All of Statistics - Chapter 14 Solutions

May 27, 2021

1.

By linearity of expectation,

$$\mathbb{E}\left[a^\intercal X\right] = \sum_i a_i \mathbb{E} X_i = \sum_i a_i \mu_i = a^\intercal \mu.$$

Since

$$AX = \begin{pmatrix} A_{1*} \\ \vdots \\ A_{n*} \end{pmatrix} X = \begin{pmatrix} A_{1*}X \\ \vdots \\ A_{n*}X \end{pmatrix},$$

it follows that

$$\mathbb{E}\left[AX\right] = \begin{pmatrix} A_{1*}\mu \\ \vdots \\ A_{n*}\mu \end{pmatrix} = A\mu.$$

Similarly,

$$\mathbb{V}(a^\intercal X) = \sum_{ij} a_i a_j \operatorname{Cov}(X_i, X_j) = a^\intercal \Sigma a.$$

It follows that

$$\left[\mathbb{V}(AX)\right]_{ij} = (A_{i*}X, A_{j*}X) = \sum_{uv} A_{iu} A_{jv} \operatorname{Cov}(X_u, X_v) = \left[A \Sigma A^\intercal\right]_{ij}.$$

2.

The log-probability of a multinomial is

$$\log f(X; p) = \log(n!) + \sum_j -\log(X_j!) + X_j \log(p_j).$$

Therefore,

$$\mathbb{E}\left[\frac{\partial^2 \log f(X;p)}{\partial p_j^2}\right] = \mathbb{E}\left[-\frac{X_j}{p_j^2}\right] = -\frac{n}{p_j}$$

so that the Fisher information matrix is

$$I(p) = n \operatorname{diag}(1/p_1, \ldots, 1/p_k).$$

3.

```
import numpy as np

def sample_multinomial(n, p):
    """Samples a multinomial distribution.

    Parameters
    ----------
    n : int
        Number of bins.
    p : array_like, shape=[k]
        Bin probabilities (summing to one).

    Returns
    -------
    counts : array_like, shape=[k]
        Realized bin counts.
    """
    k, = p.shape
    cdf = np.cumsum(p)
    samples = np.random.uniform(low=0.0, high=1.0, size=n)
    indices = np.searchsorted(cdf, samples)
    indices = np.concatenate([np.arange(k), indices])
    _, counts = np.unique(indices, return_counts=True)
    counts -= 1
    return counts
```

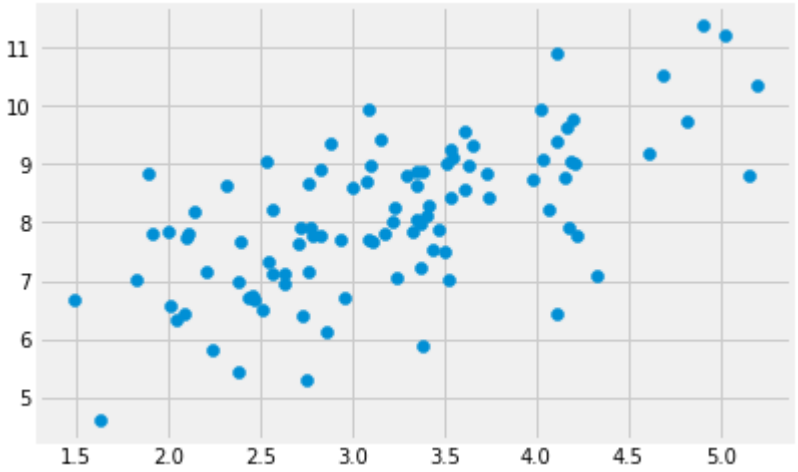
4.

```
def sample_multivariate_normal(mean, cov):
    """Samples a multivariate normal distribution.

    Parameters
    ----------
    mean : array_like, shape=[k]
        Mean.
    cov : array_like, shape=[k, k]
        Covariance matrix.

    Returns
    -------
    z : array_like, shape=[k]
        Realized samples.
    """
    k, = mean.shape
    sqrt_cov = sqrtm(cov)
    samples = np.random.randn(k)
    return mean + sqrt_cov @ samples
```

5.

The true correlation between $X1$ and $X2$ is $1/\sqrt{2}$.

