Highlights

A Platform for Assessing and Combining Autonomous Short-Horizon Distributional Predictions

Peter Cotton

- We exhibit an open-source contest platform embodying many of the recommendations made recently by experts.
- The platform suggests a transformative "prediction web" whose ambitions extend far beyond the goals of prediction competitions.

A Platform for Assessing and Combining Autonomous Short-Horizon Distributional Predictions

Peter Cotton[a]

[a]*Intech Invesments, 1 Palmer Square, Suite 441, Princeton, 08542, NJ, United States*

Abstract

The operation of a novel open-source platform where mostly autonomous algorithms are tasked with predicting a large variety of streaming data is described. Reward and combination is achieved by means of a near-the-pin mechanism generalizing lottery countbacks to continuous space.

Keywords: nowcasting, time-series
PACS: 0000, 1111
2000 MSC: 0000, 1111

1. Motivation

Bespoke quantitative optimization lies largely in the province of large corporations who can afford to hire teams of data scientists. An experimental open-source platform hosted at Microprediction.Org is described herein, and seeks to assess the feasibility of an alternative approach, in the hope of delivering very low cost bespoke data science.

Under this paradigm, companies abstract the open-ended, ongoing, costly aspects of intelligent system development (the never-ending search for better models and relevant exogenous data to make predictions more accurate) into a collection of well defined repeated prediction tasks.

Exposing those tasks is essentially synonymous with repeated publication of a live number by means of an API or client library. This allows external algorithms to solve the problem of microprediction at low cost, especially if the marginal cost for the algorithm and algorithm designer of addressing a new problem is small.

The platform can also be used to hold one-off contests. The importance of contests and standardized benchmarking needs little introduction - see discussion in [1] for example. In the context of time-series modeling specifically, there are challenges arising when only historical data, rather than live data is used. Those include the possibility of data leakage ([2], [3]); the possibility that success on one contest does not translate to others; and the difficulty of interpreting point estimate forecasts.

Needless to say the use of live data ameliorates data leakage. But also, in other setups a huge burden placed on the creator of the contest in some cases (gathering all relevant exogenous data).

The platform described herein also makes it trivial for anyone to add new sources of live data, thereby encouraging a variety of challenges which might hope improve the study of generalization ability of algorithms, and autonomous estimation.

A strong motivation for probabilistic predictions, as compared with point estimates, is that submissions can convey more information. The location of a badminton player's neck as he lunges about the court makes the point - that is one example of a stream on the microprediction platform, discussed in [4].

Some design principles for contests were laid out by Makridakis, Fry, Petropoulos and Spiliotis in [5] and discussed in [6]. The conclusions reached therein can be summarized as a desire for a *diverse* set of *live*, *distributional* prediction challenges. Also noted by the authors is the increasing importance of short-horizon prediction to industry.

The platform described herein also provides numerous advantages over prior approaches, such as the use of live prediction point-estimate contests described in [7].

It is important that anyone can host a version of the platform described, and that if there are causal relationships between data streams on one host and another it is relatively easy for reward-seeking algorithms monitoring both to exploit this.

Thereby, this may eventually facilitate a web-scale ambition we might term a "prediction web" - to harvest commonality of live feature spaces among diverse applications and drive the marginal cost of nowcasting to extremely low levels.

2. Streams and crawlers

The particular example leans on just a few concepts, which I enumerate.

A *stream* is simply a time series of scalar (float) data created by someone who repeatedly publishes a single number. It becomes a public, live moving target for community contributed prediction algorithms.

A *crawler* is a program, usually derived from a MicroCrawler in the Python microprediction package [8], that monitors one or more streams and decides whether to participate in the competitive forecasting of the same. An author of a crawler is free to modify the navigation and economic decisions made by the crawler as they see fit.

Should the crawler decide to do so, it will engage with a data stream by choosing a prediction horizon. Then, it will submit once or many times distributional forecasts. The forecast takes the form of a vector of 225 floating point numbers. Those 225 numbers are suggestive of the distribution of values that will be taken by a data point at some time in the future...say 5 minutes from now or 1 hour from now.

3. Quarantine

At the time when a crawler decides to provide a forecast, it does not know the arrival time of the ground truth it will be judged against.

The distributional prediction applies not to a fixed time horizon but rather to the time of arrival of the first data point *after* some elapsed interval.

Let us pick a delay of 3555 seconds for illustration (45 seconds shy of one hour). If the data seems to be arriving once every 90 minutes, and arrived most recently at noon, it is fair to say that a set of scenarios submitted at 12:15pm can be interpreted as a collection of equally weighted scenarios for the value that will be (probably) be revealed at 1:30pm (and is thus a 75 minute ahead forecast, loosely).

The mechanism performing rewarding and combination doesn't care about the interpretation. When a new data point arrives at 1:34pm, it looks for all predictions that were submitted at least as far back as 12:34:45pm, a cut-off point chosen to be 3555 seconds prior. Those distributional predictions qualify to be included in a reward calculation.

4. Near-the-pin reward

Each algorithm will be scored based on how many of its 225 guesses are close to the revealed truth. In a manner similar to lottery prize sharing, or the parimutuel system, net rewards across all participants are zero for each arriving data point.

This mechanism is called a near-the-pin collider. It may be seen as an incremental reward system triggered by a collision between arriving ground truth and quarantined distributional predictions.

As a detail, a small amount of noise is added to predictions and the ground truth.

There are obvious modifications, not presently adopted. Submissions could take the form of weighted samples rather than unweighted samples. The reward mechanism could employ a less abrupt kernel, and so forth.

