Algorithmic Finance 8 (2019) 1–4
DOI:10.3233/AF-190900
IOS Press

Editorial

Information leakage in financial machine learning research

Zachary David[a,b,*]
[a]*Urvin AI, Philadelphia, PA, USA*
[b]*Northwestern University, Evanston, IL, USA*

1. Introduction: The research pipeline

Over the past ten years, applications of machine learning research have steadily shown mastery of broader and more complex domains, and the tools to create advanced models have become ubiquitous and more democratized. During that same time, the resources for studying algorithmic finance have also greatly expanded with populous open source communities and more granular time series available to researchers. However, progress in applying machine learning to finance has mostly stagnated, and in some cases regressed. We not only see similar, routine mistakes to those made ten years ago, but the complexity of cutting-edge models and frameworks can easily hide those mistakes.

Drawing on both referee reports and published papers, this editorial series will review frequent problems that occur throughout the *ML research pipeline* which lead to invalid results and unwarranted conclusions.[1] The pipeline refers to the series of research stages that are common to all ML projects and independent of any specific machine learning technique or architecture. The major stages are as follows:

- Universe: the set of financial instruments and time period on which the algorithm operates.
- Targets: the types of predictions or decisions that the algorithm generates at each step.
- Features: the set of information inputs that the algorithm uses to generate the targets.
- Models: the techniques, architectures, and parameters that comprise the algorithm.
- Metrics: the criteria on which the fitness and performance of the algorithm is judged.
- Testing & Validation: the procedures used to evaluate the performance and generalizability of the algorithm.

Throughout this series we will stress the delicate, interdependent relationships between the stages and show how mistakes made in one can spoil the whole. For the inaugural edition, we focus on the problem of "information leakage" and how to prevent it.

2. Information leakage

When simulating predictions or decisions that an algorithm *would have made* based on historical data, it is important to ensure that for each point in the simulated historical time, the algorithm doesn't have access to information from after that time. We say that "information has leaked" when a predictor or decision maker has access to information that has not yet occurred at the simulated time a prediction or decision is supposed to be made.

Over half of all submissions, and even many published papers, contain errors resulting from information leakage. The good news is that these errors are primarily mechanical in nature and often can be avoided with small adjustments to data preparation procedures or coding practices. This section contains a broad, albeit non-exhaustive, sample of issues that

*Corresponding author: Zachary David, Urvin AI, Philadelphia, PA, USA. E-mail: zak@urvin.ai.

[1] In the interest of maintaining the anonymity of submitters and reviewers, as well as promoting a constructive environment, we won't list any specific papers or authors.

arise at different points in the research pipeline. For each issue, after a description and some examples, we offer a suggestion for how to avoid it in practice.

2.1. Survivorship bias in universe selection

Information often begins leaking as soon as a researcher selects the universe of securities they'd like to operate on. Survivorship bias is primarily associated with the act of "picking winners while ignoring losers." Something as simple as selecting a set of companies which currently exist means the algorithm ignores those that have merged or gone bankrupt over the sample period. More generally, selection bias occurs when stocks or other financial instruments are selected based on their current characteristics (e.g. the current S&P 500 constituents) rather than the characteristics they had at the time the algorithm is supposed to simulate a prediction. When that occurs, any algorithm developed and tested using historical data contains information about the future.

Similar to conditioning on dependent variables, this future information risks biasing algorithms and inhibiting generalizability by overweighting spurious or unimportant features. For example, if the current 100 largest companies are chosen to develop a daily trading algorithm using the past 20 years of data, it's not only likely to be directionally biased, but the model might heavily weight any features that are strongly correlated with the success of the selected stocks even if the features were identical in companies that went bankrupt during the sample period.

Suggestion: use a historical database that includes delisted companies, mergers, and symbol changes. Construct the algorithm's logic to apply the selection criteria to only the universe that is available at the simulated time.

2.2. Data snooping in feature selection

As the set of all possible features tends to be unmanageably large, researchers frequently evaluate a large set of candidate features over the entire time period in order to choose a tractable subset from which to construct their model. Common techniques include dimensionality reduction by removing correlated features or evaluating which features explain the most variance in the target set. However, these processes inevitably imbue the algorithm with the knowledge of future relationships, either between features or with the target. In most cases, these relationships cannot reasonably be assumed to be stable over time. The risk of spuriousness is further increased when the features are engineered from the original time series.

Suggestion: as the algorithm progresses through simulated historical time, apply feature selection or dimensionality reduction criteria only to data that has already been observed by that point in simulated time. While this could imply that different step sizes will result in different feature sets, it makes a much stronger claim to generalizability.

2.3. Preprocessing over the whole data set

Despite their widespread use in published research, certain data preprocessing techniques leak information—such as normalizing price returns over the full data set or bounding factors between -1 and 1 based on the max and min. An algorithm which needs to simulate a decision at a time t should not use values which have been normalized by data occurring later than time t, otherwise the decision will be made based on information about how the data at time t relate to future values. For example, a paper studying US equities returns from 2000 through 2010 might understate z-scores during the first dotcom collapse if the data are normalized by data from the future Great Recession.

Similarly, denoising and filtering methods (e.g. wavelet transforms) have become a part of the data processing pipelines for many financial machine learning papers in the past few years. This typically involves operating on the full data set to obtain a denoised time series prior to the model training and prediction steps. As a result, predictions will be made based upon values which are influenced by and contain characteristics of future data.

Suggestion: one way to avoid these types of leaks is to only normalize, create bounds, or apply denoising to data at a time t using data prior to that time.

