Introduction

Teaching basic data science, machine learning, and statistics is great due to the questions. Students ask brilliant questions, as they see what holes are present in your presentation and scaffolding. The students are not yet conditioned to ask *only* what *you* feel is easy to answer or present. They ask what is needed to further their understanding given the tools that have been already established.

This is one reason recordings can rarely match the experience of a good live course.

Such questions can take a bit of work to answer, another sign that they are valuable questions.

One data science question I am commonly asked is:

> What is a good test set size?

Let's take the time to give this question some serious consideration. I am going to have to address this question in a way I can not allow myself in class, by bringing in new tools that haven't been previously established.

For concreteness, let's suppose we have a supervised classification problem. We could also work similar arguments for regression.

We have, for each index `i` (called "rows" or "instances"), a numeric vector `x(i)` of explanatory variables (values we think we will have access to in the future), and a dependent variable `y(i)` that takes on the values `TRUE` and `FALSE`. Our goal is to learn from our training data a function `f()`, ideally such that `f(x(i))` usually near `1` when `y(i)` is `TRUE` and `f(x(i))` is usually near `0` when `y(i)` is false. The idea is: in the future we may have `x(i)` and not `y(i)`, so such an prediction procedure can be very valuable. For a general methodology how to convert arbitrary explanatory variables to numeric ones, please see the `vtreat` package (or alternately `vtreat` for Python).

Why we want `f()` to be a score, instead of a decision rule, can be found in our video "Against Accuracy".

In this note we will answer "what is a good test set size?" three ways.

- The usual practical answer.
- A decision theory answer.
- A novel variational answer.

Each of these answers is a bit different, as they are solved in slightly different assumed contexts and optimizing different objectives. Knowing all 3 solutions gives us some perspective on the problem.

Why Test

First let's remind ourselves why we need test data. Estimating the performance of a model on the same data used to fit the model, or even merely choose the model, is unreliable, unsafe, and biased. Some clear notes on this can be found here in our note "The Nature of Overfitting".

We need a reliable way to measure the performance of our fit function or model. Reserving some of the data for hold-out testing is one way to get a reliable future performance estimate.

The Usual Answer

My usual answer is to the "what is a good test set size?" is:

> Use about 80 percent of your data for training, and about 20 percent of your data for test.

This pretty standard advice. It is works under the rubric that model fitting, or training, is the harder task- so it should have most of the data. We say anything you can do with all the data, you have good chance of accomplishing with 80 percent of the data. And we use the remaining 20 percent for hold-out testing.

From a practical point of view, we are pretty much done. However, if we are already comfortable with some of the math, we can go on and analyze the data science process itself.

The Decision Theory Answer

Test data is used to estimate the future performance of a model on new data that was not known or seen during training. This sort of experiment can be characterized as a decision problem. In this section I want to convince you that from an acceptance test point of view it makes sense to talk of test set size in terms of row-count, and not in terms of what *fraction* of the available data it is. Once we have enough test rows, it doesn't matter how many more rows we could have used.

Suppose we have fit our model `f()`, and we have chosen what is called a "loss" or criticism function. In our case this is a function we can evaluate "per-row" and is of the form `loss(prediction, truth)`, where `prediction` is the number predicted, and `truth` is the `TRUE` or `FALSE` we are trying to match. For a binary (two outcome) classification problem we strongly recommend using "deviance" or using "cross entropy" for the loss function.

Let's take as our specific loss the following collared or Winsorized version of the deviance.

```
loss(prediction, truth) := -2 * (truth * log2(max(mu, p)) +
                                 (1-truth) * log2(max(mu, 1-p)))
```

where `mu` is a small positive constant, say `1/100`. `mu` is essentially saying: don't allow the model to make predictions outside of the range `[mu, 1-mu]`, which can be good advice (essentially Cromwell's rule).

An important feature of the above loss is: it is always in the range `[0, -2 * log2(mu)]`. For our assumed `mu = 1/100`, this means the per-row loss is always in the range `[0, 14]`. The bounded range is the only fact about the loss function we will use in this section!

We want `mean(loss(prediction(i), truth(i)))` evaluated on our test set to be very near the value we would see for `E[loss(prediction, truth)]` with `E[]` being the expected value over the assumed population *all* samples are drawn from. That is, we want a reliable estimate of future performance.

To achieve this, we need a sample size that makes the above correspondence true with very high probability. Such a size can be read off Hoeffding's inequality.

For our application we can take Hoeffding's inequality to be the following.

> For `t > 0`, the mean of `k > 0` identically distributed independent measurements that are all numbers in the range `[0, U]` is such that `P[|observed_mean - expected_value| > t] < 2 exp(-2 k t^2 / U^2)`.

So if we want only a `epsilon` chance of our test set being more than `t` units off in estimating future expected model performance, then it is enough to pick `k` such that `2 exp(2 k t^2 / U^2) <= epsilon`.

