Lecture 17: Multicollinearity

1 Why Collinearity Is a Problem

Remember our formula for the estimated coefficients in a multiple linear regression:
\[
\hat{\beta} = (X^TX)^{-1}X^TY
\]
This is obviously going to lead to problems if \( X^TX \) isn’t invertible. Similarly, the variance of the estimates,
\[
\text{Var} \left[ \hat{\beta} \right] = \sigma^2 (X^TX)^{-1}
\]
will blow up when \( X^TX \) is singular. If that matrix isn’t exactly singular, but is close to being non-invertible, the variances will become huge.

There are several equivalent conditions for any square matrix \( U \) to be singular or non-invertible:

- The determinant \( \det U \) (or \( |U| \)) is 0.
- At least one eigenvalue of \( u \) is 0. (This is because the determinant of a matrix is the product of its eigenvalues.)
- \( U \) is rank deficient, meaning that one or more of its columns (or rows) is equal to a linear combination of the other rows.

Since we’re not concerned with any old square matrix, but specifically with \( X^TX \), we have an additional equivalent condition:

- \( X \) is column-rank deficient, meaning one or more of its columns is equal to a linear combination of the others.

The last explains why we call this problem collinearity: it looks like we have \( p \) different predictor variables, but really some of them are linear combinations of the others, so they don’t add any information. If the exact linear relationship holds among more than two variables, we talk about multicollinearity; collinearity can refer either to the general situation of a linear dependence among the predictors, or, by contrast to multicollinearity, a linear relationship among just two of the predictors.

Again, if there isn’t an exact linear relationship among the predictors, but they’re close to one, \( X^TX \) will be invertible, but \((X^TX)^{-1}\) will be huge, and the variances of the estimated coefficients will be enormous. This can make it very hard to say anything at all precise about the coefficients, but that’s not necessarily a problem.

1.1 Dealing with Collinearity by Deleting Variables

Since not all of the \( p \) variables are actually contributing information, a natural way of dealing with collinearity is to drop some variables from the model. If you want to do this, you should think very carefully about which variable to delete. As a concrete example: if we try to include all of a student’s grades as predictors, as well as their over-all GPA, we’ll have a problem with collinearity (since GPA is a linear function of the grades). But depending on what we want to predict, it might make more sense to use just the GPA, dropping all the individual grades, or to include the individual grades and drop the average.
1.2 Diagnosing Collinearity Among Pairs of Variables

Linear relationships between pairs of variables are fairly easy to diagnose: we make the pairs plot of all the variables, and we see if any of them fall on a straight line, or close to one. Unless the number of variables is huge, this is by far the best method. If the number of variables is huge, look at the correlation matrix, and worry about any entry off the diagonal which is (nearly) ±1.

1.3 Why Multicollinearity Is Hard to Detect

A multicollinear relationship involving three or more variables might be totally invisible on a pairs plot. For instance, suppose \( X_1 \) and \( X_2 \) are independent Gaussians, of equal variance \( \sigma^2 \), and \( X_3 \) is their average, \( X_3 = (X_1 + X_2)/2 \). The correlation between \( X_1 \) and \( X_3 \) is

\[
\text{Cor}(X_1, X_3) = \frac{\text{Cov}[X_1, X_3]}{\sqrt{\text{Var}[X_1] \text{Var}[X_3]}}.
\]

This is also the correlation between \( X_2 \) and \( X_3 \). A correlation of \( 1/\sqrt{2} \) isn’t trivial, but is hardly perfect, and doesn’t really distinguish itself on a pairs plot (Figure 1).

```r
x1 = rnorm(100,70,15)
x2 = rnorm(100,70,15)
x3 = (x1 + x2)/2
X = cbind(x1,x2,x3)
pairs(X)
cor(X)
```

<table>
<thead>
<tr>
<th></th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>1.00000000 0.03788452 0.7250514</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x2</td>
<td>0.03788452 1.00000000 0.7156686</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x3</td>
<td>0.72505136 0.71566863 1.0000000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2 Variance Inflation Factors

If the predictors are correlated with each other, the standard errors of the coefficient estimates will be bigger than if the predictors were uncorrelated. If the predictors were uncorrelated, the variance of \( \hat{\beta}_i \) would be

\[
\text{Var} \left[ \hat{\beta}_i \right] = \frac{\sigma^2}{n s^2_{X_i}}.
\]

just as it is in a simple linear regression. With correlated predictors, however, we have to use our general formula for the least squares:

\[
\text{Var} \left[ \hat{\beta}_i \right] = \sigma^2 (X^T X)_{i+1,i+1}^{-1}
\]
Figure 1: Illustration that a perfect multi-collinear relationship might not show up on a pairs plot or in a correlation matrix.
The ratio between Eqs. 4 and 3 is the variance inflation factor for the $i^{th}$ coefficient, $VIF_i$. The average of the variance inflation factors across all predictors is often written $\overline{VIF}$, or just $VIF$.

