Quantitative Applications for Data Analysis: resampling

Erik Spence

SciNet HPC Consortium

17 March 2022

SciNet compute • calcul CANADA

Today's slides

Today's slides can be found here. Go to the "Quantitative Applications for Data Analysis" page, under Lectures, "Resampling".

`https://scinet.courses/1211`

Today's class

Today we will visit the following topics:

- Cross validation.
- Bootstrapping.
- Permutation tests.

With material stolen from L. Dursi.

SciNet compute • calcul CANADA

How do we choose the correct model?

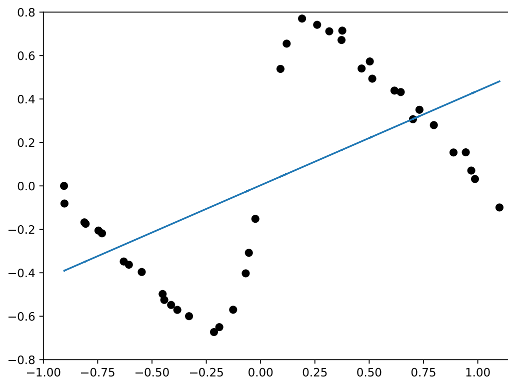
Let's consider the problem of fitting a polynomial to noisy data.

As you are likely aware, we can crank up the order of the polynomial and get a great fit to the data (even perfect!). But this won't do well on out-of-sample data.

So what do we do to choose the correct order of polynomial to fit to our data? How do we choose the correct model for our data?

Generate some data, and fit

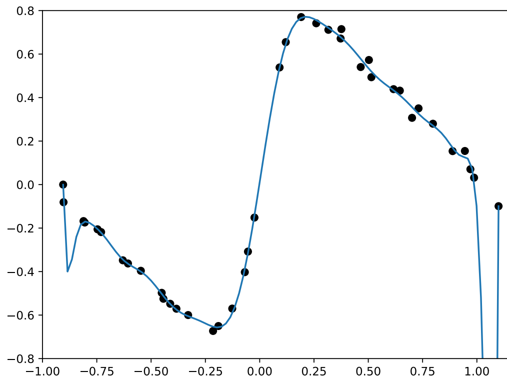
```
In [1]: import numpy as np
In [2]: import numpy.random as npr
In [3]: import matplotlib.pyplot as plt
In [4]: import sklearn.linear_model as sklm
In [5]:
In [5]: n = 40
In [6]: x = np.linspace(-1, 1, n).reshape(-1, 1)
In [7]: x += 0.1 * npr.rand(n).reshape(-1, 1)
In [8]: y = np.tanh(8 * x) - x
In [9]: y += 0.1 * npr.rand(n).reshape(-1, 1)
In [10]:
In [10]: model = sklm.LinearRegression()
In [11]: model = model.fit(x, y)
In [12]:
In [12]: plt.plot(x, y, "ko")
In [13]: plt.plot(x, model.predict(x))
```



The "+=" symbol means "add the right-hand side to the left-hand side".

Repeat with degree 20

```
In [14]: import sklearn.preprocessing as skp

In [15]:

In [15]: poly = skp.PolynomialFeatures(degree = 20)

In [16]: x_poly = poly.fit_transform(x)

In [17]: x_poly.shape
Out[17]: (40, 21)

In [18]:

In [18]: model20 = sklm.LinearRegression()

In [19]: model20 = model20.fit(x_poly, y)

In [20]:

In [20]: x2 = np.linspace(min(x), max(x), 100)

In [21]: plt.plot(x, y, 'ko')

In [22]:

In [22]: plt.plot(x2, model20.predict(
...:                    poly.transform(x2)))

In [23]:
```



Training versus validation

In general, we get our data, and that's it.

- We don't have the luxury of generating more data on a whim.
- We need to do out-of-sample testing of whatever model we generate, to make sure it generalizes to new data.
- But we often don't have any new data. What to do?
- The solution is to hold out some of the original data when we generate our model.
- Most of the data is used for training the model, the rest is used for validating it.
- These data should be chosen randomly.

Training versus validation, continued

So we hold out some data, the 'training' data, and build our model.

- Once the model is chosen, then you can train the selected model on the entire training + validiation data set.
- But you will probably still want to end your paper with a sentence like "the final model achieved 80% accuracy...".
- This can't be done using the data the model was trained on (train + validation)!
- Any data which has touched the model cannot be used for the final result.
- In this case, another chunk of data must be held out, for testing.

In the case of training-validation-testing, a common breakdown of the data sizes might be 50%-25%-25% of the initial set. If you don't need a test data set, 2/3-1/3 is common.

