Network Traffic Scenario Prediction

Oliver Hennhöfer (*alias* Olleknolle)
oliver.hennhoefer@mail.de

01.10.2023

Abstract

This writeup describes the 7*th* place solution of the *Network Traffic Scenario Prediction Challenge* held on *zindi*. The solution is based on LightGBM and feature engineering with the goal of testing the boundaries of gradient-boosted decision trees (GBDT) with respect to tasks comprising large amounts of autocorrelated time-series data, like the network traffic flow data provided as part of the challenge.

1 Introductory words on my own behalf

I joined the competition rather late in mid-September and, with a total of eight submissions, it was more of a playful participation on my site. At the time, I was on vacation and had just looked into the possibilities of the *polars* [1] python package – a faster *pandas* alternative implemented in Rust. Long story short, I found the **Network Traffic Scenario Prediction Challenge** to be the perfect playground to start learning the application of *polars* with a data set, too large to be effectively handled by *pandas*.
With this goal and only a limited time at my hands, my actual solution defaulted to LightGBM [2] and some engineering of *rolling features*. I worked with LightGBM many times in the past, so it was easiest to just rely on something as familiar to me as this particular GBDT implementation that can conveniently handle larger amounts of data.

2 Methodology

Besides the rather self-explanatory application of LightGBM (*goss*), the Optuna framework [3] was used for tuning relevant hyperparameters (*see code*). **The core of the solution is based on the engineering of different rolling window features that try to incorporate the temporal aspects of the data** into the otherwise independent row-wise [learning] nature of a GBDT, like LightGBM (in contrast to e.g. LSTMs, ...).

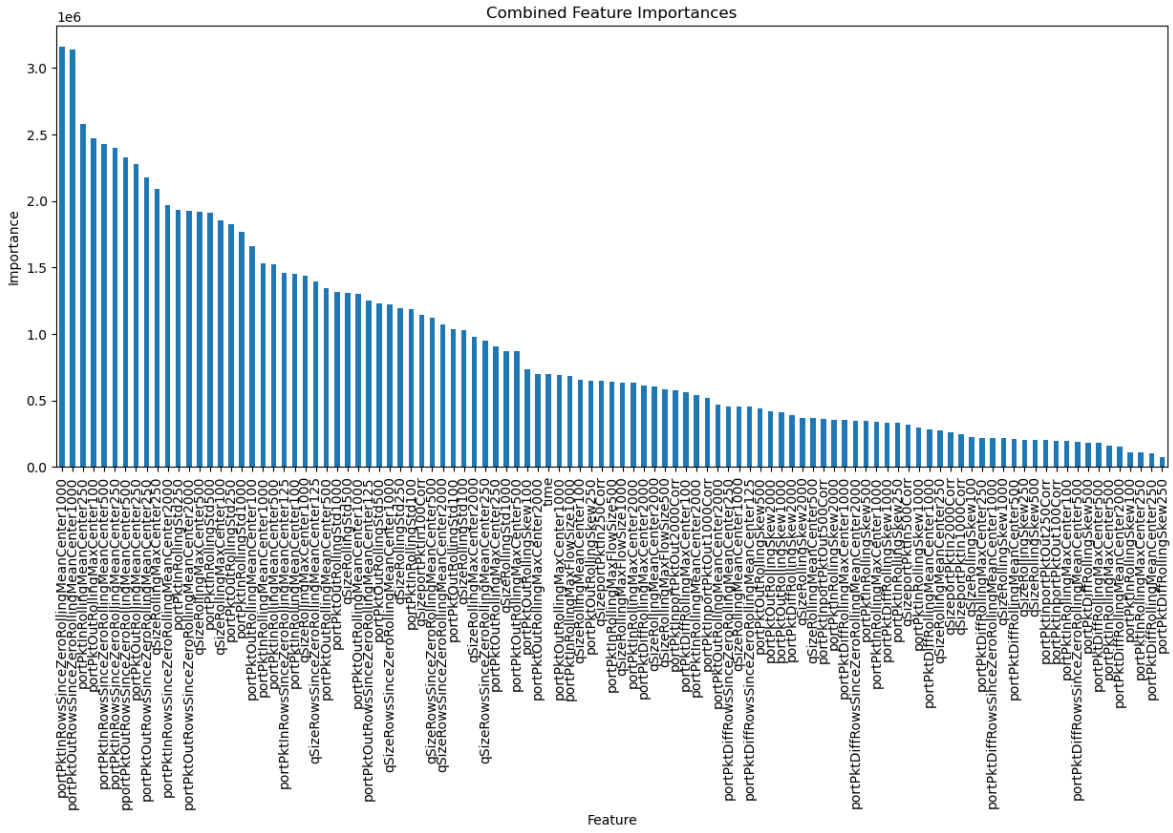
Following features were created based on the original data – identical rolling features were created separately for every distinct training file (.csv) and over different window sizes[1]:

