Applications

Of Data Science

Topics in Regression

Applications of Data Science - Class 14

Giora Simchoni

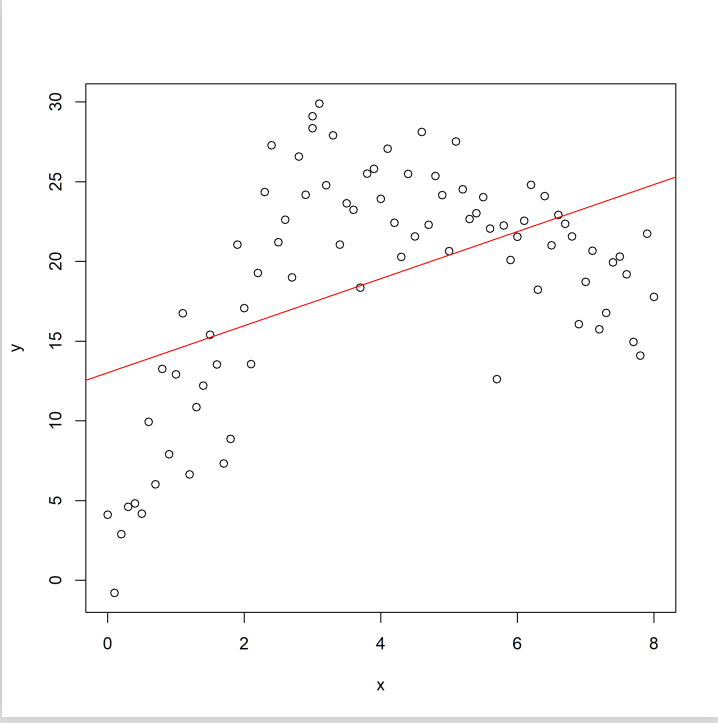
`gsimchoni@gmail.com and add #dsapps in subject`

Stat. and OR Department, TAU

2021-01-19

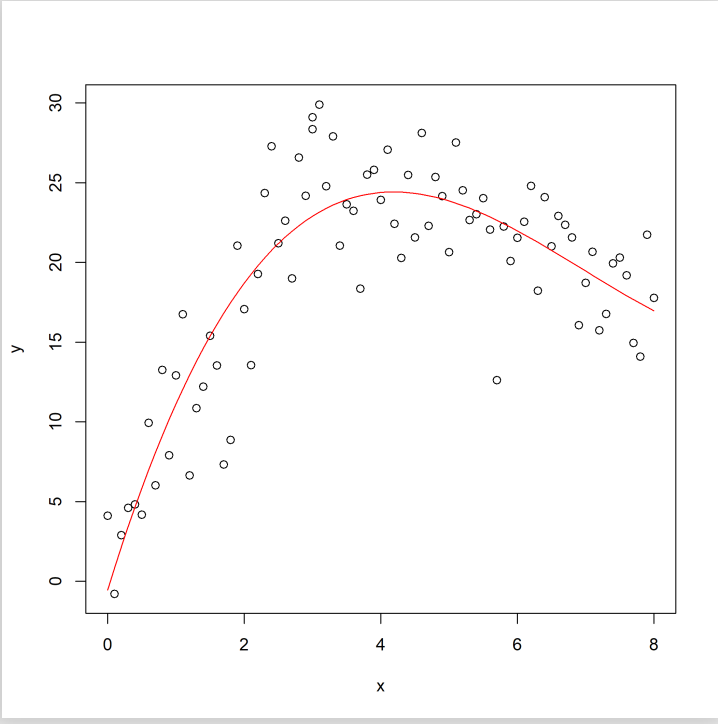
Introduction to Splines

What's the No. 1 complaint about LR?