Folklore says that $VIF_i > 10$ indicates “serious” multicollinearity for the predictor. I have been unable to discover who first proposed this threshold, or what the justification for it is. It is also quite unclear what to do about this. Large variance inflation factors do not, after all, violate any model assumptions.

It can be shown that $VIF_i = 1/(1-R_i^2)$ where $R_i^2$ is the $R^2$ you get by regressing $X_i$ on all the other covariates.

3 Matrix Perspective

Let $X$ be the $n \times q$ design matrix. (Remember that $q = p + 1$.) We call $G = X^TX$ the Gram Matrix. You should check the following facts:

1. $G$ is $q \times q$.
2. $G$ is symmetric.
3. $G$ is positive semi-definite. That means that, for any vector $a$ we have that
   \[ a^T Ga \geq 0. \]

Multicollinearity means that there exists a perfect linear relationship between the columns of $X$. This means that there is a non-zero vector $a = (a_1, \ldots, a_q)$ such that $\sum_j a_j X_j = 0$ where $X_j$ is the $j^{th}$ column of $X$. In other words, there exists $a \neq (0, \ldots, 0)$ such that $Xa = 0$. Hence

\[ a^T Ga = 0. \]

Since $G$ is a square, symmetric, positive-semidefinite matrix, it has a spectral decomposition (or eigen-decomposition). In other words, there are numbers (eigenvalues) $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_q \geq 0$ and vectors (eigenvectors) $v_1, \ldots, v_q$ such that:

1. $Gv_j = \lambda_j v_j$.
2. $v_j^T v_k = 0$ for $j \neq k$.
3. $v_j^T v_j = 1$ for each $j$.
4. $G = \sum_j \lambda_j v_j v_j^T$.
5. $G = VDV^T$ where the $j^{th}$ column of $V$ is $v_j$ and $D$ is a diagonal matrix with $D_{jj} = \lambda_j$.
6. The eigenvectors form a basis: any vector $w$ can be written as $w = \sum_j b_j v_j$ where $b_j = w^T v_j$.

Now if the design matrix is collinear then there is a $a$ such that $a^T Ga = 0$. Now

\[ 0 = a^T Ga = a^T \sum_j \lambda_j v_j v_j^T a = \sum_j \lambda_j a^T v_j v_j^T a = \sum_j \lambda_j (a^T v_j)^2 \equiv U. \]
If $\lambda_q > 0$ then $\lambda_j > 0$ for all $j$. We know that $(a^T v_j)^2 > 0$ for at least one $j$. (We know this since $a = \sum_j (a^T v_j) v_j$ and since $a \neq 0$.) So if $\lambda_q > 0$ then we get $0 = U > 0$ which is a contradiction. We conclude that $\lambda_q = 0$. (There could be other eigenvalues that are 0 as well.)

We have shown that:
Multicollinearity $\implies a^T G a = 0$ for some $a \neq 0 \implies$ at least one eigenvalue of $G$ is 0.

It is not hard to show that the reverse implications also hold.

3.1 Finding the Eigendecomposition

Because finding eigenvalues and eigenvectors of matrices is so useful for so many situations, mathematicians and computer scientists have devoted incredible efforts over the last two hundred years to fact, precise algorithms for computing them. This is not the place to go over how those algorithms work; it is the place to say that much of the fruit of those centuries of effort is embodied in the linear algebra packages R uses. Thus, when you call

```
eigen(A)
```

you get back a list, containing the eigenvalues of the matrix $A$ (in a vector), and its eigenvectors (in a matrix), and this is both a very fast and a very reliable calculation. If your matrix has very special structure (e.g., it’s sparse, meaning almost all its entries are zero), there are more specialized packages adapted to your needs, but we don’t pursue this further here; for most data-analytic purposes, ordinary `eigen` will do.

