A Smooth Approach To Putting Machine Learning Into Production by Max Halford

https://maxhalford.github.io/blog/a-smooth-approach-to-putting-machine-learning-into-production/ (https://maxhalford.github.io/blog/a-smooth-approach-to-putting-machine-learning-into-production/)

- ML in prod. is hard. And there are many products that try to automate ML production (Google AutoML, AWS Sagemaker, DataRobot).
- We are getting good at building and training extremely strong models. Average skill and knowledge is growing at a steady pace. We are converging towards standard practices and common patterns like https://developers.google.com/machine-learning/guides/rules-of-ml/ (https://developers.google.com/machine-learning/guides/rules-of-ml/)
- The way ML goes into production differs significantly from company to company but the general idea is to extract features from a dataset, train a model, and then put the model behind an API. For making a prediction for a new instance, features are generated in real-time before being fed to the model. Every once in a while a new model is trained and replaces the current one. It's pretty straightforward when you think about it, and is exactly what are doing all the services I mentioned above. Some people call this pattern the lambda architecture. However, down the road you might realize that is has a lot of kinks and things are no so smooth.

Here are a few pain points that often arise:

- The rate at which to retrain the model has to be decided upon. Should it be every day? How about every week? If it's every week, then should it take place every Sunday at 10PM or every Tuesday at 3AM?
- You have to wait until a new model is trained in order to exploit new data. In other words, your model isn't able to update itself every time a new observation arrives.
- Training models is going to require more and more computing power because you will have collected more and more data. You could argue that using only the data from, say, the past week is fine, but that's also something you have to choose.
- The features you used to train your model might be difficult to generate in real-time. For example, if you're using aggregate features then this means you need some way of storing them and retrieving them when needed. DataRobot and H2O handle some of this for you, but it is by no means a trivial task. Indeed the features are usually expected to be generated by the user beforehand.

https://github.com/creme-ml/lol-match-duration/ (https://github.com/creme-ml/lol-match-duration/)

In a lot of applications, data points arrive one by one. In other words, the data is a stream and not a dataframe.

The main idea is that most training is done in batch format and we got used to it. But in production environment re-training the model using batches often results in difficult or even impossible to maintain solutions. The solution is to use online learning and update the model every time you get a

new instance. Works really only for simple linear models (GBT's take long to re-train).

Your model can update itself and make predictions right inside an HTTP request.

An example of such an app is *League of Legends match duration predictor*.

- every time a user asks for the duration of a match, the app queries the League of Legends official API and retrieves the match information, if the match has already ended, nothing happens.
- then the raw match information is fed through the model and a predicted match duration is obtrained by calling mode.predict_one(match_info).
- every once in a while the app polls the API to check if the match has ended. Once the match has ended, the true duration is obtained and the model is updated by calling model.fin_one(match_info, match_duration). The ML part was the smallest part and the easiest.

Using scikit-learn and batch learning for this task would make life a lot harder.