Or, a test set size of `k >= ln(2/epsilon) U^2 / (2 t^2)` is a large enough test set. Calculations using distributional details of the loss function will yield tighter bounds.

We can plug in some example values:

- **`U`** 14, as above, from picking `mu = 1/100`.
- **`epsilon`** 0.05, or a 5 percent chance of being further then `mu` off in our performance estimate.
- **`t`**, say `1/10`, or wanting our estimate to be likely to be with 0.1 deviance units of the unkown true value.

This gives us any `k >= ln(2/0.05) * 14^2 / (2 (1/10)^2)` is good test set size. Or, if we use `k >= 36152` test rows then with probability at least 95 percent we have an estimate that is within 1/10th deviance unit of the unknown true future performance of our classifier, independent of training set size and number of considered variables.

Notice `k` is a count depending only on the expected range of the loss (`U`), how large an error we wish to be unlikely (`t`), and what our definition of unlikely is (`epsilon`). `k` is independent of the available data size and complexity of the model fitting procedure. So in this case, `k` is not a fixed fraction of the available data (such as “20 percent”).

The Variational Answer

There is an alternate answer to “what is a good test set size?” that *is* a fraction of the available data, and doesn’t require specifying a desired “rare error” size. This is a bit less standard that the two earlier variations, but it is quite interesting. It is something new, that I would like to develop here.

What we are going to do is: split the data so we are improving our training and test reliabilities at roughly the same rate. In this scheme we assume there is no point making a good evaluation of a bad model, and no point building a good model if we can’t evaluate it.

We will assume that we can write the unreliability of our joint model fitting and model testing or evaluation procedure as an additive process. It may not truly be so, but let’s so estimate it.

We will try a path-oriented criticism, building up the overall criticism as moving from one concern to another. Our overall criticism is the sum of: how much our test set differs from testing on the entire population, plus how much our training procedure differs from training on the entire population, plus how much an ideal model differs from the data.

We are going to assume our overall model loss is of the form:

```
E[overall_criticism^2] ~ c_test / m_test + c_train / m_train
+ c_0
```

Where `m_train` is the number of rows or examples used to fit the model, `m_test` is the number of rows used to test or evaluate the model, and `c_train`, `c_test`, `c_0` are unknowns to be solved for, or estimated. This form is motivated in the appendix. The relation may, at first, appear odd. It is in fact merely saying we anticipate the loss squared behaves like a sample variance: the expected square size shrinks linearly in sample size.

If we accept the form `E[overall_criticism^2] ~ c_test / m_test + c_train / m_train + c_0`, we can then solve for the optimal partition into test and training sets, *with respect to this formulation*, and assuming we have estimates for `c_test` and `c_train`. Knowing the structure of the solution will be very useful in forming new intuition.

To solve the constrained optimization problem we set up a Lagrangian function for our problem. In our case this is:

```
  L := sqrt(c_test / m_test + c_train / m_train + c_0)  +
       lambda ( m_train + m_test - m )
```

This is standard. The first term is the function we are trying to optimize, the second term is a constraint we expect to be zero (that `m_train + m_test = m`, the total number of available rows), and `lambda` is a Lagrange multiplier to be solved for. We inspect stationary points of `L`, places where the derivatives are zero for possible optimal solutions.

```
 d L / d m_train
    = (1/2) L^(-1/2) * (-1/2 c_train / m_train^2) + lambda
    = 0
 d L / d m_test
    = (1/2) L^(-1/2) * (-1/2 c_test  / m_test^2)  + lambda
    = 0
```

So we expect:

```
  m_test  = Z c_test^0.5
  m_train = Z c_train^0.5
  m_train + m_test = m
```

Where `Z = 1/(2 * L^(1/4) * lambda^(1/2))`. Or, after some algebra to eliminate `Z`:

```
  m_test  = m * (c_test)^0.5  / ( (c_test)^0.5 + (c_train)^0.5 )
  m_train = m * (c_train)^0.5 / ( (c_test)^0.5 + (c_train)^0.5 )
```

If we had estimates of `c_train`, and `c_test`, they would determine a fraction of the data to devote to testing. For example, we have a synthetic example where we estimated `c_train` and `c_test` as the following.

```
c_test = 1.6
c_train = 79
```

These estimated coefficients indicate how much harder training is for testing for this model family on the example data set. Using these estimates we have:

```
soln <- list(
  fraction_test = (c_test)^0.5 / ( (c_test)^0.5 + (c_train)^0.5  ),
  fraction_train = (c_train)^0.5 / ( (c_test)^0.5 + (c_train)^0.5 ))
soln
```

```
## $fraction_test
## [1] 0.1245837
##
## $fraction_train
## [1] 0.8754163
```

Or, for this specific data process and model family, we should put about 12.5 percent of the data into the test set.