3.2 Example

```r
> n = 100
> x1 = rnorm(n)
> x2 = rnorm(n)
> x3 = (x1+x2)/2
> y = 5 + 2*x1 + 4*x2 + rnorm(n)
> out = lm(y ~ x1 + x2 + x3)
> summary(out)
```

```
Coefficients: (1 not defined because of singularities)
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  5.14771   0.09667   53.25  <2e-16 ***
       x1  1.98474   0.09544   20.80  <2e-16 ***
       x2  3.97844   0.08854   44.93  <2e-16 ***
       x3     NA       NA       NA       NA

> one = rep(1,n)
> X = cbind(one,x1,x2,x3)
> G = t(X) %*% X
> tmp = eigen(G,symmetric=TRUE)
```
> names(tmp)
[1] "values"  "vectors"
> round(tmp$values,5)
[1] 194.00958 100.09923 95.29363 0.00000
>
> out = lm(y ~ x1 + x2)
> summary(out)

Coefficients:

|                | Estimate | Std. Error | t value | Pr(>|t|) |
|----------------|----------|------------|---------|----------|
| (Intercept)    | 5.14771  | 0.09667    | 53.25   | <2e-16   ***|
| x1             | 1.98474  | 0.09544    | 20.80   | <2e-16   ***|
| x2             | 3.97844  | 0.08854    | 44.93   | <2e-16   ***|


X = cbind(one,x1,x2)
> G = t(X) %*% X
> tmp = eigen(G,symmetric=TRUE)
> round(tmp$values,5)

4 Ridge Regression

The real problem with collinearity is that when it happens, there isn’t a unique solution to the estimating equations. There are rather infinitely many solutions, which all give the minimum mean squared error. This causes the variance of \( \hat{\beta} \) to be infinite.

One solution (which will also help us with high-dimensional regression) is called \textit{ridge regression}. Instead of minimizing

\[
\frac{1}{n} (Y - X\hat{\beta})^T (Y - X\hat{\beta})
\]

we instead minimize the penalized squared error

\[
\frac{1}{n} (Y - X\hat{\beta})^T (Y - X\hat{\beta}) + \frac{\lambda}{n} \beta^T \beta.
\]

The \textbf{penalty factor} \( \lambda > 0 \) will lead to a solution with some bias but it reduces the variance. In particular, it solves the problem of non-invertibility. We’ll come back later to how to pick \( \lambda \). The gradient is

\[
\nabla_\beta \left( \frac{1}{n} (Y - X\hat{\beta})^T (Y - X\hat{\beta}) + \frac{\lambda}{n} \beta^T \beta \right) = 2 \left( -X^T Y + X^T X \hat{\beta} + \lambda \beta \right).
\]

Set this to zero at the optimum, \( \hat{\beta}_\lambda \),

\[
X^T Y = (X^T X + \lambda I) \hat{\beta}_\lambda
\]
and solve to get
\[ \hat{\beta}_\lambda = (X^T X + \lambda I)^{-1} X^T Y. \]

The inverse always exists.

Let’s compute the mean and variance:

\[ \mathbb{E} \left[ \hat{\beta}_\lambda \right] = (X^T X + \lambda I)^{-1} X^T \mathbb{E} [Y] = (X^T X + \lambda I)^{-1} X^T X \beta \]  

(6)

\[ \text{Var} \left[ \hat{\beta}_\lambda \right] = \text{Var} \left[ (X^T X + \lambda I)^{-1} X^T Y \right] = \text{Var} \left[ (X^T X + \lambda I)^{-1} X^T \epsilon \right] = (X^T X + \lambda I)^{-1} X^T \sigma^2 I X (X^T X + \lambda I)^{-1}. \]  

(7)

Notice how both of these expressions smoothly approach the corresponding formulas ones for ordinary least squares as \( \lambda \to 0 \). Indeed, under the Gaussian noise assumption, \( \hat{\beta}_\lambda \) actually has a Gaussian distribution with the given expectation and variance.

It can be shown that ridge regression can also be obtained by doing a constrained minimization:

\[ \text{minimize} \quad (Y - Xb)^T (Y - Xb) \quad \text{subject to } \|b\|^2 \leq c. \]

You prove this using Lagrangian multipliers that you learned in calculus.

