STAT 345-01: Nonparametric Statistics

Problem Set 1

Assigned 2018 August 28
Due 2018 September 4

Show your work on all problems! Be sure to give credit to any collaborators, or outside sources used in solving the problems. Note that if using an outside source to do a calculation, you should use it as a reference for the method, and actually carry out the calculation yourself; it's not sufficient to quote the results of a calculation contained in an outside source.

Please hand in parts one and two separately.

1 Part One

1.1 Conover Problems on Probability

Exercise 1.4.2, **Exercise 1.5.10**, **Review Problem 1.6.2**, **Review Problem 1.6.3**, **Review Problem 1.6.14**

1.2 Conover Problems on Statistical Inference

Problem 2.1.1, **Exercise 2.2.2**, **Exercise 2.2.8**

2 Part Two

Please turn in some sort of transcript of your python session, along with answers to the questions posed. If you want to submit electronically, please send either a pdf or a plain ASCII file. No Word documents!

2.1 Practice with Python

The main point of this problem is to gain some experience with a versatile datatype, the NumPy array. This exercise is most easily done within an interactive interpreter like IPython. As a matter of practice, I recommend always starting the session with

```
from __future__ import division
```

This enforces the Python 3 behavior of doing floating point division unless integer division is explicitly invoked with the `//` operator. (Not necessary if you're already using Python 3, of course.) Then

```
import numpy as np
```

(a) Define an array

```
A_i = np.arange(5)
```

You can check the dimensions of the array with `A_i.shape` and display its contents with `print(A_i)`. Also, access the initial element with `A_i[0]` (python indexes from zero). If you want the last element you can use `A_i[-1]`. What happens if you ask for `A_i[5]`?

(b) Define another array with

```
B_i = np.ones(5)
```

check its shape, and display its contents. One of the nice things about NumPy is that arithmetic operations are vectorized. (This is faster than looping over the elements in Python because the operation is coded up in C.) Define

```
C_i = A_i + B_i
```

and show its shape and contents.

(c) You can also add indices to arrays. There are a lot of variants of the syntax, but here's my favorite. Define

```
A_xi = A_i[None,:]
E_j = 10.*np.arange(3)
E_jx = E_j[:,None]
```

and show the shape and contents of `A_xi`, `E_j`, and `E_jx`.

(d) Now see what happens when you define

```
F_ji = A_xi + E_jx
```

and show its shape and contents. This is known as **array broadcasting**. What is `F_ji[1,3]`?

(e) You can also remove indices by taking the sum (or product, or average, or various other operations) along one of the "axes" of the array. For example, try

```
G_i = F_ji.sum(axis=0)
```

and show its shape and contents.

2.2 Empirical Confidence Interval Checking

(a) In this problem, we'll be using the Central Limit Theorem to construct a confidence interval using the normal distribution, with endpoints $\overline{x} \pm z_{1-\alpha/2} s/\sqrt{n}$. We will carry out $N = 10^4$ replications of an experiment which involves drawing a sample of size $n = 50$ to construct a confidence interval of confidence level $1 - \alpha = 90\%$ on the mean μ of the sampling distribution. The fraction of those N confidence intervals containing the true mean μ (which we will arbitrarily set to 5 for this experiment) should be about 90% if the procedure is correct. We can set things up with a few standard commands

```
from __future__ import division
import numpy as np
from scipy import stats
N = 10**4
n = 50
mu = 5.
alpha = 1. - 0.90
zcrit = stats.norm.isf(0.5*alpha)
```

If each interval has a probability of 90% of containing the true value, what is the total number we expect out of 10,000? What's the standard deviation associated with that expectation?

(b) Let's check this with the normal distribution, which we can initialize with

```
mydist = stats.norm(loc=mu)
```

We can draw a sample with

```
x_ji = mydist.rvs(size=(N,n))
```

and calculate the summary statistics with

```
xbar_j = x_ji.mean(axis=-1)
s_j = x_ji.std(axis=-1,ddof=1)
```

The `ddof` stands for "degenerate degrees of freedom", which basically just means to use $n-1$ rather than n in the normalization of the sample variance. We can define the ends of the confidence interval with

```
CIlo_j = xbar_j - zcrit*s_j/np.sqrt(n)
CIhi_j = xbar_j + zcrit*s_j/np.sqrt(n)
```

To check how many intervals contain the true value of $\mu = 5$, we can use the construction that `CIlo_j<mu` will return an array of `True` and `False` values based on whether the inequality is true for each element of `CIlo_j`.

```
inCI_j = np.logical_and(CIlo_j<mu, mu<CIhi_j)
```

We can get the total number of such intervals with

```
print(np.sum(inCI_j))
```

and the sum with

```
print(np.mean(inCI_j))
```

What fraction of the N confidence intervals contain μ? Each of the N confidence intervals has a different width, but display `CIhi_j-CIlo_j`, and you'll see the first and last few. What is the median with among the N widths?

(c) Repeat the process for a Laplace distribution.

(d) Repeat the process for a Student-t distribution with $\nu = 3$.
(Hint: use `mydist = stats.t(loc=mu,df=3)`).

(e) Repeat the process for a Cauchy Distribution. What goes wrong in this case?