Multiple Linear Regression

-Applied Analytics-

APM Chapter 6.1 & 6.2, ISLR Chapter 3.1, 3.2

Lecturer: Darren Homrighausen, PhD

Multiple linear regression: Notation

Recall: For regression, squared-error is the usual loss function

Specify the model: $f(X) = \beta_0 + X^\top \beta = \beta_0 + \sum_{j=1}^{p} x_j \beta_j$
(This means that we think the relationship is approximately linear in X)

Then we recover the usual linear regression formulation

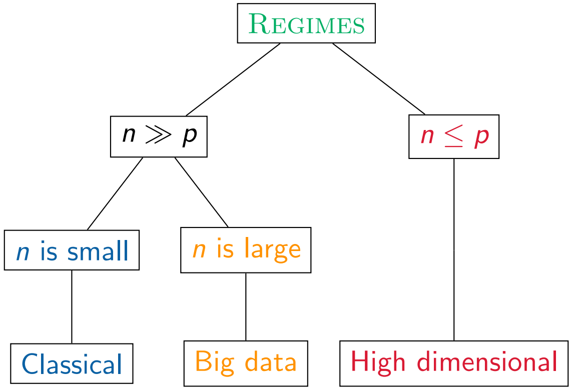
$$\mathbb{X} = \begin{bmatrix} x_1 & \cdots & x_p \end{bmatrix} = \begin{bmatrix} X_1^\top \\ X_2^\top \\ \vdots \\ X_n^\top \end{bmatrix} \text{ is } n \times p \text{ and } \mathbb{Y} = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} \text{ is length } n$$

Commonly, a column $x_0^\top = \underbrace{(1, \ldots, 1)}_{n \text{ times}}$ is included
(This encodes an intercept term, with intercept parameter β_0)

We could (should?) seek to find a β such that $\mathbb{Y} \approx \mathbb{X}\beta$

Turning these ideas into procedures

Back to the three regimes of interest, assuming $\mathbb{X} \in \mathbb{R}^{n \times p}$



Linear regression is useful in the Classical and very useful in the Big data cases

Example: Biometrics

Example

Suppose we have 4 subjects in an experiment & we record

- BMI
- minutes spent exercising in the last 7 days

We want to predict each subject's resting heart rate

The classic linear model would model the regression function as

$$f(X) = \beta_0 + \beta_1 \mathrm{BMI} + \beta_2 \mathrm{exercise}$$

In other words, we are modeling the population mean of the supervisor as a linear function of the features

Example

Under this model, the feature matrix and supervisor vector look like

$$\mathbb{X} = \begin{bmatrix} x_0 & x_1 & x_2 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 21 & 92 \\ 1 & 17 & 12 \\ 1 & 29 & 306 \\ 1 & 25 & 53 \end{bmatrix}}_{\text{int. BMI} \quad \text{exercise}}$$

and

$$\mathbb{Y} = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_4 \end{bmatrix} = \begin{bmatrix} 72 \\ 47 \\ 82 \\ 64 \end{bmatrix}$$

End example

Multiple linear regression: Notation

Using the training data, we can translate the model:

$$f(X) = \beta_0 + \sum_{j=1}^{p} x_j \beta_j$$

into an equation involving the feature matrix $\mathbb{X}$ and supervisor vector $\mathbb{Y}$

$$\mathbb{Y} = \mathbb{X}\beta + \epsilon$$

where ϵ is a random error term

(You can think of it like the irreducible error and hence all the structure is in $\mathbb{X}\beta$)

We need to use the data to form an estimate of β

Classical least squares: Minimize the training/apparent error over all β

We will notate this solution $\hat{\beta}_{LS}$

Multiple linear regression: Properties of $\hat{\beta}_{LS}$

The most prominent property of $\hat{\beta}_{LS}$ is that it

- is unbiased as an estimator of β
- has the lowest variance of any unbiased estimator of β

(These properties sound great, but remember the bias^2/variance trade-off)

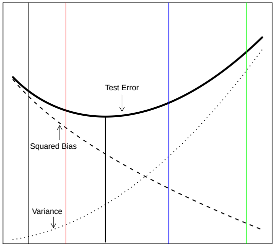


Figure: Model Complexity $\nearrow$

Estimation and uses

We can estimate the coefficients in practice in three main ways in R

```
lmOut    = lm(heartRate ~ bmi + exercise)
dataFull = cbind(heartRate, bmi, exercise)
lmOut    = train(heartRate ~ ., data = dataFull, method = "lm")
X        = cbind(bmi, exercise)
lmOut    = train(x = X, y = heartRate, method = "lm")
```