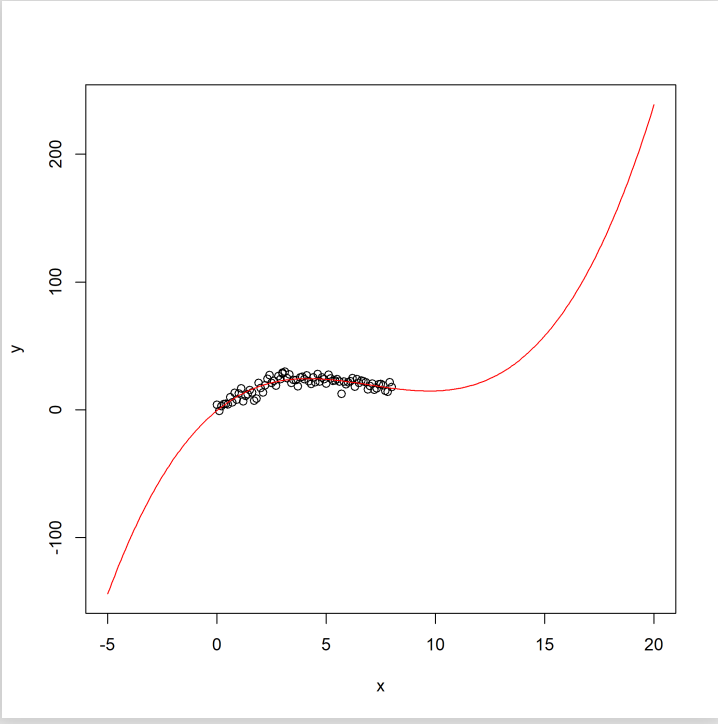
$$\hat{y} = 13 + 1.5x$$

We could use polynomial regression



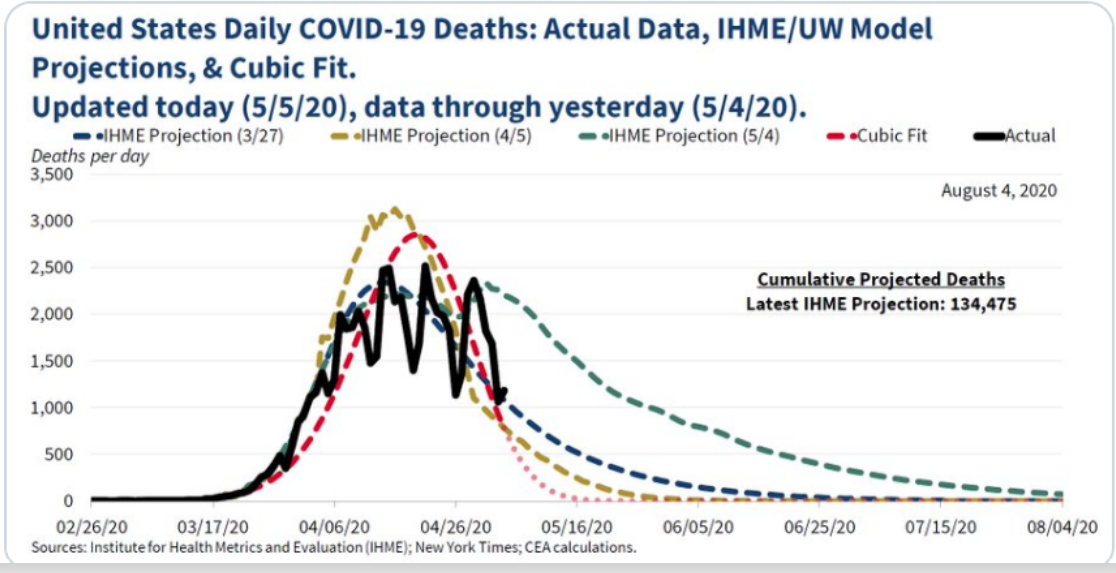
$$\hat{y} = -0.5 + 13.9x - 2.4x^2 + 0.1x^3$$

But...



CEA
@WhiteHouseCEA

To better visualize observed data, we also continually update a curve-fitting exercise to summarize COVID-19's observed trajectory. Particularly with irregular data, curve fitting can improve data visualization. As shown, IHME's mortality curves have matched the data fairly well.



BUSINESS INSIDER

Subscribe

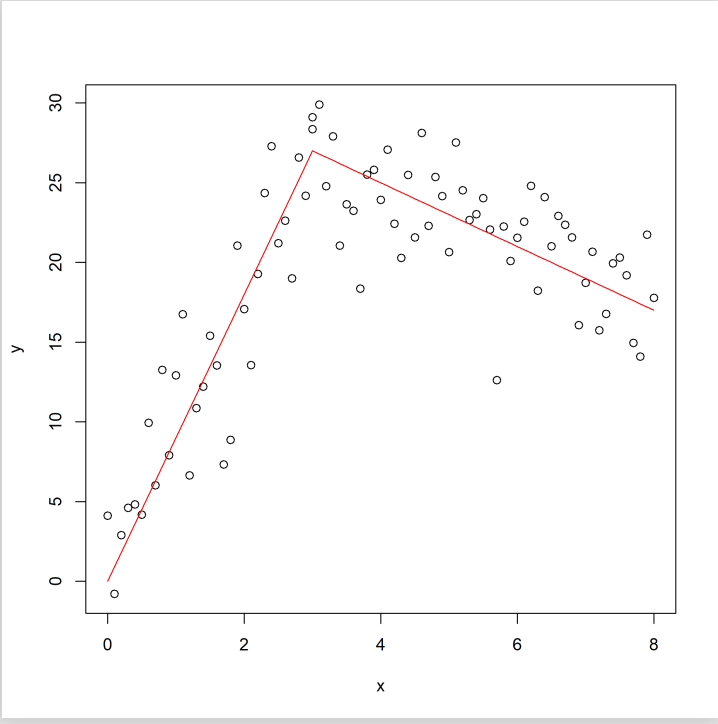
Trump's advisers released a 'beyond stupid' mathematical model of coronavirus deaths created in Excel by a controversial economist