2.4. Off-By-One Errors

The simplest form of information leakage commonly occurring in code is an off-by-one error, where the feature and target sets are misaligned so that the values going into the model were constructed from the same information as the target values. This frequently affects papers which attempt to predict an instrument's future price return or direction. For example, if the feature set is constructed from values using a stock's daily closing price and the target set is the sign of the difference between the closing prices

at a time t and t–1, then the feature set at a time t will contain the target information unless one of the sets is shifted.

Suggestion: to fix the above example, we can either shift the features set forward by one so that the inputs only use variables from time t–1 or shift the target set back by one so that the targets are always from time $t + 1$. Overall, how we fix this type of error will depend on the relationship between the feature and target sets. For algorithms concerned with predicting future values, the general rule is that the time period of the target should not overlap with that of the features.

2.5. *Time traveling in fundamentals and accounting databases*

While this a problem that has historically affected factor analysis research, more machine learning papers have started integrating information from company filings and other fundamental databases. Information is leaked when algorithms assume that quarterly reports were available on the closing day of the reporting period rather than on the day that they were actually released. As Marcos Lopez de Prado writes in *Advances in Financial Machine Learning*:

> For example, fundamental data published by Bloomberg is indexed by the last date included in the report, which precedes the date of the release (often by 1.5 months). In other words, Bloomberg is assigning those values to a date when they were not known.

Indeed, throughout "earnings season," companies not only report earnings on different days, but also at different times of the day (e.g. pre- or post- market hours). High-quality research which incorporates fundamental factors will be keenly aware of these differences and explicitly account for them in both the manuscript and code.

Suggestion: first, ensure that the fundamentals database provides the dates for when the reports were publicly available. Next, as there likely won't be a single date on which everything can be rebalanced, consider how to structure the algorithm so that it uses the most available information up to a given time.

2.6. *Training/Test sets and cross validation*

The most subtle and challenging forms of information leakage occur during the testing and validation steps. In order to illustrate a few common examples, let's consider a scenario where our feature set consists of quarterly fundamental and accounting factors from various companies over time, and for each quarter we're trying to target something about each company—e.g. stock performance, merger, or bankruptcy in the following quarter. We'll refer to the individual entries of features and corresponding targets by a tuple of (company, quarter), and for a visual aid, we can consider the entries as if they were organized in the table depicted by Fig. 2.6.1.

One way training data can wind up in the test set is during rolling-window recalibrations. Say we want to train and test a model using the previous year of data, then advance one quarter and repeat. Then in 2018_{Q4}, we'll take from the set of tuples of all companies going back to 2018_{Q1} and randomly split the data 80% and 20% between training and test sets respectively. When we advance to 2019_{Q1} and repeat the same steps, it's possible that the randomly divided test set contains values that were used in the training set in the previous step. When this occurs, the model is merely reusing training data.

Another form of information leakage affecting techniques which randomly split data has to do with the relative stability of features. Say our training set contains $(A,\ 2018_{Q1})$ and $(A,\ 2018_{Q3})$. If the test set contains $(A,\ 2018_{Q2})$ then it's likely that all the test features are in the neighborhood established by the training features from the quarters on either side. This is problematic when we want to talk about generalizability and the usefulness of the algorithm in true out-of-sample scenarios.

Lastly, future information leaks into the model training when there are two correlated companies and one of them contains training data which occurs after another. Say A and B are correlated and in the same sector and the training set contains $(A,\ 2018_{Q1}) \ldots (A,\ 2019_{Q2})$, $(B,\ 2018_{Q1}) \ldots (B,\ 2019_{Q1})$. If we test on $(B,\ 2019_{Q2})$ then the model is already aware of the likely path of the features for B via the 2018_{Q2} information in A — we could probably even predict the features of B with high accuracy. Again, this implies that the model might underperform on true out-of-sample data (e.g. 2019_{Q3}) when the model wasn't trained on information from that quarter.

Suggestion: how we structure our tests should reflect what we ultimately want to be able to say about the efficacy of the algorithm. In most cases, a time-based hold-out is preferable. Ensure that the time range of the test period does not overlap with the time range of the training period — including any periods where data is fed into the algorithm with-

Company / Quarter	2018_{Q1}	2018_{Q2}	2018_{Q3}	2018_{Q4}	2019_{Q1}	2019_{Q2}
A	$\{F,T\}_{A,2018_{Q1}}$	$\{F,T\}_{A,2018_{Q2}}$	$\{F,T\}_{A,2018_{Q3}}$	$\{F,T\}_{A,2018_{Q4}}$	$\{F,T\}_{A,2019_{Q1}}$	$\{F,T\}_{A,2019_{Q2}}$
B	$\{F,T\}_{B,2018_{Q1}}$	$\{F,T\}_{B,2018_{Q2}}$	$\{F,T\}_{B,2018_{Q3}}$	$\{F,T\}_{B,2018_{Q4}}$	$\{F,T\}_{B,2019_{Q1}}$	$\{F,T\}_{B,2019_{Q2}}$
C	$\{F,T\}_{C,2018_{Q1}}$	$\{F,T\}_{C,2018_{Q2}}$	$\{F,T\}_{C,2018_{Q3}}$	$\{F,T\}_{C,2018_{Q4}}$	$\{F,T\}_{C,2019_{Q1}}$	$\{F,T\}_{C,2019_{Q2}}$

Fig. 2.6.1. The hypothetical data set of features and targets used for the examples in Sec. 2.6.

out explicitly triggering a prediction (e.g. “warm-up periods”).

3. Conclusion

In this editorial we looked at several ways information leaks throughout the research pipeline. Preventing these problems will improve the reliability and generalizability of a paper’s results, and we look forward to reviewing submissions that explicitly address these issues.

In future editions we’ll discuss topics like the importance of domain knowledge with respect to market hours and contract-based instruments, selecting appropriate measurements, simulating trading activity, and the perpetually looming question of multiple hypothesis testing.