January 5, 2015 Nina Zumel data science, Expository Writing, Opinion, Practical Data Science, Pragmatic Data Science, Pragmatic Machine Learning, Statistics, Statistics To English Translation cross-validation, test/train split

Random Test/Train Split is not Always Enough

Most data science projects are well served by a random test/train split. In our book *Practical Data Science with R* we strongly advise preparing data and including enough variables so that data is exchangeable, and scoring classifiers using a random test/train split.

With enough data and a big enough arsenal of methods, it's relatively easy to find a classifier that *looks* good; the trick is finding one that *is* good. What many data science practitioners (and consumers) don't seem to remember is that when evaluating a model, a random test/train split may not always be enough.

The true purpose of a test procedure is to estimate how well a classifier will work in future production situations. We don't evaluate the classifier on training data because training error has a significant undesirable

bias of the model's future performance. The underlying assumption of

using a random test/train split is that future data is exchangeable with past data: that is, the informative variables will be distributed the same way, so that the training data is a good estimate of the test data — and the test data is a good estimate of future data.

However in many fields your data is not exchangeable due to time based issues such as auto-correlation, or because of omitted variables. In these situations, a random test/train split will cause the test data to look *too much* like the training data, and *not enough* like future data. This will tend to make a classifier look better than it really is, so you can't be sure that your testing procedure has eliminated bad classifiers. In fact, you might accidentally eliminate what would be a good classifier in favor of a worse one that outperforms it in this artificial situation. Random test/train split is clearly unbiased, but bad classifiers benefit more from insensitivity of tests than good ones. To prevent this, you must apply some of your domain knowledge to build a testing procedure that will safely simulate the possible future performance of your classifier.

This may seem like contrary information as many people mis-remember "random test/train split" as being the only possible practice and the only legitimate procedure for things like a clinical trial. This is in fact not true. For example, there are fields where a random test/train split would never be considered appropriate.

One such field is finance. A trading strategy is always tested only on data that is entirely from the future of any data used in training. Nobody ever builds a trading strategy using a random subset of the days from 2014 and then claims it is a good strategy if it makes money on a random set of test days from 2014. You would be laughed out of the market. You

is known in many fields as *backtesting*, or *hindcasting*. Finance would

happily use random test-train split — it is much easier to implement and less sensitive to seasonal effects — if it worked for them. But it does not work, due to unignorable details of their application domain, so they have to use domain knowledge to build more representative splits.

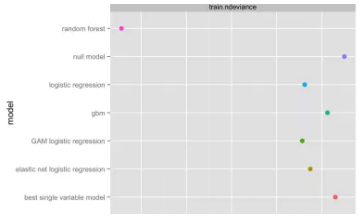
Another example is news topic classification. Classifying articles into categories (sports, medicine, finance, and so on) is a common task. The problem is that many articles are duplicated through multiple feeds. So a simple random test/train split (without article clustering and de-duplication) will likely put a burst of near duplicate articles into both the test and train sets, even if all of these articles come out together in a short time frame. Consider a very simple lookup procedure: classify each article as being in the topic of the closest training article. With a simple random test/train split, the test set will almost always contain a near duplicate of each article in the training set, so this nearest-neighbor classifier will work very well in evaluation. But it will not work as well in actual application, because you will not have such close duplicates in your historic training data to rely on. The random test/train split did not respect how time works in the actual application — that it moves forward and there are bursts of very correlated articles — and the bad testing procedure could lead you to pick a very ineffective procedure over other procedures that may work just fine.

Any classification problem where there are alignments to external data, grouping of data, concept changes, time, key omitted variables, auto-correlation, burstiness of data, or any other problem that breaks the exchangeability hypothesis needs a bit of care during model evaluation. Random test/train split may work, but there also may be obvious reasons why it will not work, and you may need to take the time to design appli-

bility for designing testing procedures that are good estimates of future

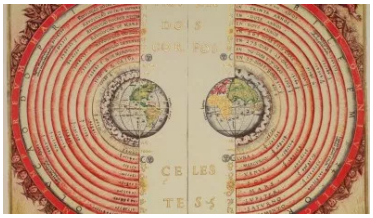
application, rather than simply claim random test/train split is always sufficient by an appeal to authority.

RELATED



How do you know if your model is going to work?
September 22, 2015
In "Opinion"

A deeper theory of testing
September 26, 2015
In "data science"



How do you know if your model is going to work? Part 1: The problem
September 2, 2015
In "Opinion"

January 5, 2015 Nina Zumel data science, Expository Writing, Opinion, Practical Data Science, Pragmatic Data Science, Pragmatic Machine Learning, Statistics, Statistics To English Translation cross-validation, test/train split

One thought on “Random Test/Train Split is not Always Enough”

I like this article, because it points out that there are multiple possible sources of testing error, and contrary to popular opinion there is not always a single automated domain-agnostic way to fix them all at once. You end up having to trade off possible bad influences in your test design. Obviously when you have reason to believe you have exchangeability a random test/train split is provably very good.

One view is: the random test/train split procedure is so powerful that after such a selection items in test and train sets are provably exchangeable prior to model construction. In principle this is a good property. But the procedure ensures exchangeability in the initial training data even if there were noticeable blockers to useful exchangeability in the original problem (concept drift, grouping, and so-on). So you may fail to notice non-exchangeability in your application (true future data not being exchangeable with past training data, needed for actual models in production) due to the strength of the random evaluation procedure. If you could exploit this strategy in production (use data available only in the future when scoring items in the past) you wouldn't mind.

This issue can set unrealistic expectations: "the model worked good during random test/train split, why doesn't it look as good after deployment."

Comments are closed.
