M5 Forecasting - Accuracy

Jan Ondruch
s1045948

Koen Baarda
s4613147

David Roefs
s4666623

Radboud University
Nijmegen, The Netherlands

1 Introduction

New advances in machine learning make predicting future trends more feasible and easier to develop. Predicting future sales can be an important advantage in the retail sector, reducing waste and avoiding the risk of items being out of stock.

To advance the theory and practice of forecasting, the University of Nicosia is hosting the M5 Competition on the machine learning platform Kaggle[1]. There are two parts to the competition, the first part is the prediction of sales of 3049 products sold by Walmart; the other part is about estimating the uncertainty distribution of those predictions. We focus on the prediction of the sales part of the competition[2].

To evaluate a prediction of all products, the Weighted Root Mean Squared Scaled Error (WRMSSE) is computed. The WRMSSE is defined by the following function:

$$WRMSSE = \sum_{i=1}^{42840} w_i \times \sqrt{\frac{1}{h} \frac{\sum_{t=n+1}^{n+h} (Y_t - \hat{Y}_t)^2}{\frac{1}{n-1} \sum_{t=2}^{n} (Y_t - Y_{t-1})^2}}$$

2 Methods

The approach to this competition consisted of multiple phases. First, a variety of forecast methods was explored in order to examine their potential and learn what does and what does not work. Afterward, the method yielding the most-promising results was picked, which turned out to be LightGBM[3]. Lastly, after a baseline configuration was decided on, the model was improved by utilization of feature engineering, parameter optimization, and other tweaks, which are to be discussed in the following sections.

Additionally, the release of the full training labels on June 1[4] practically marked the end of the efforts presented in this report (after a new final submission was made), because of a set course deadline. Nonetheless, the competition continues until June 30, up to which it is still possible to improve the results.

2.1 Data

The available dataset involved the unit sales of 3,049 products, classified in 3 product categories (Hobbies, Foods, and Household) and 7 product departments, in which the above-mentioned categories were disaggregated. The products were sold across ten stores, located in three states (CA, TX, and WI) [1]. The historical data accounted for 1,941 days / 5.4 years (2011-01-29 to 2016-06-19), from which the last 28 days were not included, and instead, they were used for the purposes of the test data.

Downcasting Due to a large amount of data and the limited computing resources, it was advisable to first reduce the size of storage used by it. This was done by using a "popular" function `reduce_mem_usage` appearing across many notebooks. Read Appendix A to see its impact.

Data Melting Another useful function during the data preparation phase was `melt_and_merge`, which first transformed the sales dataframe from a wide format into a long format and then combined three input data files into one dataframe, which comprises of 60,034,810 rows (30490 unique values of the sales information $\times$ 1969 days) [4].

[1] https://www.kaggle.com/
[2] https://www.kaggle.com/c/m5-forecasting-accuracy
[3] https://lightgbm.readthedocs.io/en/latest/
[4] https://www.kaggle.com/c/m5-forecasting-accuracy/overview/timeline

2.2 Feature Engineering

Feature engineering constituted a major part of the work. The baseline version contained only 11 features, all representing just columns of the input data. In contrast, the final version had 28 features including various lags, rolling means or e.g. skewness and kurtosis. See Appendix B for a complete list containing all of the 28 used features.

Most of the features were chosen based on a common sense, exploratory data analysis, and the competition's trends, such as [5], or [6]. However, some of them were purely experimentation and driven by intuition, recognizing that the model is partially a black box, which is not fully understood.

Label encoding The goal of label encoding was to encode categorical features and simultaneously replace cells containing undefined values. Also, rows with an unknown value of a `sell_price` were dropped, which was crucial for further processing.

Lags Lags (also known as shifts) account for values at prior time steps and are the classical way that time series forecasting problems are transformed into supervised learning problems [4]. Lags along with rolling means were arguably the key features, that helped to achieve significantly better scores. In terms of the final feature set reported in this paper - lags of 28, 29, and 30 days were used.

Rolling means Rolling means help to determine the average demand for products in a given time-span. They are a great tell of "trends", often used in prediction-related challenges. Here, it was weekly, monthly, quarterly, and half-yearly rolling means that were eventually used.

Skewness and kurtosis Skewness and kurtosis are useful descriptive statistics of data distribution, therefore, it was common sense to incorporate these. These features were implemented as `rolling_skew_t30` and `rolling_kurt_t30`.

