataset. Consequently, all of the RMSLE reported in this paper were computed from D_v ranging between 2017-07-31 and 2017-08-15.

Based on the feature selection and validation split described above, we report and compare the performance of different machine learning and forecasting algorithms we explored from scratch in this project (Table 1). Due to the limitations in the number of daily submissions, we report the test errors for only a few models with a relatively lower validation error. Furthermore, we tested k-nn and

[2]`https://scikit-learn.org/stable/`

[3]`https://lightgbm.readthedocs.io/en/v3.3.2/`

[4]`https://pytorch.org/`

support vector regression algorithms but did not include them in Table 1 as their validation errors were significantly higher than those reported in the table. We noticed that the models we built from scratch were not significantly better than our homework's baseline linear regression model, which achieved an RMSLE of 0.4581. Therefore, we explored and adopted different feature engineering strategies publicly shared in the competition's official discussion board.

Algorithm	**Features used**	**Algorithm Parameters**	**Validation Error**	**Test Error**
Exponential smoothing	Sales	seasonality: N/A # of samples: 10,000	0.4232	0.4684
		seasonality: 7 # of samples: 10,000	0.3922	**0.4316**
		seasonality: 15 # of samples: 10,000	0.4233	-
		seasonality: 30 # of samples: 10,000	0.4236	-
AutoARIMA		# of samples: 1,000	0.418	-
LightGBM	`sales_past_30`	-	0.4302	-
	`sales_past_60`		0.4268	-
	`sales_past_90`		0.4251	-
	`sales_past_180`		0.4287	-

	`sales_past_60`
	`oil_price`
	`wd`
			0.4278	-

	`sales_past_90`
	`oil_price`
	`wd`
			0.4193	-

Linear Regression	`sales_past_1`
	`oil_price`
	`wd`
		-	0.4414	0.5174

	`sales_past_3`
	`oil_price`
	`wd`
			0.426	-

	`sales_past_7`
	`oil_price`
	`wd`
			0.3959	0.4521

	`sales_past_60`
	`oil_price`
	`wd`
			0.4094	-

	`sales_past_90`
	`oil_price`
	`wd`
			0.4706	-

Random Forest	`sales_past_60`	# of estimators: 300 max_depth: 3	**0.3756**	0.4358
	`sales_past_90`		0.3836	0.4492
	`sales_past_120`		0.3891	0.4691
GRU	`sales_past_100`	epochs: 100 learning rate: 10^{-3}	1.0421	-
		epochs: 100 learning rate: 10^{-5}	1.7703	1.2592

Table 1: Comparison of different machine learning and forecasting algorithms built from scratch. Other hyperparameters not listed in the table are set to their respective default values used in `Darts`.

4 Fine-tuning publicly available models

4.1 Feature engineering

Based on the results of our initial model efforts as described in Table 1, we realized the importance of feature engineering. Therefore, we explored various feature engineering strategies adopted by publicly available models shared in the competition's official discussion board. Among various models available in the discussion board, we referred to a reference notebook shared by a user named Ferdinand Berr[5] as he similarly uses `Darts` and LightGBM algorithm as we had previously explored.

Regarding the number of prediction models and the input features, there were several noticeable differences between this reference model and our models from Section 3. Instead of training a single model for every combination of `family` and `store_nbr`, this reference model adopted training a single model for each `family`, resulting in a smaller number of models of 33. As the sales from different stores with the same `family` serve as an input to the model, the reference model adopts `city`, `state`, `type`, and `cluster` of each `store_nbr` as provided in `stores.csv` as additional input features.

In addition to these features, we followed the feature engineering strategies used in the reference notebook. The complete list of all features used in fine-tuning our reference model is summarized in Table 2. Compared to the features used to train the models from Section 3, we notice that the reference model fully utilizes features from all of the files provided in this competition, which once again highlights the critical importance of feature engineering in machine learning projects.

Due to the time constraint, we did not explore all of the algorithms from Section 3. However, we focused on fine-tuning the hyperparameters of lightGBM. Furthermore, as for the model selection, we similarly defined D_v to be sales ranging between 2017-07-31 and 2017-08-15 and calculated the validation error.

