

Linear Regression

Max Turgeon

DATA 2010—Tools and Techniques in Data Science

Lecture Objectives

- Build a model using linear regression
- Explain how parameters can be estimated
- Explore the flexibility of linear regression

Motivation

- Over the last few lectures we discussed:
 - What are scores and how they can be used for ranking.
 - How to build models and validate them.
- For the rest of the semester, we will look at different methodologies for building models.
 - And we start with **linear regression**.

Disclaimer

- You've probably seen linear regression in other courses.
 - E.g. STAT 1150 and 2150
- In statistics, linear regression is typically presented as a way to study correlation and/or differences in means.
 - I.e. it is used for inference
- In this course, we will focus on linear regression as a *prediction model*.
 - Therefore, we won't focus on statistical assumptions or model fit.

General notation

- Linear regression estimates the relationship between a single variable Y , called the *outcome variable*, and a series of variables X_1, \dots, X_p , called *covariates*.
 - Machine learning uses the terms “target” and “features”, respectively.
- The outcome Y is typically a continuous variable.
 - Eg. Height, income, blood pressure, etc.
- The covariates X_1, \dots, X_p can be anything.
- We want to collect all variables Y, X_1, \dots, X_p on the same unit of observation (e.g. person, school, animal, olive oil).

Multiple Linear Regression i

- The linear regression equation is

$$E(Y \mid X_1, \dots, X_p) = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p.$$

- Some authors also write the following equation:

$$Y = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p + \epsilon.$$

- Here, ϵ is a random variable with mean 0 and variance σ^2 .
 - You can use either equation; I prefer the first one.

Multiple Linear Regression ii

- In matrix notation, we have

$$E(Y \mid \mathbf{X}) = \beta^T \mathbf{X},$$

where

$$\beta = (\beta_0, \beta_1, \dots, \beta_p),$$

$$\mathbf{X} = (1, X_1, \dots, X_p).$$

Least-Squares Estimation

- Let Y_1, \dots, Y_n be a random sample of size n , and let $\mathbf{X}_1, \dots, \mathbf{X}_n$ be the corresponding sample of covariates.
- We will write \mathbb{Y} for the vector whose i -th element is Y_i , and \mathbb{X} for the matrix whose i -th row is \mathbf{X}_i .
- The Least-Squares estimate $\hat{\beta}$ is given by

$$\hat{\beta} = (\mathbb{X}^T \mathbb{X})^{-1} \mathbb{X}^T \mathbb{Y}.$$

Fitted values and residuals i

- After we have estimated the regression coefficients $\beta_0, \beta_1, \dots, \beta_p$, we can compute **fitted values** and **residuals**.
- We will use the hat notation to indicate that a parameter has been estimated:
 - β_0 is the (population) parameter.
 - $\hat{\beta}_0$ is the estimate from linear regression.
- Now assume we have our estimates $\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_p$. For a given observation in our dataset, we also have a set of covariate values X_{i1}, \dots, X_{ip} .

Fitted values and residuals ii

- We get the i -th **fitted value** by plugging all these values in the regression equation:

$$\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_{i1} + \cdots + \hat{\beta}_p X_{ip}.$$

- In matrix notation:

$$\hat{\mathbf{Y}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y} \mathbf{X}.$$

- We get the i -th **residual** by taking the difference between the observed value Y_i and the fitted value \hat{Y}_i :

$$\hat{e}_i = Y_i - \hat{Y}_i.$$

Simplest linear regression

- The simplest linear regression only has an outcome variable Y , no covariates.
 - It's equivalent to a one-sample t-test.
 - It's often a good “baseline model” against which you can compare a more sophisticated model.
- The linear regression equation can be written as

$$E(Y) = \beta_0.$$

- In other words, we are saying the population mean of Y (i.e. $E(Y)$) is equal to a parameter β_0 .