Additionally, it is important to say more features had been experimented with, such as some advanced time or rolling pricing features. They were not discussed further here, as they are not part of the final set. However, some attention is given to them in Section 4.

Feature enhancement Related to the idea of feature engineering is feature enhancement, which was considered to be worth investigating. The `event_name_1` feature was chosen and it was hypothesized, that it would be beneficial to mark the day before each event as "special" since it indicated the next day's importance. Impact of this objective is further mentioned in Section 3.

2.3 Model and prediction

During the first phase of the project, when examining feasible methods for the competition, many data forecasting models were tested: Average approach[5], LSTM[6], Prophet[7] and LightGBM. Eventually, LightGBM was chosen because of its supreme performance compared to the other methods, its overall fit to the data, and

[5] https://en.wikipedia.org/wiki/Forecasting#Average_approach
[6] https://en.wikipedia.org/wiki/Long_short-term_memory
[7] https://facebook.github.io/prophet/docs/quick_start.html

available computing resources, as well as of its general popularity among the highest-scoring entries in the competition.

LightGBM was relatively simple to implement thanks to the LightGBM Python-package[8]. However, the main challenge is generally parameter tuning, as the model covers more than 100 parameters. Therefore, many different configurations were tested, usually inspired by public notebooks[9] on the Kaggle platform to avoid "reinventing the wheel" and instead, channel the efforts towards the feature engineering, which was decided to be of higher importance. Table 1 shows the configuration of the final model, which was also used in one of the most popular notebooks[10]. Training data comprised of 28 features, as discussed in Section 2.2, whereas the validation set encompassed a single feature representing the number of a given item sold.

In terms of prediction, it was done for the last 28 days (`d_1914 - d_1941`), combining data from all stores and locations. Finally, for the submission purposes, the dataframe format had to be transformed again using the `pivot` function.

Parameter	Value
objective	poisson
metric	rmse
force_row_wise	True
learning_rate	0.075
sub_row	0.75
bagging_freq	1
lambda_l2	0.1
nthread	4
verbosity	1
num_iterations	1200
num_leaves	128
min_data_in_leaf	100

Table 1: The parameter configuration of the final model.

2.4 Evaluation

Once submitted to the leaderboard, the evaluation was computed using the WRMSSE formula. However, locally, the Root Mean Square Error (RMSE) was used as a proxy. It is known, that there's not a 100% match between WRMSSE and RMSE, but there's a strong correlation between these two metrics. The use of RMSE may cause overfitting though, which is a common problem in the Kaggle competitions. Therefore, it was important not to tweak the model just for the reason of lowering the local error, but instead, consider the metric more as a guide in general.

Lastly, after the release of the full training labels, it was not practically possible to use the public leaderboard for submissions anymore that would give objective ranking. Therefore, the last entry efforts during our project were done utilizing a notebook[11], that could still compute a relevant public leaderboard score with the newly-issued training labels.

3 Results

In total, 43 submissions were done. 36 of those where successful submissions, as 7 submissions contained an error. Many of the explored approaches did not give satisfying results. Eventually, the best-performing model, which was based on LightGBM, reached the WRMSSE score of 0.544.

As mentioned in Section 2.3, a variety of models was explored. The best-obtained scores for each of these is can be seen in Table 2. Average approach and LSTM did not yield satisfactory results. One can notice though that Prophet performed relatively well but since it did not offer any clear way for improvement (more to be discussed in Section 4), it was abandoned and the efforts were directed towards LightGBM.

LightGBM and its score improvement over a variety of versions can be looked at in Table 3. The first (baseline) version of the model with a score of 1.402 was not particularly exciting, yet it was rapidly improved on, soon reaching WRMSSE under 0.6. From then on though, not much improvement was made, rendering version 18

[8] https://lightgbm.readthedocs.io/en/latest/Python-Intro.html
[9] https://www.kaggle.com/c/m5-forecasting-accuracy/notebooks
[10] https://www.kaggle.com/kneroma/m5-first-public-notebook-under-0-50
[11] https://www.kaggle.com/rohanrao/m5-how-to-get-your-public-lb-score-rank

Model	WRMSSE
Average approach	1.691
LSTM	1.283
Prophet	0.777
LightGBM	**0.544**

Table 2: Best achieved scores for the explored models.

to be the best-performing one. The latter attempts only lead to worse outcomes, such as the experiment with the event feature enhancement. Comparison of the baseline LightGBM versus its best variation is presented in Figure 1. The visualization clearly shows a large improvement in prediction performance. The difference in their RMSSE with regards to categories can be seen in Appendix C.