5. Implementation

Soliciting distribution estimates in this manner may seem like a lot of extra traffic, compared to live point estimates. However, clearing is incremental and fast. An additional compensating advantage is that the lottery style reward mechanism is stateless.

The current implementation optimizes the response time to an incoming ground truth (a value in a stream), as compared with optimizing the handling of an incoming prediction vector. The implementation constructs pipelined operations implemented on a redis cache.

All details are available in the collection of microprediction Github repositories [8].

6. Pseudo-Oracle

The collider serves as a pseudo-oracle for distributional time-series prediction at fixed horizons, in the sense that the creation of a stream begins the process of luring reward seeking algorithms.

After a few hours, or days, or weeks depending on the data publishing frequency, this leads to community distributional forecast at various horizons. Those horizons are, as noted, roughly 1 minute ahead, 5 minutes ahead, 15 minutes ahead and 1 hour ahead.

The collective result is summarized as a collection of four community generated cumulative distribution functions (CDFs). The bad algorithms give up or get kicked out, and better ones arrive. The CDF gets more accurate over time as algorithms (and people) find relevant exogenous data or fine-tune the use of the data they already see.

7. Creative use

As an aside, one might publish once a day. Then, as a practical matter, one receives a lot of day ahead predictions as many of the algorithms make their submissions soon after a data point is received.

Another pattern involves publishing changes to the predictions made by an in-house algorithm. Those predictions might supposedly be martingales, for example, but publication can help discern the veracity of that property, or not.

8. Recursion and z-streams

The near-the-pin collider uses itself recursively in order to facilitate anomaly detection, specialization, monitoring and other objectives.

The community implied CDF implies a percentile for each arriving data point. Let's suppose it has surprised the algorithms on the high side and so the percentile is 0.72 say. We call 0.72 the community implied percentile.

A community implied percentile must be defined relative to some choice of quarantine period (i.e. forecast horizon). For example the data point might be a big surprise relative to one hour ahead prediction, but less so compared to forecasts that have not been quarantined as long. The reverse can also be true.

As it happens there are only two community percentiles computed at the time of writing: one computed using forecasts delayed more than a minute (actually 70 seconds) and one relative to those delayed by almost one hour or more (actually 3555 seconds).

Next, we define a community z-score as the inverse normal cumulative distribution of the community implied percentile. We are using the community to define a distributional transform - then transforming to normal (which most time-series algorithms prefer). If the community of human and artificial life is good at making distributional predictions, the z-scores will be normally distributed.

The use of the terminology z-score is an overloading - yet another example of my inability to solve the hardest problem in computer science (naming things). Of course in statistics, z-scores often refer to a different standardization of data that assumes it is normally distributed. (Invariably that isn't the case, and thus the collider z-scores can be considered an attempt to improve on this practice.)

Finally, the z-scores are fed back to the collider, so they can themselves be predicted. Think of it as automated model review.

9. Multivariate residuals and copulas

The near-the-pin collider is actually a little more elaborate than this suggests, because it also attempts surveillance of two and three dimensional implied community percentiles.

Space filling curves fold combinations of community implied percentiles back into univariate streams, so they also can be fed back to the collider.

As a mildly technical aside, algorithms tasked with predicting these so-called z2 and z3 streams might be seen as estimators of market-implied copulas. This gives rise to an extremely fine-grained understanding of the relationships between variables, facilitated by specialization (since the algorithms predicting the margins need not be the same as those predicting the copulas).

10. Making Python crawlers

Getting down to nuts and bolts, the conversion of a time-series algorithm into a micro-manager comprises the following steps.

1. Sub-classing a provided Python class.
2. Modifying the prediction logic as needed.
3. Modifying the default logic that navigates to data streams.
4. Running it.

In this example the default navigation ability amounts to a random selection of a stream. The default economic logic is a stop-loss. The micro-manager will give up on the task of predicting a data stream if it loses too many credits. Many, but not all crawlers exploit an open-source Python package that is intended to make it easier to create and assess autonomous time-series prediction algorithms [9].

This is rather simple and anyone can improve it. Yet even these minimal non-suicidal economic tendencies can be enough for a crawler. It travels from game to game, learning where to give up, and eventually making a lasting contribution - hopefully.

References

[1] D. Donoho, 50 Years of Data Science, R Software (2015) 41.

[2] S. Kaufman, S. Rosset, Leakage in data mining: Formulation, detection, and avoidance, KDD '11 Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining (2012) 556–563doi:10.1145/2020408.2020496.
URL `http://dl.acm.org/citation.cfm?id=2382579`

[3] A. Narayanan, E. Shi, B. I. Rubinstein, Link prediction by de-anonymization: How We Won the Kaggle Social Network Challenge, in: Proceedings of the International Joint Conference on Neural Networks, 2011. doi:10.1109/IJCNN.2011.6033446.

[4] P. Cotton, Where Will a Badminton Player Move to Next, and How Should we Adjudicate Predictions of the Same? (July 2020).
URL `https://www.microprediction.com/blog/badminton`

[5] S. Makridakis, C. Fry, F. Petropoulos, E. Spiliotis, The future of forecasting competitions: Design attributes and principles.
URL `https://arxiv.org/abs/2102.04879`

[6] P. Cotton, The future of forecasting, according to the experts.
URL `https://www.microprediction.com/blog/future`

[7] P. Cotton, Self Organizing Supply Chains for Micro-Prediction: Present and Future uses of the ROAR Protocol (2019).

[8] P. Cotton, Microprediction.Org (July 2020).
URL `https://www.microprediction.org`

[9] P. Cotton, Timemachines: A Python Package for Creating and Assessing Autonomous Time-Series Prediction Algorithms (2021).
URL `https://github.com/microprediction/timemachines`