Example i

```
library(tidyverse)
url <- paste0("https://web.stanford.edu/~hastie/",
              "ElemStatLearn/datasets/prostate.data")
dataset <- read.table(url)

# Separate train and test
data_train <- filter(dataset, train)
data_test <- filter(dataset, !train)
```

Example ii

```
# Use function lm
fit1 <- lm(lpsa ~ 1, data = data_train)
fit1

##
## Call:
## lm(formula = lpsa ~ 1, data = data_train)
##
## Coefficients:
## (Intercept)
##          2.452
```

Example iii

```
# Getting predicted values
```

```
pred_vals1 <- predict(fit1, newdata = data_test)  
head(pred_vals1)
```

```
## 7 9 10 15 22 25
```

```
## 2.452345 2.452345 2.452345 2.452345 2.452345  
2.452345
```

One continuous covariate i

- Next we look at the case of a single *continuous* covariate X
- The linear regression equation for this situation can also be written as

$$E(Y|X) = \beta_0 + \beta_1 X.$$

- Let's unpack this:
 - When $X = 0$, the RHS still simplifies to β_0 , and we get

$$E(Y|X = 0) = \beta_0.$$

One continuous covariate ii

- Let's compare two values of X that differ by 1 unit, e.g. x and $x + 1$. We have

$$\begin{aligned}E(Y|X = x) &= \beta_0 + \beta_1 x \\E(Y|X = x + 1) &= \beta_0 + \beta_1(x + 1) \\&= (\beta_0 + \beta_1 x) + \beta_1 \\&= E(Y|X = x) + \beta_1.\end{aligned}$$

One continuous covariate iii

- Rearranging, we get

$$\beta_1 = E(Y|X = x + 1) - E(Y|X = x).$$

- In other words, β_1 represents the **difference** in population means between two subgroups that differ by 1 unit in their value of the covariate X .
- β_0 still represents the population mean of Y when $X = 0$.
 - But depending on what X represent (e.g. age, cholesterol), $X = 0$ may not be possible!

Example

```
# Use function lm
fit2 <- lm(lpsa ~ age, data = data_train)
fit2

##
## Call:
## lm(formula = lpsa ~ age, data = data_train)
##
## Coefficients:
## (Intercept)          age
##      0.07945      0.03665
```

Exercise

Compute the RMSE for this model (i.e. `lpsa ~ age`), and compare to the RMSE we get from only using the intercept. Which model performs better?

Solution i

```
# Recall from previous lecture
```

```
target <- data_test$lpsa
```

```
rmse1 <- sqrt(mean((target - pred_vals1)2))
```

```
rmse1
```

```
## [1] 1.027975
```

```
pred_vals2 <- predict(fit2, newdata = data_test)
```

```
rmse2 <- sqrt(mean((target - pred_vals2)2))
```

```
c(rmse1, rmse2)
```

```
## [1] 1.027975 1.062179
```

The second model (with age as a feature) performs slightly worse...

Improving the model

- There are various ways of trying to improve the model:
 - Add more features that can explain the remaining variation.
 - Model the effect of age non-linearly.
 - Regularization (next lecture).
- We will look at these approaches one at a time.

Multiple features i

- The dataset contains 8 covariates that we can use in our model.
 - Some are continuous (`lcavol`, `lweight`, `age`, `lbph`, `lcp`, `pgg45`).
 - Some are categorical (`svi`, `gleason`).
- We will add all of them in the model.

```
# Remove column train
data_train <- select(data_train, -train)
# The dot is a short-hand for all variables
fit3 <- lm(lpsa ~ ., data = data_train)
fit3
```

Multiple features ii

```
##  
## Call:  
## lm(formula = lpsa ~ ., data = data_train)  
##  
## Coefficients:  
## (Intercept) lcavol lweight age lbph svi  
## 0.429170 0.576543 0.614020 -0.019001 0.144848  
0.737209  
## lcp gleason pgg45  
## -0.206324 -0.029503 0.009465
```


Multiple features iii

```
pred_vals3 <- predict(fit3, newdata = data_test)
rmse3 <- sqrt(mean((target - pred_vals3)^2))
c(rmse1, rmse2, rmse3)
```

```
## [1] 1.0279753 1.0621793 0.7219931
```

- This was a huge improvement: we reduced the RMSE by almost 30%.
- Note: **gleason** has four levels, but it only has 1 regression coefficient...
 - We would expect 3 coefficients.