[1] Besides the original features, some engineered features are based on the variable *portPktDiff* which in turn was calculated as $portPktIn - portPktOut$.

Feature Name	**Variable**	**Description of Window Function**	**Window Size(s)**
RowsSinceZeroRollingMeanCenter	portPktIn, portPktOut, portPktDiff, qSize	Mean amount of rows since last zero value	125, 250, 500, 1000, 2000
RollingMeanCenter	portPktIn, portPktOut, portPktDiff, qSize	Simple rolling mean	100, 250, 500, 1000, 2000
RollingMaxCenter	portPktIn, portPktOut, portPktDiff, qSize	Simple rolling maximum	100, 250, 500, 1000, 2000
RollingSkew	portPktIn, portPktOut, portPktDiff, qSize	Rolling skewness	100, 250, 500, 1000, 2000
RollingCorr	qSize/portPktIn, portPktIn/portPktOut	Rolling (Pearson) correlation	100, 250, 500, 1000, 2000
RollingStd	portPktIn, portPktOut, qSize	Rolling standard deviation	100, 250, 500, 1000
RollingMaxFlowSize	portPktIn, qSize	Maximum amount of consecutive non-zero values	500, 1000

The original features (*qSize, portPktIn, portPktOut, portPktOut, portPktDiff*) were dropped for the final model training.

The hyperparameter tuning was conducted on only a subsample (taken scenario-wise) of the whole data set for faster computation. Naturally, the rolling window features were calculated upfront on the entirety of available training data, so before the splits. Hyperparameter tuning trials were conducted via 5-fold cross-validation by the means of a *GroupShuffleSplit* (again, *group* $\triangleq$ *scenario*). During hyperparameter tuning following feature importances (*gain*) were observed[2]:



As the final postprocessing step, the predictions were *smoothed* by applying a *rolling (center) mode* window function with a size of 300 by group/scenario. Since the class appearances are auto-correlated, this postprocessing step should account for (minor) deviations with respect to the predicted class.

3 Final Remarks

The solution provided is treating the problem as a simple tabular data task, so it's neither exceptionally innovative nor shows particular originality. However, with regard to the other top solutions using deep learning, it shows the capability of one of my favorite machine learning algorithms – gradient-boosted decision trees. Either way, it was a great competition experience overall (and a great motivation to start learning *polars*), especially with respect to reading through all the (arguably more elegant) solutions of the other participants.

Nonetheless, there might be some value provided, by incorporating the engineered features (with potentially more *signal* than the original data) into the deep learning-based solutions – at least where applicable.

[2]The importances were summed up for the five models trained during cross-validation for the best tuning trial.

References

[1] Ritchie Vink, Stijn de Gooijer, Alexander Beedie, Marco Edward Gorelli, J van Zundert, Gert Hulselmans, Cory Grinstead, Matteo Santamaria, Weijie Guo, Daniël Heres, Josh Magarick, Marshall, ibENPC, Orson Peters, Jorge Leitao, Moritz Wilksch, Marc van Heerden, Oliver Borchert, Colin Jermain, Jonas Haag, Joshua Peek, Ryan Russell, Chris Pryer, Adrián Gallego Castellanos, Jeremy Goh, illumination k, Liam Brannigan, Max Conradt, and Robert. pola-rs/polars: Python polars 0.19.0, August 2023.

[2] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017.

[3] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.