100 points were sampled:



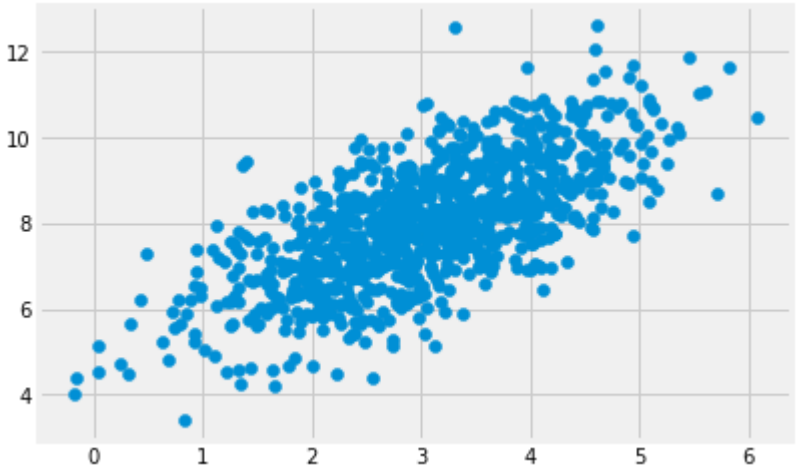
Various quantities were computed from these samples: (see Appendix for code)

```
Sample mean:
[3.19678877 8.08354934]

Sample covariance matrix:
[[0.68362962 0.67545853]
 [0.67545853 1.62768145]]

Sample correlation:
0.6403295531123555

Fisher's 95% confidence interval:
(0.5047523958530407, 0.7450792923252504)

Bootstrap 95% confidence interval:
(0.5124584072915059, 0.7682006989332051)
```

Fisher’s method and the (nonparametric) bootstrap yield comparable results.

6.

1000 points were sampled:



Various quantities were computed from these samples: (see Appendix for code)

```
Sample mean:
[3.04724353 8.05340182]

Sample covariance matrix:
[[0.98527055 0.95830459]
 [0.95830459 1.97142287]]

Sample correlation:
0.6876000872761875

Fisher's 95% confidence interval:
(0.6527346353163089, 0.7195590209752238)

Bootstrap 95% confidence interval:
(0.6532707565958478, 0.7219294179565272)
```

Fisher's method and the (nonparametric) bootstrap yield comparable results.

Appendix: Code for Exercises 5 and 6

```
n_points = 100
mean = np.array([3., 8.])
cov = np.array([[1., 1.], [1., 2.]])

np.random.seed(1)

def sample_cov_and_corr(points):
    sample_cov = np.cov(points, ddof=1, rowvar=False)
    sample_std = np.sqrt(np.diag(sample_cov))
    sample_corr = ((sample_cov / sample_std).T / sample_std).T
    return sample_cov, sample_corr

points = np.array(
    [sample_multivariate_normal(mean, cov) for _ in range(n_points)])
sample_mean = np.mean(points, axis=0)
sample_cov, sample_corr = sample_cov_and_corr(points)
sample_corr = sample_corr[0, 1]

def fisher_func(r):
    return 0.5 * (np.log(1 + r) - np.log(1 - r))

def fisher_func_inv(z):
    return (np.exp(2. * z) - 1) / (np.exp(2. * z) + 1)

theta = fisher_func(sample_corr)
a = theta - 2. / np.sqrt(n_points - 3)
b = theta + 2. / np.sqrt(n_points - 3)
fisher_lb = fisher_func_inv(a)
fisher_ub = fisher_func_inv(b)

bstrap_sample_corrs = []
for _ in range(1000):
    indices = np.random.randint(0, high=points.shape[0], size=points.shape[0])
    new_points = points[indices]
    _, bstrap_sample_corr = sample_cov_and_corr(new_points)
    bstrap_sample_corr = bstrap_sample_corr[0, 1]
    bstrap_sample_corrs.append(bstrap_sample_corr)
se = np.std(bstrap_sample_corrs)
bstrap_lb = sample_corr - 2. * se
bstrap_ub = sample_corr + 2. * se

print("""
Sample mean:
```

```
{}

Sample covariance matrix:
{}

Sample correlation:
{}

Fisher's 95% confidence interval:
({}, {})

Bootstrap 95% confidence interval:
({}, {})
""".format(sample_mean, sample_cov, sample_corr, fisher_lb, fisher_ub,
           bstrap_lb, bstrap_ub))
```