Multiple features iv

- We can fix that by transforming `gleason` from integer to factor.
 - Let's see if that leads to any improvement.

```
# Turn gleason into factor
```

```
data_train <- mutate(data_train,  
                      gleason = factor(gleason))
```

```
fit4 <- lm(lpsa ~ ., data = data_train)  
fit4
```

Multiple features v

```
##  
## Call:  
## lm(formula = lpsa ~ ., data = data_train)  
##  
## Coefficients:  
## (Intercept) lcavol lweight age lbph svi  
## 0.19622 0.56811 0.64616 -0.02218 0.13193  
0.70716  
## lcp gleason7 gleason8 gleason9 pgg45  
## -0.24550 0.18307 0.72688 -0.49720 0.01105
```

Multiple features vi

```
# gleason needs to be a factor in the test data too!  
data_test <- mutate(data_test,  
                     gleason = factor(gleason))  
pred_vals4 <- predict(fit4, newdata = data_test)  
rmse4 <- sqrt(mean((target - pred_vals4)^2))  
c(rmse1, rmse2, rmse3, rmse4)
```

```
## [1] 1.0279753 1.0621793 0.7219931 0.7424015
```

- It leads to a slightly worse model, but the effect was marginal.

Exercise

Consider a model with all covariates except **gleason**. Compute its RMSE. Does it perform better or worse than the model with all 8 covariates?

Solution i

```
fit5 <- lm(lpsa ~ lcavol + lweight + age + lbph +  
          svi + lcp + pgg45, data = data_train)
```

```
pred_vals5 <- predict(fit5, newdata = data_test)  
rmse5 <- sqrt(mean((target - pred_vals5)^2))
```

```
c(rmse1, rmse2, rmse3, rmse4, rmse5)
```

```
## [1] 1.0279753 1.0621793 0.7219931 0.7424015  
0.7186887
```

- It performs slightly better, but the difference is minimal.

Splines

Motivation

- Recall the linear regression equation:

$$E(Y \mid X_1, \dots, X_p) = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p.$$

- We implicitly assume that the X_i are different measurements/features.
 - They more or less match the columns of the dataset.
- However, we can also include transformations of the same covariate (e.g. age and its square).
- For example, we can perform *cubic regression* within the framework of linear regression as follows:

$$E(Y \mid X) = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3.$$

Example i

```
# poly gives us the polynomial expansion  
head(poly(data_train$age, 3))
```

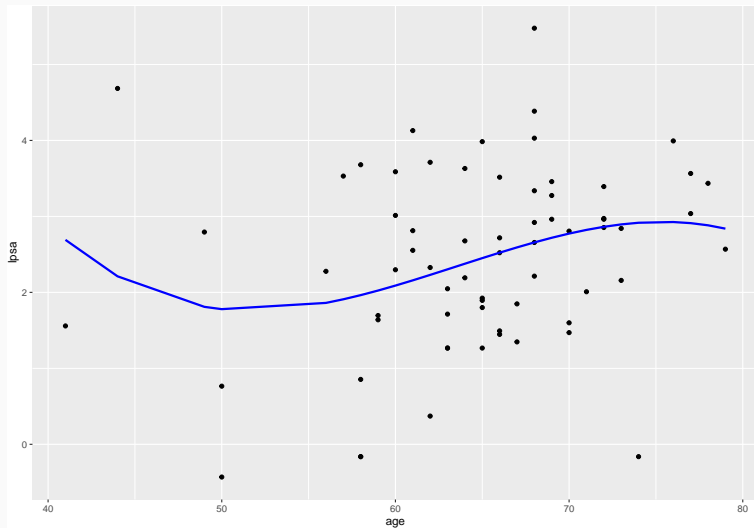
```
##              1              2              3  
## [1,] -0.24194745  0.10877056  0.189860637  
## [2,] -0.11068851 -0.06639027  0.138164924  
## [3,]  0.15182937  0.11396035 -0.009135548  
## [4,] -0.11068851 -0.06639027  0.138164924  
## [5,] -0.04505904 -0.08763665  0.027096941  
## [6,] -0.24194745  0.10877056  0.189860637
```