Eliza Relman May 6, 2020, 10:13 PM

MARS

(Multivariate Adaptive Regression Splines)

It would have been much nicer if we could...

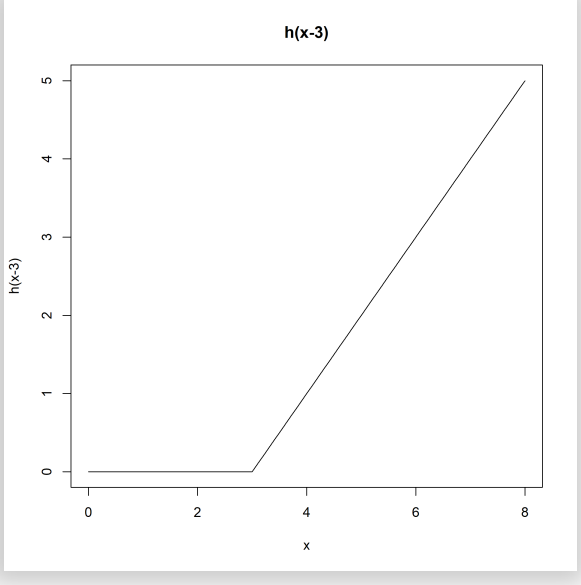


$$\hat{y} = [9x]I(x < 3) + [33 - 2x]I(x > 3) =$$
$$= 27 - 9\max(0, 3 - x) - 2\max(0, x - 3) =$$
$$27 - 9[-(x - 3)]_+ - 2[+(x - 3)]_+$$

The Hinge Function

$$h(x-c) = [(x-c)]_+ = max(0, x-c)$$

Where c would be called a knot.



If you know about Deep Learning this would be called...

MARS via `earth`

```
library(earth)

mod_mars <- earth(y ~ x, degree = 1)

summary(mod_mars)
```

```
## Call: earth(formula=y~x, degree=1)
##
##             coefficients
## (Intercept)    26.902230
## h(3.1-x)       -8.403566
## h(x-3.1)       -1.995450
##
## Selected 3 of 3 terms, and 1 of 1 predictors
## Termination condition: RSq changed by less than 0.001 at 3 terms
## Importance: x
## Number of terms at each degree of interaction: 1 2 (additive model)
## GCV 1.319067    RSS 95.37498    GRSq 0.9715939    RSq 0.9743302
```

Why on earth is the package for MARS called... 🤦

This is exactly like

```
X <- data.frame(y, h1 = pmax(0, 3.1 - x), h2 = pmax(0, x - 3.1))

head(X)
```

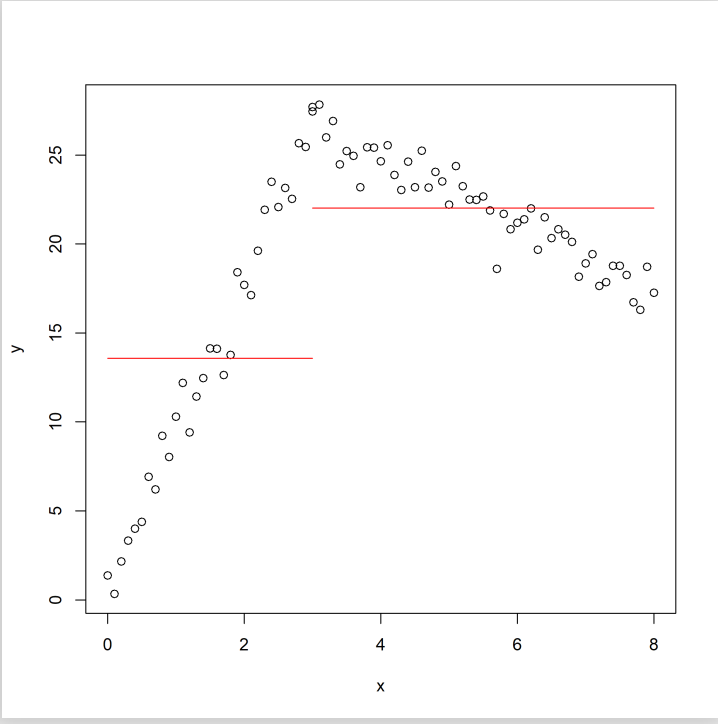
```
##           y  h1 h2
## 1 1.3709584 3.1  0
## 2 0.3353018 3.0  0
## 3 2.1631284 2.9  0
## 4 3.3328626 2.8  0
## 5 4.0042683 2.7  0
## 6 4.3938755 2.6  0
```

```
lm(y ~ h1 + h2, data = X)
```

