REVISITING ESTIMATED TIME OF ARRIVAL MODELS

Dániel Szokolics @ **Hiflylabs**
CEU Business Analytics
2019-2020

1. PROJECT DESCRIPTION

Our client delivers pharmaceuticals to Hungarian pharmacies on a daily basis, and we built an application for them which predicts the estimated time of arrival (ETA) of these goods. This application aims at enhancing customer satisfaction as the pharmacies can tell their clients when to come back for their drugs, and they can adjust their daily schedules accordingly. Before the application goes live, our client asked for an update in the prediction process in order to decrease the number of predictions with large errors. My capstone project was this prediction enhancement.

Our client's main request was to decrease the rate of large errors among the predictions (errors above 30 minutes). These errors added up to 15% of the total errors before the COVID-19 pandemic, and 25% during lockdown period. During the analysis I focused on the pre-pandemic data, as I expected that the situation will get back to normal. There were two options to reduce these errors:

1. Fine-tune the underlying models
2. Drop some of the predictions which seem risky in advance

The tasks were broken up between me and one of my junior colleagues. As part of the project, I had to supervise his job, and enhance his professional advancement. The parts which were done by him, will be noted (and some, smaller side tasks will be omitted).

2. THE UNDERLYING SYSTEM

The ETA prediction engine has multiple parts, and I will describe only the part which is related to my capstone. This is the part that does the prediction[1].

We use pre-trained models to make real time predictions based on the incoming data. We receive a small pocket of data when the van is loaded (start data) and when the unloading happens (arrival data). We call one full loading and unloading cycle a sprint. The predictions are given within one minute after the start data arrives, and the predictions are done in the following manner:

1. Predict the time between the arrival of the start data, and the actual start of the van.
2. Predict the move time between the site and the first client.
3. Predict the time spent at the first client. The prediction which is shown to the pharmacies is in the middle of the stop.
4. Predict the move time between the first and the second client.
5. And so on, until the end of the sprint.

In order to make our predictions more stable, the system uses a validation method. Based on the historical arrival times for each client, it is checked if the predicted arrival time is within the highest

[1] The others are responsible for maintaining the database, collecting and loading new data, training the models and monitoring performance

density interval (30 or 60 minutes, with at least 70% of the historical arrival times). If the prediction falls into this interval, we leave it as is, but if it is outside of the interval, it gets modified.

3. MODEL UPGRADE

There were two reasons why the underlying models' performance could be increased. 1) There was far less data during the original calibration. 2) The original parametrization was very conservative[2]. We had three models to upgrade:

1. start model: predicting the time difference between the start signal and the real start time
2. move model: predicting the time spent on travelling between pharmacies
3. stop model: predicting the time spent at pharmacies

The first two were done by my colleague. On average, all the updated models outperformed the original ones by 5-10%.

However, the individual model performance is not the main interest; they are only parts of the arrival time predictions. After creating the arrival time predictions (along with the validation step), the results are close to the old model's results. The new model has better performance for the 900- and 1200-seconds thresholds, but for the others, the results are very similar (in the case of RMSE, the old model is a bit better).

	new model	old model
900	**69.58 %**	68.68 %
1800	**88.95 %**	88.63 %
3600	96.74 %	**96.79 %**
RMSE	1486.77	**1451.19**

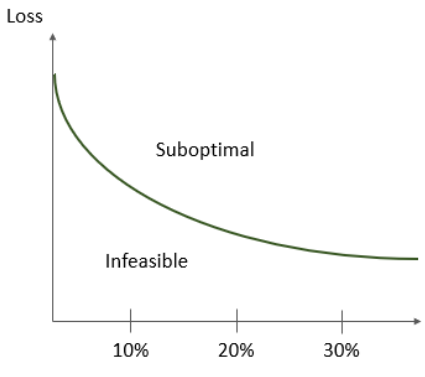
In the end, the old model was kept, because of the following reasons:

1. The new model's performance is not clearly better.
2. The new model is much more complex, which makes me think that it is less stable.
3. The old model proved to be stable, and changing it entails risks.

Although the model updates did not bring about any performance gains, some enhancements could be introduced by fixing bugs and introducing cleaning steps to the data. For example, some of the pharmacies had Polish coordinates in the input data, which is obviously incorrect.

4. ERROR PREDICTION

The other part of my project was the error prediction. Here the goal was to find the optimal tradeoffs between the quantity and the quality of our predictions by filtering out some of them. The optimal compromise is a compromise, where no more predictions can be added without lowering the performance of the predictions, and the performance can't be tuned without further filtering. In terms of RMSE, the frontier of these optimal tradeoffs can be seen on figure 2.



1. Optimal tradeoffs

The client is responsible for making the final tradeoff (e.g. choose the dropout rate).