Example ii

```
fit_cube <- lm(lpsa ~ poly(age, 3), data = data_train)

data_train |>
  bind_cols(.fitted = fitted(fit_cube)) |>
  ggplot(aes(x = age)) +
  geom_point(aes(y = lpsa)) +
  geom_line(aes(y = .fitted),
            colour = "blue", size = 1)
```

Example iii



Splines i

- Polynomial regression is simple, but its main drawback is that it requires a global fit.
 - The resulting cubic polynomial is the same over the whole range of values.
- **Splines** increase the flexibility by fitting polynomial over subsets of the range.
 - E.g. a certain cubic polynomial on the range $[0, 10]$, and a different cubic polynomial on the range $[10, 20]$.
- They also have to satisfy extra constraints.
 - They have to be continuous (i.e. the polynomials have to match on the boundary points).

- They have to be differentiable everywhere (so no sharp corners on the boundary points).
- However, they require many degrees of freedom.
 - Assume we have k knots (i.e. the boundary points).
 - On each piece, we fit polynomials of degree d .
 - Continuity/Differentiability gives use dk constraints.
 - Therefore, we need $(d + 1)(k + 1) - dk = d + k + 1$ degrees of freedom.

Example i

- You can manually specify the knots.

```
library(splines)
head(bs(data_train$age,
        knots = c(62, 65, 69),
        degree = 3))
```

Example ii

```
## 1 2 3 4 5 6
## [1,] 0.4604136 0.3013393 0.05165816 0.00000000
0.00000000 0.000
## [2,] 0.1432098 0.5017361 0.34814342 0.00000000
0.00000000 0.000
## [3,] 0.00000000 0.00000000 0.05252101 0.3301321
0.4923469 0.125
## [4,] 0.1432098 0.5017361 0.34814342 0.00000000
0.00000000 0.000
## [5,] 0.0156250 0.3281250 0.65625000 0.00000000
0.00000000 0.000
## [6,] 0.4604136 0.3013393 0.05165816 0.00000000
```

Example iii

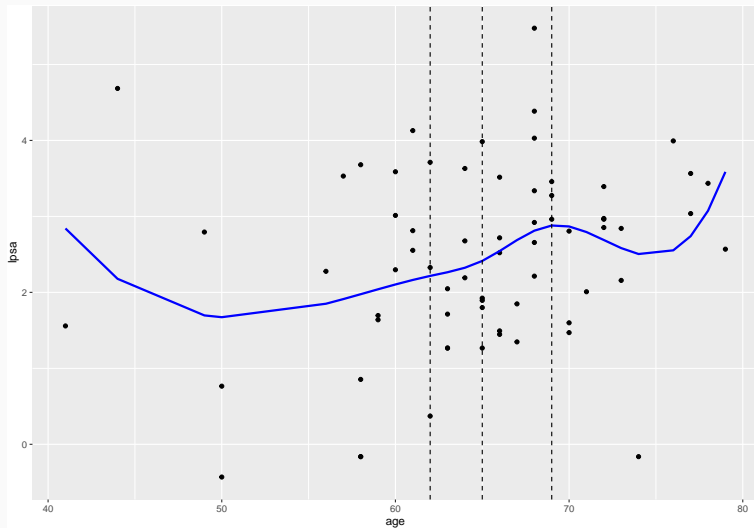
0.00000000 0.000

```
fit_spl <- lm(lpsa ~ bs(age, degree = 3,  
                        knots = c(62, 65, 69)),  
             data = data_train)
```


Example iv

```
data_train |>
  bind_cols(.fitted = fitted(fit_spl)) |>
  ggplot(aes(x = age)) +
  geom_point(aes(y = lpsa)) +
  geom_line(aes(y = .fitted),
            colour = "blue", size = 1) +
  geom_vline(xintercept = c(62, 65, 69),
            linetype = "dashed")
```

Example v



Example vi

- Or you can specify the degrees of freedom.