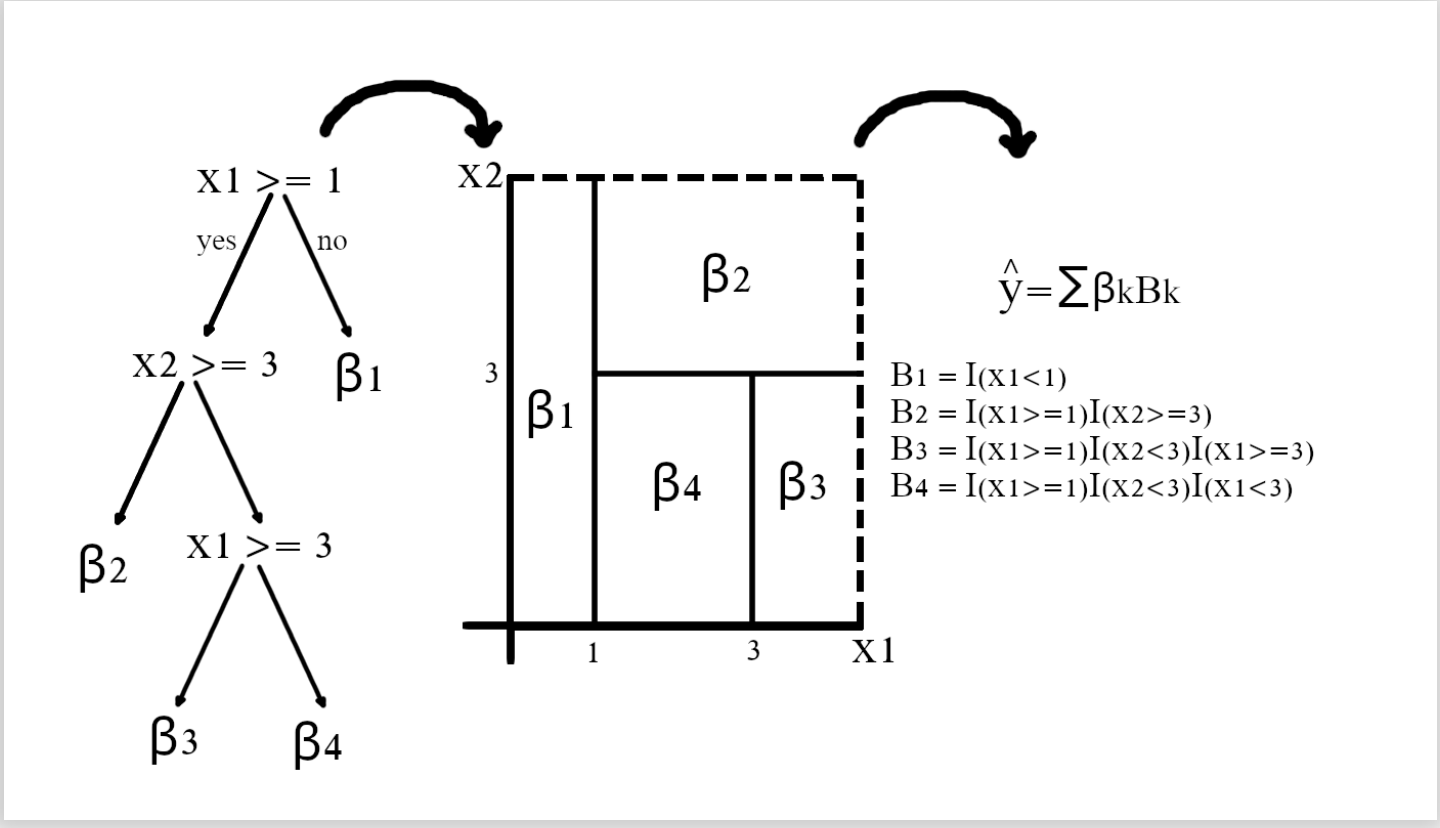
```
##
## Call:
## lm(formula = y ~ h1 + h2, data = X)
##
## Coefficients:
## (Intercept)           h1           h2
##      26.902       -8.404       -1.995
```

And which method knows how to scan X for the best partition?