SciNet compute • calcul CANADA

k-fold Cross Validation

There are some downsides to the approach we've taken for validation hold-out. What if most outliers happen to be in the training set?

Ideally, we should do several partitions and average over the results. This is called k-fold Cross Validation:

- Partition the data set (randomly) into k sets.
- For each set:
  - Train on the remaining $k - 1$ sets.
  - Validate on the held-out set.
- Average the results.

Makes very efficient use of the data set, easily automated.

k-fold Cross Validation, continued

How do we choose k?

- if k is too large - the different training sets are very highly correlated (almost all of their points are the same).
- if k is too small - we don't get very much advantage of averaging.

In practice, 10 is a very commonly-used value for k; but again, this depends on the size of your data set.

k-fold cross-validation, regression example

The sklearn package has built-in functionality to make cross-validation easy.

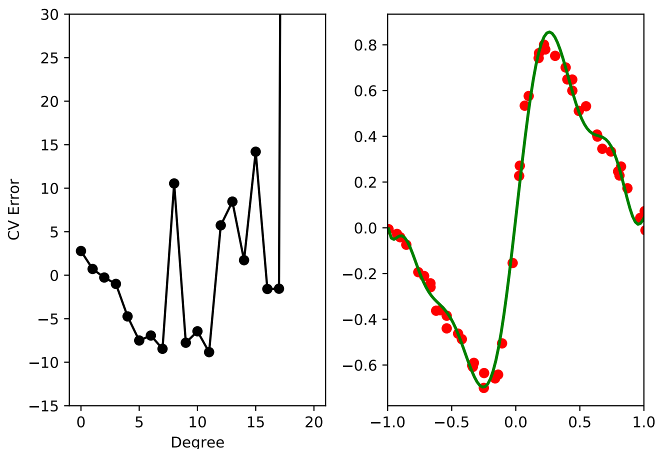
- The cross_validate function will do cross validaton for you.
- Unfortunately, it won't do so randomly. You need to use the KFold function to force it to be random.
- The model_selection subpackage has a KFold function. It returns the indices of the training and testing data.
- By default KFold does not shuffle the indices, you need to tell it to do so.

```python
# crossvalidation.py
import numpy as np
import sklearn.model_selection as skms
import sklearn.preprocessing as skp
import sklearn.linear_model as sklm
import sklearn.pipeline as skpi

def estimateError(x, y, d, kfolds = 10):

  pipe = skpi.Pipeline([
    ("poly", skp.PolynomialFeatures(degree = d)),
    ("model", sklm.LinearRegression())])

  cv_score = skms.cross_validate(pipe, x, y,
    cv = skms.KFold(n_splits = kfolds,
      shuffle = True).split(x))["test_score"]

  return -np.sum(cv_score)
```

k-fold cross-validation, regression example, continued

We run this function on 50 points using 10-fold cross-validation.

We calculated the error for each degree; the minimum is chosen. In practise, the simplest model that is "close enough" to the minimum is generally a good choice.



Cross-validation and boostrapping

Cross-validation is closely related to a more fundamental method, bootstrapping.

Let's say you want to find some statistic on some statistic of your data.

- What is the standard deviation of the 5th quantile of your data?
- What is the mean and standard deviation of an estimation error for a given model?
- What is the 95% confidence interval for some calculated quantity?

You'd like new sets of data that you could calculate your statistics on, and then look at the distribution of those.

Non-parametric Bootstrapping

The key insight to the non-parametric bootstrap is that you already have an unbiased description of the process that generated your data - the data itself.

The approach for the non-parametric bootstrap is:

- Generate synthetic data sets from the original data set by resampling;
- Calculate the statistic of interest on these synthetic data sets, and get the distribution of that particular statistic.

Cross-validation is a particular case: CV takes k (sub)samples of the original data set, applied a function (fit the data set to part, calculate error on the remainder), and calculates the mean.

Bootstrapping can be used far more generally: any time you need to estimate statistics on a quantity whose statistics aren't automatically calculated.

Non-parametric bootstrapping, example

Suppose you want to get statistics on the median of your data. How would you get the uncertainty on the median?

