In-Class Programming Task 11

Math 253: Statistical Computing & Machine Learning

Correlated Random Variables

This activity involves *correlated* random variables. You're going to see how to describe simple correlations between variables, how to generate correlated variables, and the meaning and interpretation of the Σ^{-1} in the hard-to-understand Equation 4.18 from ISL:

$$f(x) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{p/2}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right).$$

Correlation and Covariance

The correlation, r, is a very simple description of a relationship between two variables. As you know, r is a number between -1 and 1. We're going to work with the *covariance*, which is an unnormalized version of correlation.

In the following, $\mathbf{X}_i$ will denote a vector and $\mathbf{X}_j$ is another vector. Think of them as different quantitative variables in a data table.

The covariance between vectors $\mathbf{X}_i$ and $\mathbf{X}_j$, with means μ_i and μ_j respectively, is:

$$\text{cov}(X_i, X_j) \equiv \text{E}\left[(\mathbf{X}_i - \mu_i)^T(\mathbf{X}_j - \mu_i)\right]$$

The *covariance matrix* is $\Sigma_{ij} \equiv \text{cov}(\mathbf{X}_i, \mathbf{X}_j)$. It is always a *symmetrical* matrix.

Matrices and their inverses

Consider this covariance matrix:

$$\Sigma \equiv \begin{bmatrix} 1.0 & -0.7 \\ -0.7 & 1.0 \end{bmatrix}$$

- Create a matrix object `Sigma` to represent Σ
- Create a matrix object `Sigma_inv` holding the inverse Σ^{-1}. (*Hint*: See `solve()`.)

You can construct the matrix product $\Sigma\Sigma^{-1}$ using the matrix multiplication operator `%*%` in R.

- Create a matrix object `test_inverse` that holds the product of `Sigma` and `Sigma_inv`.

Matrix Compositions

A major technique for working with matrices is to *factor* them. Factoring a matrix is much like finding the prime numbers that compose an integer, e.g. 14 factors into 2 and 7, since $2 \cdot 7 = 14$. There are many different matrix factorizations with names like LU, QR, and SVD. Each of the matrix factors typically has a very simple structure. One such structure is *upper triangular*, a matrix with all zeros below the diagonal. There are also *lower triangular* matrices, matrices whose columns are *mutually orthogonal*, and, of course, the *diagonal matrix* which is both upper and lower triangular.

For matrices that are symmetrical[1], an important factorization is called the *Cholesky decomposition*. (Sometimes this is called the "square root" of a matrix.) You can calculate the Cholesky decomposition using the `chol()` function in R.

[1] And positive definite, but never mind that for now since all covariance matrices are positive definite

- Create a matrix `A` which is the Cholesky decomposition of Σ.

The result of `chol()` is a single upper triangular matrix.

The *transpose* of a matrix is simply a flip around the diagonal. In R, this is accomplished with the `t()` function. Calculate `t(A)` and confirm that it is a *lower* triangular matrix.

Verify that the matrix product of `t(A)` and `A` is equivalent to A.

Orthogonal vectors and matrices

In previous work, you've generated random vectors.

- To start, make two vectors `x1` and `x2` of length 10 using `rnorm(10)`. The mean of a vector generated in this way is usually close to zero. (The standard error of the mean is $1/\sqrt{n}$, where n is the length of the vector.)

The *inner product* between two R vectors can be computed as `sum(x1 * x2)`. But we are going to work with vectors in the mathematical organization: a vector is a one-column matrix.

- Revise `x1` and `x2` to be *one-column* matrices. You can use `cbind()` for this.

In the matrix notation, the inner product can be computed as `t(x1) %*% x2`. Note the use of matrix multiply `%*%` rather than ordinary multiplication. Also note that the vector to the left of `%*%` is a *row* matrix. The transpose turns columns into rows and *vice versa*.

- Make a matrix `X` that has `x1` and `x2` side-by-side: a two-column matrix. Again, you can use `cbind()` for this purpose.

Multiply `t(X)` by `X`. You should get a symmetrical matrix where the off-diagonal elements are much smaller than the diagonal elements. This symmetrical matrix is closely related to the covariance matrix of the variables `x1` and `x2`. To get the covariance matrix, multiply `t(X)` by `X` and divide by the number of rows in `X`.[2]

[2] Strictly speaking, we should subtract off the mean of `x1` from `x1`, and similarly for `y1`.

- Generate vectors `w1` and `w2` and the matrix `W` in the same way as you just generated `X`, but instead of length 10, make them length 10,000. Also calculate `W_cov`, the covariance matrix for the two variables in `W`.

Note that the covariance matrix is very close to being diagonal. The is because the correlation between `y1` and `y2` is almost zero. When a correlation is zero, the variables are *uncorrelated*. Random vectors are almost always close to orthogonal. As the length n of the random vectors gets bigger, the correlation tends toward zero as $1/\sqrt{n}$.

Generating correlated random vectors

It's easy to generate sets of uncorrelated random vectors, such as `X` or `W`. Now you're going to modify `X` to produce a new matrix with covariance A (from the first section).

- Create matrix `A_inv` which is the Cholesky decomposition of the inverse of Σ. (Note that `A_inv` is meant to imply "Take the inverse of Σ, and then the Cholesky decomposition of that.")
- Multiply `X` by `A`. The result — call it `Y` — should be a matrix of the same dimensions as `X`.

The covariance matrix of `Y` should be similar to `Sigma`, but `Y` is so short that there will be lots of random fluctuations.

- Resassign `Y` to be the product of `W` times `A`. This will give you 10,000 random samples with a covariance matrix close to `A`.

Check whether the covariance matrix of this new, bigger `Y` is close to `Sigma`. The difference between `Sigma` and the covariance matrix of `Y` should be similar in size to $1/\sqrt{n}$. Is it?

- Finally, plot out the first column of `Y` against the second column. (Recall that the index brackets work as `[rows, cols]` and leaving an element blank means "all of them.") Your graph will give a picture of the multivariate Gaussian distribution with covariance `A`.

It's helpful to set at a low value the transparency of the of the points. This lets you see more detail of the dense part of the distribution. Try `col = rgb(0, 0, 0, .05)`.