The source code of the final version as well as of a notebook for used for the objective evaluation after June 1 is linked in Appendix D.

Version	WRMSSE
1 (baseline)	1.402
3	1.324
5	0.811
7	0.574
8	0.559
18	**0.544**
19	0.620
26	0.560

Table 3: Improvements on the LightGBM.

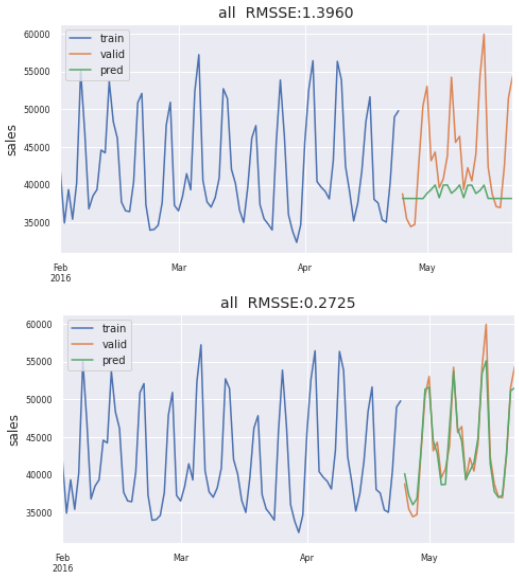

Figure 1: Prediction of the baseline LightGBM (above) versus its best-performing version (below).

4 Discussion

The final score of 0.544 made for position 2126 out of 4952 teams on Kaggle at the time of writing. The results could be still improved though by different means.

First, feature engineering is what separates a good LightGBM model form a bad one, therefore a lot of time went into trying different combinations of features. For instance, adding some price-based rolling means or advanced time features resulted in lower accuracy, which was surprising. This could mean that either the best scoring model was overfitting on the test set or that the implementation just was not applicable to the model. The latter is due to the model being a black box, i.e. the exact mapping of features to score is not clear. This causes features, that might intuitively seem appropriate to humans, to sometimes yield counter-intuitive results.

Next, parameter configuration would be something to look into more in-depth. It is true that during this project, multiple versions were tested, but as also mentioned in Section 2.3, having more than 100 different parameters to play with gives one a lot of variations to try.

Another way to possibly improve the final score would be to create a separate model for each store. This approach yields good results on the leaderboard and seems intuitive since stores in different locations have different costumers with different shopping habits. However, due to time restrictions, this approach was not implemented.

Regarding forecast methods, it could be worth taking advantage of Catboost[12], which is a technique that improves accuracy at the cost of speed. It works the best when there are many categorical features available[3], although, in the Walmart dataset, not many of those are present. Therefore, implementing Catboost would probably not yield better results, yet, this is something to be tested.

Last but not least, there is an idea of combining LightGBM with Prophet. This has been done before [2], but again, this would need to be further experimented with in order to see its performance in this particular competition.

5 Conclusion

The final best score on the public leaderboard of 0.544 was achieved by a notebook based on LightGBM. During the process, a number of other models was experimented with, which, however, did not result in satisfactory scores. The most-successful version at first prepared the input files using downcasting and data melting. Later on, with the use of feature engineering, a total of 28 features was created, which served as an input to the LightGBM's training data. Lastly, the model's parameters were set appropriately, the network was trained, eventually predicting the number of items sold for the unknown 28 days. Further improvement of the score can still be achieved since the competition is still running at the time of writing.

References

[1] The M Open Forecasting Center. The m5 competition. https://mofc.unic.ac.cy/m5-competition/, 2020.

[2] W. Fang, P. Lan, W. Lin, H. Chang, H. Chang, and Y. Wang. Combine facebook prophet and lstm with bpnn forecasting financial markets: the morgan taiwan index. In *2019 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)*, pages 1–2, 2019.

[3] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. https://arxiv.org/pdf/1706.09516.pdf, 2017.

[4] Anshul Sharma. M5 forecasting-eda, fe modelling. https://www.kaggle.com/anshuls235/m5-forecasting-eda-fe-modelling, 2020.

[5] Bojan Tunguz. Best features only. https://www.kaggle.com/tunguz/best-features-only, 2020.

[6] Konstantin Yakovlev. Few thoughts about m5 competition. https://www.kaggle.com/c/m5-forecasting-accuracy/discussion/138881, 2020.