Feature Name	Description	Source File
`sales_ma_7`	7 days moving average of sales	train.csv
`sales_ma_28`	28 days moving average of sales	
`year`	one-hot encoding of the year in which a data point is recorded	
`month`	one-hot encoding of the month in which a data point is recorded	
`day`	one-hot encoding of the day in which a data point is recorded	
`dayofyear`	the ordinal day of the year	
`weekday`	the day of the week with Monday=0, Sunday=6.	
`weekyear`	the ordinal week of the year.	
`timesteps`	Increasing index of each time point	
`oil_ma_7`	7 days moving average of oil price	oil.csv
`oil_ma_28`	28 days moving average of oil price	
`transactions_ma_7`	7 days moving average of the number of transactions	transactions.csv
`transactions_ma_28`	28 days moving average of the number of transactions	
`transferred`	A binary variable indicating a holiday that was moved to another date by the government	holidays_events.csv
`national_holiday`	A binary variable indicating if a date is a national holiday	
`earthquake_relief`	A binary variable indicating dates when people rallied in earthquake relief efforts	
`christmas`	A binary variable indicating if a date is a Christmas	
`football_event`	A binary variable indicating if a football event is on the day	
`regional_holiday`	A binary variable indicating if a regional holiday falls on the date	
`national_event`	A binary variable indicating if a national event occurs on the date	
`local_holiday`	A binary variable indicating if a local holiday occurs on the date	
`workday`	A binary variable indicating if a local holiday occurs on the date	
`wage`	A binary variable indicating a date on which wages are paid	
`promotion_ma_7`	7 days moving average of the total number of items on promotion	train.csv
`promotion_ma_28`	28 days moving average of the total number of items on promotion	

Table 2: List of all features used in a reference model

4.2 Hyperparameter tuning

Among various hyperparameters of LightGBM, we chose to focus on three input-related hyperparameters: `lags`, `lags_past_covariates`, and `lags_future_covariate`. `lags` represents the number of previous sales data points used as a part of the input features. Furthermore, we suspected there would also be a significant correlation between the predicted sales and both previous and future values of additional features listed in Table 2. Therefore, we also evaluated the impact of

[5] `https://www.kaggle.com/code/ferdinandberr/darts-forecasting-deep-learning-global-models`

`lags_past_covariates`, and `lags_future_covariate`, which represent the number of historical and future data points of the features listed in Table 2.

For example, to predict the sales on a fixed day t, `lags=60` indicates sales between $t-60 \sim t-1$ days would be used as input features. Similarly, `lags_past_covariates` $= \{-5, -4, -3\}$ represents that the past features from Table 2 between $t-5$ and $t-3$ would be included as inputs. Furthermore, `lags_future_covariates` $= (14, 1)$ would imply that the future features from Table 2 between t and $t-14$ (assuming forecast horizon ≥ 15) would also serve as additional inputs to the algorithm.

As we trained a model that would outperform both the one from the reference notebook and those explored in Section 3, we used a Python package `Optuna`[6] that systematically searches for robust hyperparameters using our validation dataset D_v. In this library, we defined an objective function, which would return an error value, in this case, RMSLE, that we aim to minimize. The defined objective function enabled us to select 33 LightGBM models with the best set of hyperparameters that would induce both the lowest validation error and, potentially, the test error.

Based on this automatic hyperparamerter search from `Optuna`, hyperparameters `lags=63`, `lags_future_covariates` $= (14, 1)$, and `lags_past_covariates` $= \{i : -32 \leq i \leq -16\}$ were deemed to be the best set of hyperparameters by `Optuna`. This result also suggests that while the past values may have some influence in forecasting future values, using a longer `lags` does not necessarily lead to a better prediction.

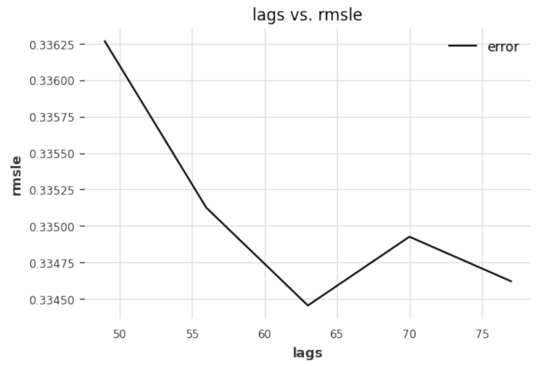

Figure 1: RMSLE vs. `lags` with other two hyperparameters fixed at the values listed in Table 3