The summary is: for this formulation `c_train` and `c_test` approximately quantify the relative difficulty or sensitivity of training versus test for a given data population and model family. Knowing the relative difficulty of training to evaluation lets us pick an optimal fraction of data to use in training versus test. Obvious factors that can determine the difficulty of fitting include model structure, number of variables, and relations between variables.

In this formulation, we improve both training reliability and test reliability simultaneously instead optimizing one, and using the rest of the data for the other.

Conclusion

We have worked out in detail how to pick supervised machine learning test set size. We covered

- The usual Pareto principle style practical answer: use 80 percent of the data for training, and test on the rest.
- A more detailed "testing as an acceptance procedure" calculation, that yields a desired test set size as an absolute row count, and not as a fraction of available data.
- A novel variational argument which converts an estimate of training difficulty and test difficulty into an optimal fraction of data to use for training and testing.

And that is just a few of the interesting ideas one can develop from the question "what is a good test set size?"

Appendix: Deriving the Variational Loss Model

We can attempt to derive or motivate our variational loss model:

```
E[overall_criticism^2] ~ c_test / m_test + c_train / m_train
+ c_0
```

However, if one is willing to assume such a form, one can skip this derivation. In fact this entire appendix is just a very long-winded way of saying the loss squared should behave like a sample variance: the expected size should shrink linearly in sample size.

This appendix is *exactly* what I promise my students will not be presented in our courses: derivations that attempt to stand up core concepts they care about in terms of background concepts they may not care about! For our courses we are much stricter about assuming a small set of core concepts, and using only those. In this case, we would demonstrate the above relation empirically and move on.

However, as out of course preparation work, we are careful to have derivations. It cuts down our mistakes.

Let's define:

- **overall_criticism**: what we are trying to estimate, how bad our model *plus* model evaluation is as a function of training and test set sizes.
- **fit model test loss**: the loss of the model fit on our fixed training data when evaluated on our fixed test sample.
- **fit model ideal loss**: the loss of the model fit on our fixed training data when evaluated on the entire (unobserved) population all samples are being drawn from.
- **ideal model ideal loss**: the loss of a model fit using our procedures on the entire

(unobserved) population all samples are being drawn from, and then evaluated on this same population.

With these terms we can choose our overall loss or criticism as follows. It is an attempt to separate loss contributions into components driven by the test data set size, the training data set size, and the family of models we are using.

We factor the expected value of our criticism as follows.

```
E[overall_criticism^2]
   ( our assumption that we can expand as a sum of squares )
   := E[(test_criticism)^2  +
        (train_criticism)^2  +
        (ideal model ideal loss)^2]
   ( substituting in more precise definitions )
   := E[(mean(fit model test loss) - (fit model ideal loss))^2]  +
       E[((fit model ideal loss)  -  (ideal model ideal loss))^2] +
        (ideal model ideal loss)^2
    ( replacing each expectation by an ideal form )
    ~ c_test / m_test + c_train / m_train + c_0
```

Each of the above steps is something we are assuming, not something that is externally justified.

The asserted `E[(mean(fit model test loss) - (fit model ideal loss))^2] ~ c_test / m_test` identification we used above is derived as follows. Take `ifml` as equal to `fit model ideal loss`, so that `E[mean(fit model test loss) - ifml] = 0`.

```
E[(mean(fit model test loss) - (fit model ideal loss))^2]
    ( by definition )
    = E[( (1 / m_test) sum_{i = 1 ... m_test}
       loss(prediction(x(i)), truth(i)) - ifml )^2]
    ( expanding the square of the sum,
     and pushing E[] deeper into the expression,
     by linearity of expectation )
    = (1 / m_test)^2 sum_{i, j = 1 ... m_test}
       E[(loss(prediction(x(i)), truth(i)) - ifml) *
          loss(prediction(x(j)), truth(j)) - ifml)]
    ( applying independence )
    = (1 / m_test)^2 sum_{i, j = 1 ... m_test}
       E[(loss(prediction(x(i)), truth(i)) - ifml)] *
       E[loss(prediction(x(j)), truth(j)) - ifml)]
    ( applying E[mean(fit model test loss) - ifml] = 0 )
    = (1 / m_test)^2 sum_{i = 1 ... m_test}
       E[(loss(prediction(x(i)), truth(i)) - ifml)^2]
    ( removing index to simplify notation )
    = (1 / m_test)^2 m_test
       E[(loss(prediction, truth) - ifml)^2]
    ( simplifying )
    = (1 / m_test) E[(loss(prediction, truth) - ifml)^2]
```

Notice the final step is a term that shrinks at a rate of `1/m_test`.

We are assuming *with less overt justification*, that the training error term has a similar variance-

like structure. The idea is: in model fitting a number of things are estimated, and the quality of these estimates depends on the size of the training set. For many specific models (linear models, decision trees, bagged decision trees, and more) we *can* derive the matching `c_train / m_train` form using PAC-style or bounded VC-dimension-style arguments. So the form is, perhaps, plausible.