In-Class Computing Task 15

Math 253: Statistical Computing & Machine Learning

Dimension Reduction Methods

In this activity, you're going to synthesize data from a linear system with multiple inputs and a single output, where only a few of the inputs contribute to the output.

Overview

A linear system is often written

$$\mathbf{Y} = \mathbf{X} \cdot \beta + \epsilon$$

In this notation, β are the *coefficients* of the system, $\mathbf{X}$ is the model matrix, and $\mathbf{Y}$ is the output. $\mathbf{Y}$ is not completely set by $\mathbf{X} \cdot \beta$, there is some random part to $\mathbf{Y}$, unrelated to any of the columns in $\mathbf{X}$, represented by ϵ.

We are going to make a seemingly strange choice of $\mathbf{X}$: the columns from a matrix representing a monochrome version of the Mona Lisa.[1] This matrix has strong correlations among the columns.

You will use the matrix `mona` to form $\mathbf{X}$ in your simulation.[2]

```
X <- t(mona) - mean(mona[])
```

The `[]` in `mona[]` means to treat `mona` as one long vector rather than as a matrix.[3]

Make two more matrices:

1. `X_rand` with has the same size as `X` but consists of iid N(0,1^2) noise.
2. `X_corr` with columns that have the covariance as `X`.[4]

Sparse beta

There are 191 columns in each of `X`, `Xrand` and `Xcorr`. Create a vector `beta` that has 191 numbers. Of these 191 numbers, 175 should be 0. The other 16 should have values of 2, 5, -3, or -4. The order should be random. (Hints: Use these functions `rep()`, `sample(1:191)` as well as indexing.) A vector or matrix consisting mainly of zeros is called *sparse*. The sparse β you are using here simulates a system where there are many inter-related variables in $\mathbf{X}$, but just a few of them contribute to the formation of $\mathbf{Y}$

The output

Create two output vectors based on $\mathbf{X} \cdot \beta$, using `X` for $\mathbf{X}$. Each of these output vectors will play the role of Y in the linear system you are simulating.

[1] The file is available at `"http://tiny.cc/dcf/mona.rda"` and contains a matrix `mona`. I suggest you first download the file (`download.file()`) to your computer, with `destfile = "mona.rda"`. Do this just once and **don't** put the `download.file()` command in your `.R` script unless you have commented it out. It is meant to be run only once. In your script, you can load `mona` using `load("mona.rda")`.

[2] Why use the transpose operator, `t()`? The row-column convention for images are reversed from those for matrices. `t(mona)` gives a matrix with 250 rows and 191 columns.

[3] A matrix can be thought of as a collection of vectors.

[4] Remember, you can create correlated noise from iid noise by post-multiplying by the square-root of the desired covariance matrix. That is, post-multiply `Xrand` with `chol(var(X))`.

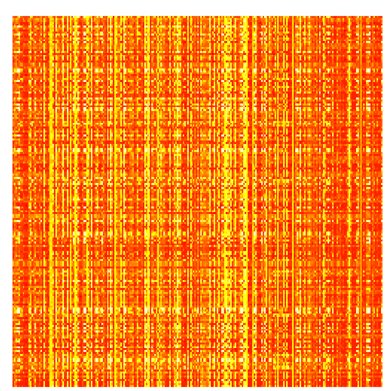

Figure 1: Mona Lisa? Same data as the original, but the order of variables and cases has been randomized.

1. Y_pure which is simply $\mathbf{X} \cdot \beta$.
2. Y_real which is $\mathbf{X} \cdot \beta$ plus noise which is iid normal with mean 0 and a variance that is 10% of the variance of Ypure.

Least squares

Use `lm()` to fit Y_pure against X. Create a vector beta_hat_pure to hold the coefficients. Plot beta_hat_pure against beta and draw a conclusion about the performance of `lm()` in this case.[5]

Now do the same for Y_real, creating a vector of fitted coefficients beta_hat_real and comparing that to your actual beta.

Would you be able to detect from beta_hat_real that β is sparse?

[5] Remember that your X has no vector corresponding to the intercept. Discard the intercept from the coefficients found by `lm()`.

The lasso estimator

Use the lasso method to estimate $\hat{\beta}$, which you can store in a vector beta_lasso. The glmnet package has a command `cv.glmnet()` which uses cross-validation to choose an appropriate value of λ. The commands look like this:

```
library(glmnet)
lasso_mod <- cv.glmnet(X, Y_real, alpha=1)
beta_lasso <- predict(lasso_mod, type = "coefficients", s = lasso_mod$lambda.min)
```

Principal components

Recall that each of the principal components of a matrix has a scalar — called the *singular value* — indicating the "size" of that principal component in contributing to the reconstruction of the matrix. You can find these scalars like this:

```
sing_vals <- svd(X)$d
```

The cumulative sum `cumsum(sing_vals^2)` divided by `sum(sing_vals^2)` produces the R^2 of the approximation of X using successively $k = 1, 2, 3, ..., 191$ principal components.

Find the singular values of X_rand and X_corr. On the same graph, plot out R^2 versus k for the singular values from each of X, X_rand, and X_corr.

Calculate how many principal components are needed to reconstruct the matrix with an R^2 of 99%. Call your answers n99, n99_rand and n99_corr respectively.

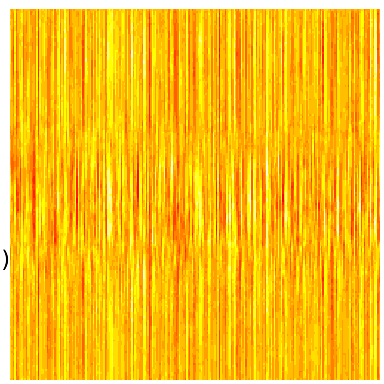

Figure 2: Linear Lisa? This is X_corr and has the same covariance as mona.

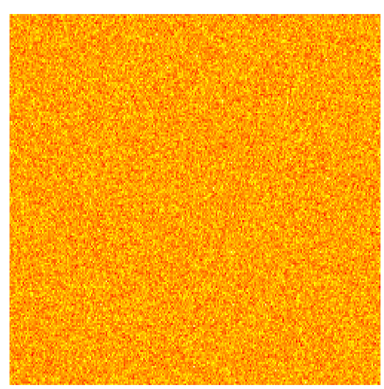

Figure 3: Iid Lisa?

FINALLY, USE PRINCIPAL COMPONENTS TO MODEL Y_real against X. The commands look like this.

```
library(pls)
pcr.fit <- pcr(Y_real ~ X, scale = TRUE, validation="CV")
```

Using `R2(pcr.fit)`, examine the cross-validated R^2 as a function of the number of components used. How many components are needed to get to, say, $R^2 = 0.85$.