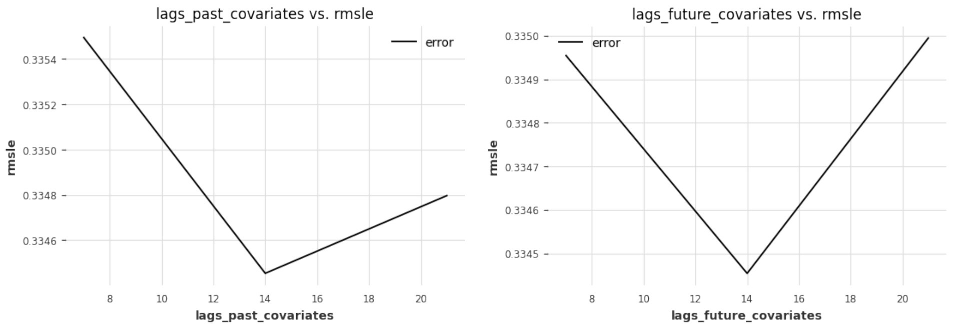

Figure 2: RMSLE vs. `lags_past_covariates` and `lags_future_covariates` with other two hyperparameters fixed at the values listed in Table 3, respectively. For simplicity, X-axis represents the total lag days contained in `lags_past_covariates` (starting from $t-16$) and `lags_future_covariates` (starting from $t-1$)

The model trained with this set of hyperparameters found by `Optuna` resulted in a test error of 0.3849, which was close to the test error of 0.38558 in the original reference model. As the hyperparameter search through `Optuna` took a long time, we also manually performed a series of hyperparameter

[6] `https://optuna.org/`

search. Based on the initial values of each hyperparameter used in the reference model, we perturb the values of `lags`, `lags_past_covariates`, and `lags_future_covariates`, independently. When plotting the changes of RMSLE in D_v as shown in Figures $1 \sim 2$, we noticed that the original reference model was almost effectively fine-tuned with respect to these three hyperparameters except the `lags_past_covariates`. By changing `lags_past_covariates` to $\{i : -29 \leq i \leq -16\}$, we get a better test score of 0.38403 surpassing the original reference.

While our fine-tuning efforts with `Optuna` did not result in a superior prediction score, it was a good learning experience to familiarize ourselves with `Optuna`, which significantly reduced the implementation time required in designing an effective hyperparameter search.

In the official Kaggle competition, the model with the best test RMSLE of 0.38403 resulted from the hyperparameters listed in Table 3. Furthermore, this set of hyperparameter also resulted in the lowest validation error of 0.3396.

`lags`	`lags_past_covariates`	`lags_future_covariate`
63	{i:-29 $\leq i \leq -16$}	(14,1)

Table 3: List of the hyperparameters that results in be lowest test error

5 Conclusion

Throughout this project, we applied and tested various machine learning and forecasting algorithms to generate accurate predictions of future sales. During this process, we had a chance to review the concepts of linear regression, random forest, GRU, gradient-boosting regressions, and several baseline forecasting methods. Furthermore, by using the validation dataset, we had a chance to observe the impacts of different hyperparameters have on these models.

The key takeaway from this project was the importance of feature engineering and the challenges of hyperparameter tuning in machine learning projects. To gain a better insight into feature engineering, we explored various publicly shared models in the official Kaggle competition discussion board. However, we were surprised to witness the sheer efforts required to find even a set of adequately performing features. Furthermore, from our experience with manual feature engineering and hyperparameter searching, we came to appreciate the convenience that the packages such as `Darts` and `Optuna` offered. Based on these never-ending cycles of feature engineering-model training and validation-hyperparameter tuning, we generated a LightGBM model with an RMSLE of 0.384.

References

George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

Charles C Holt. Planning production, inventories, and work force. 1960.

Rob J Hyndman and Yeasmin Khandakar. Automatic time series forecasting: the forecast package for r. *Journal of statistical software*, 27:1–22, 2008.

Yosiyuki Sakamoto, Makio Ishiguro, and Genshiro Kitagawa. Akaike information criterion statistics. *Dordrecht, The Netherlands: D. Reidel*, 81(10.5555):26853, 1986.

Gideon Schwarz. Estimating the dimension of a model. *The annals of statistics*, pages 461–464, 1978.

Xiaozhe Wang, Kate Smith, and Rob Hyndman. Characteristic-based clustering for time series data. *Data mining and knowledge Discovery*, 13(3):335–364, 2006.

Peter R Winters. Forecasting sales by exponentially weighted moving averages. *Management science*, 6(3): 324–342, 1960.