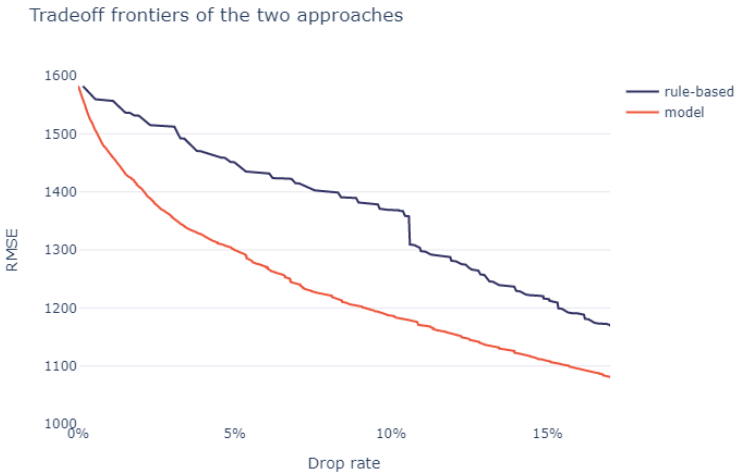
There were two different approaches to tackle this problem. The first was to take a deep-dive in the data and find those segments – mostly relying on business logic – which are riskier. This is the rule-based approach. The second was to build

[2] Random forest regressors that use only a few features and are only parametrized by maximum depth

a model to predict the expected absolute error of a prediction (ML-based approach). The first approach is clearly less efficient, but the findings in themselves have business value, and they are transparent. If the performance difference is not too big, then the one with the business rule set is preferred.

The rules were set up by examining the predictions going from the largest to the lowest, and attaching flags of the failure causes. Especially for the highest residuals, they were often very easy to find. As an example, one of these flags was the stop order prediction method[3]. If it was based on scarce data, it is rarely correct, hence the arrival time predictions will be worse.

The frontier of feasible tradeoffs was created from the different subsets of the ex-ante labels. In total I had 512 of these subsets (2^9). All the suboptimal tradeoffs (e.g. had a combination with less dismissed predictions and better performance) are excluded.

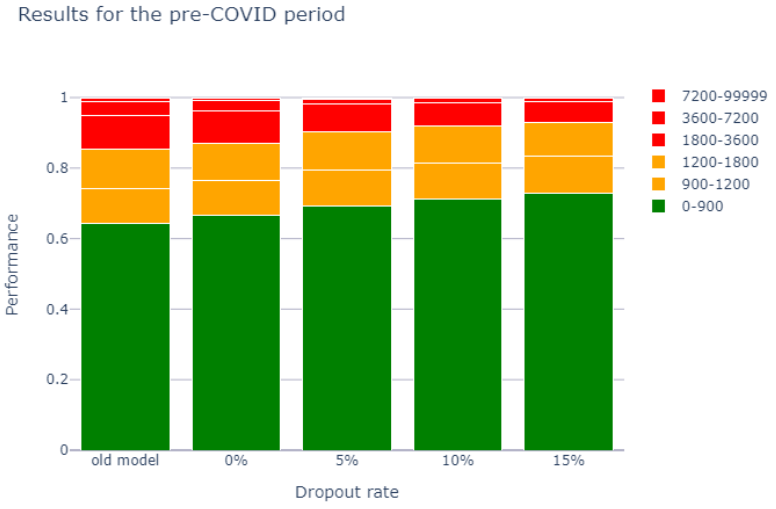


2. Comparison of the two frontiers

The other approach was to build a random forest regression for predicting the absolute error. It was quite straightforward: the absolute error was the predicted variable and all the ex-ante available features served as predictors. The model had an R2 of 41% on the test set, which is pretty impressive, and well above the expected.

The model version performed much better, the RMSE difference is between 100 and 200. Because of the superiority in performance, the model-based approach was implemented.

5. RESULTS



The effect of the dropout is significant. The colors of the bars correspond to the colors of the performance dashboard which the client uses. They formulated the project purpose as “reducing the amount of red predictions”. The proportion of the highest errors drop quickly, which is aligned with this purpose. The difference between the old model and the zero dropout rate is the data cleaning.

As it was mentioned in the beginning, the application’s goal is to increase customer satisfaction via showing the arrival times of the drugs. By exposing too many erroneous predictions to the clients, they will lose the trust in the

[3] We do not know the exact order of the pharmacies when the sprint starts, so we have to make predictions for them.

application, and they won't use it as frequently as they should. With this project we were able to show alternatives to our client to reduce these errors.

6. PERSONAL TAKEAWAYS

It was not emphasized during this summary, but I had to manage one of my colleague's work through the project. He was responsible for updating two models and he also did some side tasks, which were not part of the capstone. The most challenging part was to help him to gain knowledge and make good quality codes and models. In case of the modelling my approach was to let him work, and then give feedbacks, recommendations and readings on how improve his work further.

A very specific takeaway is that I thought of hyperparameter fine-tuning as something really important and impactful. However, it turned out that in this case, a really simple, and quite conservative parametrization can have almost as good performance as some really fine-tuned one.

Another useful thing is that I have to learn a bit of Docker in order to acquire the data from the system (this part of the app was developed originally by one of my former colleagues). Besides learning that, I found it really useful for model training. With Docker, you can run applications on your computer in a way that the resources used by it are separated from the other parts of your computer. In practice it means that you can train your models without freezing your laptop. It increased my productivity a lot.