In either case, we produce the estimates $\hat{\beta}_{LS}$ along with other information

what we do with it depends on our goal:

- Prediction. Use the model to predict new data
- Inference. Estimate a relationship between the supervisor and particular features

  (We will return to inference when we talk about feature selection)

Prediction

Making predictions

Prediction is all about observing a new set of feature values $\mathbb{X}_{test}$ but not observing/using the corresponding supervisor $\mathbb{Y}_{test}$

We want to use our trained model $\hat{f}$ to make predictions via $\hat{Y} = \hat{f}(X)$ for each test X

In multiple regression, this looks like

$$\hat{Y}_{test} = \mathbb{X}_{test}\hat{\beta}_{LS}$$

In R, getting predictions on a test set looks like

```
YhatTest = predict(lmOut, newdata = Xtest)
```

Making predictions

How do we know the predictions are any good?

We want to make sure our model seems sensible given the data

Right now, we will refer to three ideas:

- Residual plots
- Computing the test error (or an estimate of it e.g. CV or even the apparent error)
- R^2

Residuals

Looking back at the model statement:

$$\mathbb{Y} = \mathbb{X}\beta + \epsilon \leftrightarrow \mathbb{Y} - \mathbb{X}\beta = \epsilon$$

If we have captured most of the predictable structure, then $\mathbb{Y} - \mathbb{X}\beta$ should look mostly like 'random noise'

(Note that here we are referring to $\mathbb{X}$ and $\mathbb{Y}$ as the training data)

Problem: We don't know β! $\rightarrow$ estimate it with $\hat{\beta}_{LS}$:

$$\text{residuals} = \mathbb{Y} - \mathbb{X}\hat{\beta}_{LS}$$

This difference is known as the residuals and making a residual plot helps us check if we have captured the structure

Residual plots

We can form the residuals by getting the predictions on the training data and comparing them to the supervisor $\mathbb{Y}$

(The training predictions are often called the fitted values)

$$\hat{Y} = \mathbb{X}\hat{\beta}_{LS} \rightarrow \text{residuals} = \mathbb{Y} - \hat{Y}$$

In R, we can get the training predictions similar to test predictions

```
Yhat         = predict(lmOut, newdata = X)
residuals    = Y - Yhat
```

We can make a plot of residuals vs. Yhat to form the residual plot

```
residualPlotData = data.frame(residuals, Yhat)
ggplot(data = residualPlotData) +
    geom_point(aes(x = Yhat, y = residuals)) +
    geom_hline(yintercept = 0, color = 'red')
```

Residual plots examples

Computing the test error (or an estimate)

Each of the following are estimates of the test loss (or test error)

- The apparent/training error: $\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{f}(X_i))^2$

  ```
  mean( (Y - Yhat)^2 )
  ```

  (Two points: 1. this is the mean of the squared residuals 2. sometimes the mean is reported (like here), other times the sum (slide 9 in 'measuringPerformance'))
- The K-Fold CV (here $K = 5$)

  ```
  trControl = trainControl(method = 'cv', number = 5)
  train(x = X, y = Y, method = "lm", trControl = trControl)
  CVloss = mean(lmOut$resample$RMSE**2)
  ```

  (we could also readily compute this via lines 96-103 in 'caretPackageIntro.Rmd')
- If we have the test supervisors, we can get a finite sample approximation to the test error: $\frac{1}{n_{test}}\sum_{i=1}^{n_{test}}(Y_i - \hat{f}(X_i))^2$

  ```
  mean( (Ytest - YhatTest)^2 )
  ```

Coefficient of determination: R^2

We can look at the R^2 of the model to assess fit

```
SSE = sum( (Yhat - Y)**2 )
TSS = sum( (Y - mean(Y) )**2 )
1 - SSE/TSS
[1] 0.5036201
```

Remembering that R^2 depends on the apparent error, we can get a CV version instead:

```
lmOut$results
  intercept      RMSE   Rsquared     MAE    RMSESD RsquaredSD MAESD
1      TRUE    1.4535    0.5029   1.11454    0.0795 0.0489 0.0607
```