Training and Storing System1's Time Series Models

Gergely Daróczi, Nathan Janos

Building and optimizing complex models is becoming relatively easy due to the growing set of tools available to data scientists. We have free (not only as in "free beer" but as in "free speech") access to open source software packages and tools like R, python, H20.ai, and TensorFlow. These tools can be used together with hyperparameter optimization methods, cross validation, integration tests to perform scoring operations for recommendation systems.

It then becomes fairly easy to use these tools in combination with a clean dataset and cheap computer power to build predictive models--even without understanding the mathematical or statistical foundation of the models, although this is also important for a number of reasons--using these tools as black boxes. But, in reality it turns out that collecting, joining and transforming the raw data required to make clean data sets for modeling is not a trivial task. And, it's not a small feat to deploy and monitor these models in production systems that make millions of decisions a day either. This is sometimes the reality of the nature of work that data scientists have to perform outside of their core modeling and R&D efforts.

This article describes how we train forecasting models on time series at System1, focusing on reproducibility, data standardization, model validation, logging, storing the actual models, and serving live recommendations.

Standardizing the Modeling Workflow

It seems that every data scientist has a go-to language or favorite method when it comes to specific modeling problems and goals, which can quickly lead to a very adaptable but extremely diverse environment that is unwieldy to integrate in production systems.

Some data-driven teams are structured by having the data scientists do the research and solve business problems with free choice of programming languages and other tools, which then become productionized by the engineers in a standardized way. Unfortunately, this approach isolates human resources and often creates slower and more disconnected product solutions and release cycles--not the best approach for most startup environments.

Instead, at System1 we execute quick rounds of research and generalize the results early on into frameworks that can easily be applied to similar business problems or products within the company. This saves a lot of time in the long run, not only because we can apply the exact same method on a new type of data with a single click of a button, but because this approach accumulates common knowledge available to all team members instead of compartmentalizing domain specific information and experiences.

As an example look at the TBATS (Exponential Smoothing State Space Model With Box-Cox Transformation, ARMA Errors, Trend And Seasonal Components) model we use to model hourly revenue-per-click time series. These time series are treated by removing the outliers based on some business-defined rules and filling in the data holes by applying a Kalman filter. The next day, we can easily apply the same framework for predicting the volume of ad clicks or other financial metrics without spending a lot of time on data preparations or quality checks on the time-series because we can leverage this prior work.

And if it turns out that the existing workflow does not work well for a new type of data, then the team can work on generalizing it further and making it more robust to assist in future efforts.

Standardizing the Model Output

Standardizing the workflow does not limit us to using the very same model on all types of data; the workflow can also automatically decide the optimal model for the dataset or business goal. For example, intraday seasonality might be a very important factor for some financial metrics, while other models will better integrate holiday effects. So technically speaking we might want to use the *prophet* instead of the *forecast* R package for predicting the number of ad-clicks around Thanksgiving.

Of course, we could have also used a neural networks with a long short-term memory model, or TensorFlow instead of R for building the model, so it's important to specify a standard model output format that is independent from the actual model or computing environment.

Without diving too deeply in the technical details, we solved this problem by storing the models as JSON objects in Amazon S3 following a folder hierarchy describing the nature of the data and method. In addition to this we store the models in multiple files with some redundancy to serve the needs of explorative dashboards, batch processing and live recommendations. These objects include the raw and language-specific models for future predictions, but there are plenty of other standardized fields stored as well. For example, the raw and cleansed time-series data, the model parameters, the training log, and short-term future predictions.

Advantages of a Standardized Modeling Workflow

Being able to build a forecasting model on any time series in a standardized way has a number of great advantages over training ad-hoc models for specific needs:

- Build new models (on new metrics) in a few hours
- Automatically deploy the new models into production (for live or batch forecasting)
- Never commit the same mistake (like low-volume data points obscuring seasonality)
- Easily develop and then reuse models

- Share dashboards and explorative tools with other stakeholders to investigate model parameters and forecasts
- Storing all model versions in a standardized way also allows rolling back to previous versions and backtesting variants in a standardized way