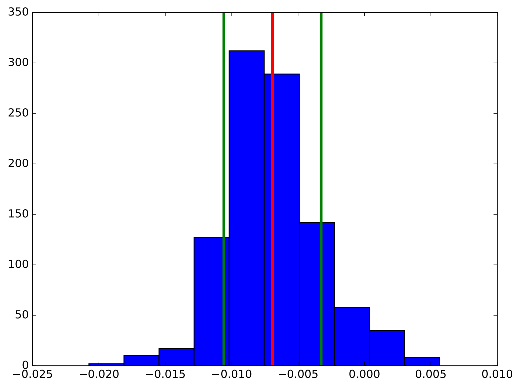
- Randomly sample from your data to create a fake data set.
- By default numpy.random.choice sets "replace = True", so that you are sampling from the full population.
- Do this many times.
- Calculate statistics on the resulting distribution.

```
In [23]: import sklearn.datasets as skd
In [24]: import numpy.random as npr
In [25]: dia = skd.load_diabetes()
In [26]:
In [26]: bmi = dia['data'][:,2]
In [27]:
In [27]: meds = [np.median(npr.choice(bmi, 200))
                 for i in range(1000)]
In [28]:
In [28]: np.mean(meds)
Out[28]: -0.0069275493192715136
In [29]:
In [29]: np.var(meds)
Out[29]: 1.3415090494694259e-05
In [30]:
```

SciNet compute • calcul CANADA

Non-parametric bootstrapping, example, continued

```
In [30]:
In [30]: plt.hist(meds)
In [31]:
In [31]: mean_meds = np.mean(meds)
In [32]: std_meds = np.sqrt(np.var(meds))
In [33]:
In [33]: plt.axvline(mean_meds, lw = 3,
                     color = 'red')
In [34]: plt.axvline(mean_meds + std_meds,
                     lw = 3, color = 'green')
In [35]: plt.axvline(mean_meds - std_meds,
                     lw = 3, color = 'green')
In [36]:
```



We now have an estimate of the uncertainty on the median.

Notes on Bootstrapping

Bootstrapping strengths:

- Allows you to get information on a calculated quantity when the true distribution of that quantity is unknown.

Bootstrapping weaknesses:

- If the statistic of interest is at the edge of parameter space (minimum, maximum, for example) the bootstrapped distribution does not converge to the true distribution.
- If you have too few data points to begin with, bootstrapping will not magically make things better. Your data must be a true representation of the population from which it is drawn.
- If your data's probability distribution has a long tail, or infinite moments, bootstrapping will fail, or give wildly inaccurate results. Examples include the Cauchy distribution, and non-central Student t distribution with 2 degrees of freedom.

SciNet compute • calcul CANADA

Parametric Bootstrapping

If you know the form of the distribution that describes your data, you can simulate new data sets:

- Fit the distribution to the data;
- Generate synthetic data sets from the now-known distribution to your heart's content;
- Calculate the statistics on these synthetic data sets, and get their distribution.

This works perfectly well if you know a model that will correctly describe your data; and indeed if you do know that, it would be madness *not* to make use of it in your analysis.

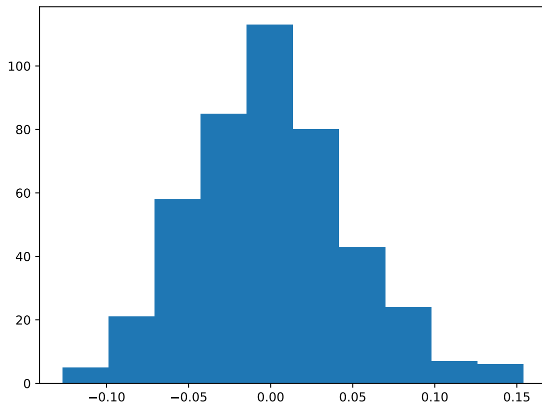
SciNet compute • calcul CANADA

Parametric Bootstrapping, example

Suppose we want to do a parametric bootstrap on our data, instead of non-parametric.

The data look pretty Gaussian, let's pretend that we know that the data are Gaussian.

```
In [36]: d = skd.load_diabetes()
In [37]:
In [37]: s1 = d['data'][:,4]
In [38]:
In [38]: plt.hist(s1)
```



Parametric bootstrapping, example, continued

Let's assume that the data is Gaussian. Let's calculate the distribution of the third data point.

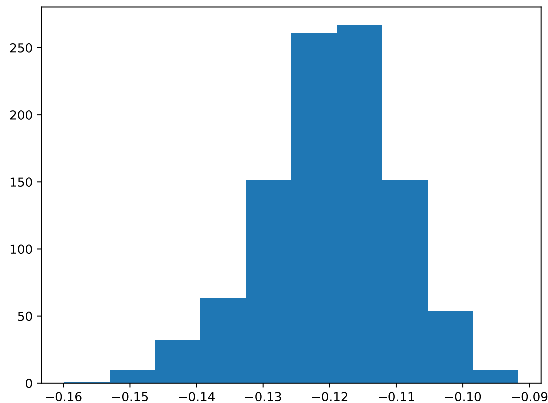
- Create a function which creates new data for you, based on the functional form that you are assuming.
- Create another function that calculates the statistic that you're interested in, from some data.
- Run many times, as with non-parametric bootstrapping.

```
In [39]:
In [39]: from scipy.stats import norm as ssn
In [40]:
In [40]: def my_third(data): return(np.sort(data)[2])
In [41]:
In [41]: def my_data(data):
             return(ssn.rvs(size = len(data),
                   loc = np.mean(data),
                   scale = np.sqrt(np.var(data))))
In [42]:
In [42]: thirds = [my_third(my_data(s1))
                   for i in range(1000)]
In [43]:
```