We usually choose \( \lambda \) using cross-validation which we will explain later in the course.

**Units and Standardization.** If the different predictor variables don’t have physically comparable units it’s a good idea to standardize them first, so they all have mean 0 and variance 1. Otherwise, penalizing \( \beta^T \beta = \sum_{i=1}^p \beta_i^2 \) seems to be adding up apples, oranges, and the occasional bout of regret. (Some people always pre-standardize the predictors.)

### 4.1 Ridge Regression in R

There are several R implementations of ridge regression; the MASS package contains one, \texttt{lm.ridge}, which needs you to specify \( \lambda \). The ridge package has \texttt{linearRidge}, which gives you the option to set \( \lambda \), or to select it automatically via cross-validation.

### 4.2 Other Penalties/Constraints

Ridge regression penalizes the mean squared error with \( \|b\|^2 \), the squared length of the coefficient vector. This suggests the idea of using some other measure of how big the vector is, some other norm. A mathematically popular family of norms are the \( \ell_p \) norms defined as

\[ \|b\|_p = \left( \sum_{i=1}^p |b_i|^p \right)^{1/p}. \]
The usual Euclidean length is $\ell_2$, while $\ell_1$ is

$$||b||_1 = \sum_{i=1}^{p} |b_i|$$

and (by continuity $||b||_0$ is just the number of non-zero entries in $b$. When $p \neq 2$, penalizing the $||b||_q$ does not, usually, have a nice closed-form solution like ridge regression does. Finding the minimum of the mean squared error under an $\ell_1$ penalty is called lasso regression or the lasso estimator, or just the lasso. This has the nice property that it gives sparse solutions — it sets coefficients to be exactly zero (unlike ridge). There are no closed forms for the lasso, but there are efficient numerical algorithms. The lasso is one of the most popular methods for high-dimensional regression today. We will discuss the lasso in detail later.

Penalizing $\ell_0$, the number of non-zero coefficients, sounds like a good idea, but there are, provably, no algorithms for quickly trying all possible combinations of variables. (The problem is NP hard.)

4.3 High-Dimensional Regression

One situation where we know that we will always have multicollinearity is when $n < p$. After all, $n$ points always define a linear subspace of (at most) $n-1$ dimensions. When the number of predictors we measure for each data point is bigger than the number of data points, the predictors have to be collinear, indeed multicollinear. We are then said to be in a high-dimensional regime.

This is an increasingly common situation in data analysis. A very large genetic study might sequence the genes of, say, 500 people — but measure 500,000 genetic markers in each person. If we want to predict some characteristic of the people from the genes (say their height, or blood pressure, or how quickly they would reject a transplanted organ), there is simply no way to estimate a model by ordinary least squares. Any approach to high-dimensional regression must involve either reducing the number of dimensions or penalizing the estimates such as ridge regression or the lasso.

We will discuss high-dimensional regression in more detail later. Our main tools will be: the lasso, ridge regression and something called forward stepwise regression.

4.4 Example

Let’s apply ridge regression to the simulated data already created, where one predictor variable ($X_3$) is just the average of two others ($X_1$ and $X_2$).

```r
library(ridge)
out = linearRidge(y ~ x1 + x2 + x3 + x4,lambda="automatic")
coefficients(out)
```

```
## (Intercept)   x1    x2    x3    x4
## 3.755564075 0.419326727 0.035301803 0.495563414 0.005749006
```

I had trouble installing the `ridge` package. There are other packages you can use too. For example
library(MASS)
out = lm.ridge(y ~ x1 + x2 + x3, lambda=.1)
> coefficients(out)
          x1         x2         x3
4.7644327  1.8683289 -0.5579901  1.0366070
>
You can also use the glmnet package:

library(glmnet)
lambdas = 10^seq(3, -2, by = -.1)
X = cbind(x1,x2,x3)
cvfit = cv.glmnet(X, y, alpha = 0, lambda = lambdas)
### Note: for this function you need to put the covariates in a matrix.
plot(cvfit)  ##this shows the estimated MSE as a function of lambda
bestlambda = cvfit$lambda.min  ### this is the best lambda
print(bestlambda)
## [1] 0.01258925
out = glmnet(X,y,alpha=0,lambdas,bestlambda)
coefficients(out)

(Intercept)  5.368734
x1          1.468612
x2         -0.511296
x3          1.310621