The intercept is the 7th df

```
spl_mat <- bs(data_train$age, df = 6)
head(spl_mat)
```

```
## 1 2 3 4 5 6
```

```
## [1,] 0.46659375 0.3127902 0.05424107
0.000000000 0.00000000 0.000
```

```
## [2,] 0.12862153 0.5024529 0.36555060
0.000000000 0.00000000 0.000
```

```
## [3,] 0.00000000 0.00000000 0.04960317
```

Example vii

```
0.333049887 0.4923469 0.125
## [4,] 0.12862153 0.5024529 0.36555060
0.000000000 0.00000000 0.000
## [5,] 0.01171875 0.3007812 0.68576389
0.001736111 0.00000000 0.000
## [6,] 0.46659375 0.3127902 0.05424107
0.000000000 0.00000000 0.000
```

```
attr(spl_mat, "knots")
```

```
## 25% 50% 75%
## 61 65 69
```

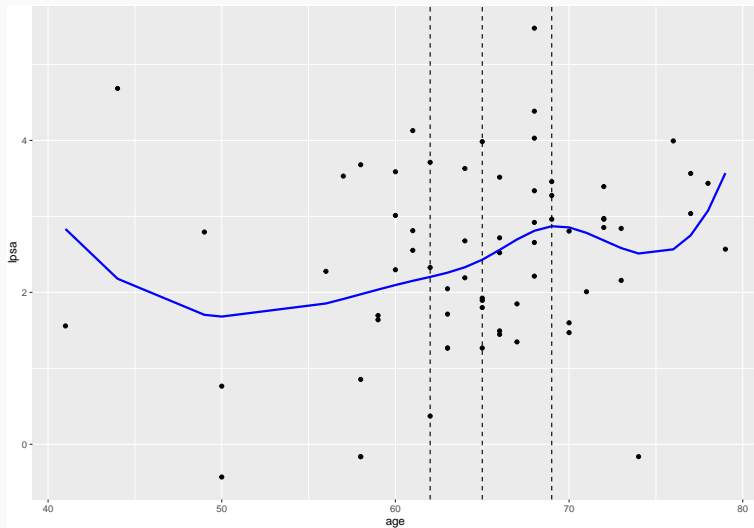
Example viii

```
# Same result  
fit_spl <- lm(lpsa ~ bs(age, df = 6),  
              data = data_train)
```

Example ix

```
data_train |>
  bind_cols(.fitted = fitted(fit_spl)) |>
  ggplot(aes(x = age)) +
  geom_point(aes(y = lpsa)) +
  geom_line(aes(y = .fitted),
            colour = "blue", size = 1) +
  geom_vline(xintercept = c(62, 65, 69),
            linetype = "dashed")
```

Example x



Exercise

Compute the RMSE for this model, and compare with the model with only age as a covariate (i.e. `fit2`).


```
pred_vals_spl <- predict(fit_spl, newdata = data_test)
rmse_spl <- sqrt(mean((target - pred_vals_spl)^2))

c(rmse2, rmse_spl)

## [1] 1.062179 1.083787
```

Final remarks

- Linear regression is a general framework for building models.
- Its main asset is its high interpretability:
 - The coefficients represent differences in expected outcome.
- It is also quite flexible, especially when combined with splines.
- However, it can be quite sensitive to outliers in the training data, and the number of degrees of freedom we can use is bounded above by the number of observations.
- Next lecture, we will discuss regularization, which tries to address both points.