Parametric Bootstrapping, example, continued more

```
In [43]:
In [43]: plt.hist(thirds)
In [44]:
In [44]: np.percentile(thirds, [2.5, 97.5])
Out[44]: array([-0.14248356, -0.1021044 ])
In [45]:
```

You can use the 'percentile' function to get the 95% confidence interval.



Jackknifing

Another resampling technique is 'jackknifing'.

- This is a special case of non-parametric bootstrapping.
- Generally used to estimate the bias and variance of a particular statistic.
- In this use-case, the statistic of interest repeatedly recalculated while leaving out one or more different data points. The distribution of the statistic is then analysed.
- Less computationally intensive than bootstrapping, since random numbers are left out.
- Not as common as bootstrapping.
- The 'bootstrap' package contains functionality to perform jackknifing.

We won't do an example of this, but you need to be aware that it exists.

Permutation tests

Another resampling tool is the permutation test.

- Permutation tests commonly appear when we are interested in the null hypothesis of no difference between two treatment groups.
- Like non-parametric bootstrapping, we build distributions by sampling from our existing data set. In permutation tests, this is done by "shuffling" the observations in the data (move the data from group A to group B).
- In this case, the permutation test exactly represents the inference process we are testing.
- Why? Because the null hypothesis is that there's no difference between the two groups. Thus, if we change the outcome of a particular subject from category A to B, the statistics shouldn't change if the null hypothesis is true.
- The two-sample t-test (parametric) and Mann-Whitney U test (non-parametric) are also used for testing this null hypothesis.

SciNet compute • calcul CANADA

Permutation tests, continued

How does it work, exactly?

- A full permutation test would consider every single possible permutation of the data (shuffling group A and group B data).
- This gets out of hand quickly, even for small data sets. Shuffling 20 data points would mean $\binom{20}{10}$ combinations, (assuming two equally-sized groups) which is 184,756.
- We instead perform an "approximate permutation test" by randomly sampling from the space of all possible permutations.
- For each permutation, we calculate the statistic that we're after, and thus get a distribution. We then compare the distribution to the original value of the statistic (usually the mean).

SciNet compute • calcul CANADA

Permutation test, example

Consider again the breast cancer data set.

- Let's separate the data into malignant and benign tumours.
- First lets do a two-sample t test.
- We'll use column 17 (the "concavity" of the tumour).

```
In [45]: data = skd.load_breast_cancer()

In [46]:

In [46]: malignant = data['data'][data['target'] == 1, ]

In [47]: benign = data['data'][data['target'] == 0, ]

In [48]:

In [48]: import scipy.stats as ss

In [49]:

In [49]: ttest = ss.ttest_ind(malignant[:,16],
                              benign[:,16])

In [50]:

In [50]: ttest.pvalue
Out[50]: 8.260176167970112e-10

In [51]:
```

Permutation test, example, continued

Let's do a permutation test.

- We use the 'mlxtend' library (you may need to install this).
- The 'approximate' flag indicates that we're not going to do every possible permutation.
- The 'num_rounds' flag indicates how many permutations to perform.

It turns out that this package approximates 10^{-10} as zero.

```
In [51]:
In [51]: import mlxtend.evaluate as mlx
In [52]:
In [52]: mlx.permutation_test(malignant[:,16],
                              benign[:,16],
                              method = 'approximate',
                              num_rounds = 2000)
Out[52]: 0.0004997501249375312
In [53]:
```

Summary

We've taken a look at resampling methods. Some things to remember:

- Split your data into training, testing, and optionally, validation data sets. Train using the training data, test the model on the test data.
- Use cross-validation to determine the free parameters of your models!
- Bootstrapping can be used to get statistics on calculated quantities.
- Use non-parametric bootstrapping if you don't know the distribution of your data. Use parametric if you do.
- If you're doing your analysis in R, look into the 'boot' package, which has a lot of bootstrapping functionality built into it.
- Permutation tests are a family of resampling techniques which perform tests on data, by shuffling the data sets. They can be used to complement other tests.

SciNet compute • calcul CANADA