6 Author contributions

Jan	Koen	David
LSTM	Average approach	model from scratch
LightGBM	Prophet	feature enhancement
FE	FE	FE
project lead	dataframes	visualisation

[12] https://catboost.ai/

7 Evaluation of the process

Thanks to lessons learned during the first competition, the start of this one was smoother. It was possible to begin working on the code right away without the need of learning about Jupyter notebooks etc.

Consequently, this created the possibility to think about a good strategy, which again consisted of two techniques. The first one was a bottom-up approach and work from scratch. The second method was a top-down approach, using existing notebooks to learn on a high level. Disappointingly for us though, the distance between low-level theory and the high-level of knowledge required by the competition was too large to bridge in the given time. Therefore, it was decided to use the latter procedure and work mostly on a high level, using existing notebooks, discussions, trial and error, and additional background reading resources. We were simply not able to program a fully-working solution yielding better scores just by ourselves. The final solution still contains bits of our coding efforts though.

COVID-19 made it impossible to meet physically, however, by using online voice communication tools, it was viable to maintain our weekly meetings. This was important because we could keep a steady pace and keep up the gradual progress on the project. And also time to time, some of the methods or models did not lead to productive results, meaning a change of plans had to be discussed.

All things considered, this second competition, we think, resulted better for us than the first one, where we were just more-or-less lost. And that makes us quite content.

8 Evaluation of the supervision

All of the meetings with our supervisor Chris as well as with the teachers were valuable. We were given advice on the strategy of how to approach the project, we received many ideas regarding potential enhancements of our methods and models and last but not least, we were helped to avoid possible pitfalls and dead-ends.

Especially the last meeting with Chris was very beneficial because, at that point, we were improving our scores bit by bit, submitting "one notebook after another". After Chris heard this, he warned us about the possibility of overfitting and that it is not advisable to "game the leaderboard" like that. This was a wake-up call for us. Even though we knew the theory of overfitting, we did not realize our strategy would do just that. The meeting made us more aware of this fact and forced us to re-think our strategy. Afterward, we would mostly submit only versions that promised more radical changes in our score.

And finally, two more reasons why the supervision was useful. First, it was good to know that the project is well on track, and second, it is arguable that the supervision kept us more accountable, and thus, we learned more which consequently lead to a better result.

A Downcasting

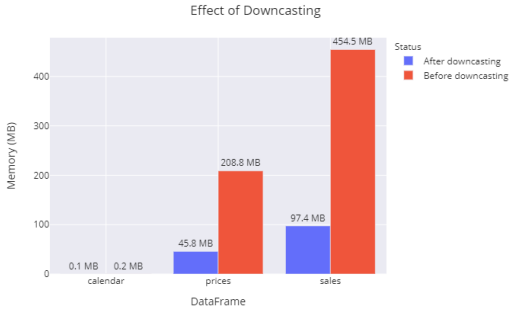

Figure 2: Downcasting effect of the input data[4].

B Features

The following is the list of the features used in the final version of the submission:

1. item_id
2. dept_id
3. cat_id
4. store_id
5. state_id
6. wm_yr_wk
7. weekday
8. wday
9. month
10. year
11. event_name_1
12. event_type_1
13. event_name_2
14. event_type_2
15. snap_CA
16. snap_TX
17. snap_WI
18. sell_price
19. lag_t28
20. lag_t7
21. lag_t1
22. rolling_mean_t7
23. rolling_mean_t30
24. rolling_mean_t60
25. rolling_mean_t90
26. rolling_mean_t180
27. rolling_skew_t30
28. rolling_kurt_t30

C RMSSE comparison

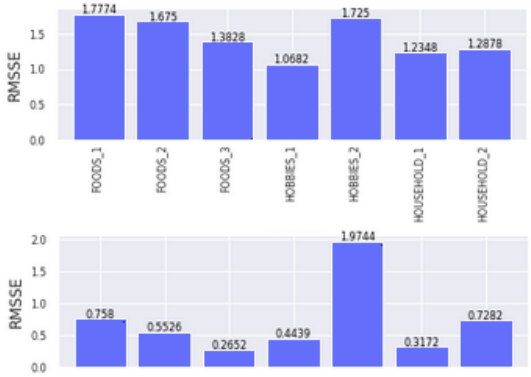

Figure 3: RMSSE per category, with the baseline RMSSE above and the RMSSE of the final notebook below.

D Source code

The source code can be found on GitHub.