What's the No. 1 complaint about Trees?



Why are CART also called "Recursive Partitioning"?



From CART to MARS

Let
$$B_m(X) = \prod_{k=1}^{K_m} I(X_{v(km)} \in R_{km}) = \prod_{k=1}^{K_m} I(\pm X_{v(km)} \mp t_{km} \geq 0)$$

Where m is the branch (leaf) number, K_m is the number of splits that gave rise to B_m, $X_{v(km)}$ is the variable at split k at branch m, and t_{km} is the split value.

Following Friedman (1991), let:

$$H(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

(a.k.a a *step* function)

So: $B_m(X) = \prod_{k=1}^{K_m} H[s_{km}(X_{v(km)} - t_{km})]$ where $s_{km} = \pm 1$

So, Each branch m of the tree is just a multiplication of step functions!

With β_m at each leaf.

And how does one decide on the next split?

"For all branches currently in the model, for all variables, for all values, find the (m^*, v^*, t^*) combination which decreases most the SSE criterion, if *from this stage* we binary split branch m^* by variable X_{v^*} on t^*"

In step function jargon:

"For all branches currently in the model, for all variables, for all values, find the (m^*, v^*, t^*) combination which decreases most the SSE criterion, if B_{m^*} will be replaced by its product with the step function $H[+(X_{v^*} - t^*)]$ and with the addition of a new branch which is the product of B_{m^*} and the reflected step function $H[-(X_{v^*} - t^*)]$"

And, $\hat{y} = g(X) = \sum_m \beta_m B_m(X)$

And, $LOF(g)$ is the Lack of Fit function for $g(X)$, typically the SSE critetion

And, Switch all our βs to αs

Then the growing of trees algorithm is really...

Recursive Partitioning as a Nested For Loop

Algorithm 1 (recursive partitioning)

$B_1(\mathbf{x}) \leftarrow 1$
For $M = 2$ to $M_{\max}$ do: lof$^* \leftarrow \infty$
  For $m = 1$ to $M - 1$ do:
    For $v = 1$ to n do:
      For $t \in \{x_{vj} | B_m(\mathbf{x}_j) > 0\}$
        $g \leftarrow \sum_{i \neq m} a_i B_i(\mathbf{x}) + a_m B_m(\mathbf{x}) H[+(x_v - t)] + a_M B_m(\mathbf{x}) H[-(x_v - t)]$
        lof $\leftarrow \min_{a_1, \ldots, a_M}$ LOF(g)
        if lof $<$ lof*, then lof$^* \leftarrow$ lof; $m^* \leftarrow m$; $v^* \leftarrow v$; $t^* \leftarrow t$ end if
      end for
    end for
  end for
  $B_M(\mathbf{x}) \leftarrow B_{m^*}(\mathbf{x}) H[-(x_{v^*} - t^*)]$
  $B_{m^*}(\mathbf{x}) \leftarrow B_{m^*}(\mathbf{x}) H[+(x_{v^*} - t^*)]$
end for
end algorithm

Why gotta be H?

Algorithm 2 (MARS—forward stepwise)

$B_1(\mathbf{x}) \leftarrow 1$; $M \leftarrow 2$
Loop until $M > M_{\max}$: lof* $\leftarrow \infty$
  For $m = 1$ to $M - 1$ do:
    For $v \notin \{v(k, m) | 1 \le k \le K_m\}$
      For $t \in \{x_{vj} | B_m(\mathbf{x}_j) > 0\}$
        $g \leftarrow \sum_{i=1}^{M-1} a_i B_i(\mathbf{x}) + a_M B_m(\mathbf{x})[+(x_v - t)]_+ + a_{M+1} B_m(\mathbf{x})[-(x_v - t)]_+$
        lof $\leftarrow \min_{a_1, \ldots, a_{M+1}}$ LOF(g)
        if lof < lof*, then lof* $\leftarrow$ lof; $m^* \leftarrow m$; $v^* \leftarrow v$; $t^* \leftarrow t$ end if
      end for
    end for
  end for
  $B_M(\mathbf{x}) \leftarrow B_{m^*}(\mathbf{x})[+(x_{v^*} - t^*)]_+$
  $B_{M+1}(\mathbf{x}) \leftarrow B_{m^*}(\mathbf{x})[-(x_{v^*} - t^*)]_+$
  $M \leftarrow M + 2$
end loop
end algorithm

More than H changed on the way to MARS

- Most important: NOT removing the parent $B_{m*}(X)$, we can always go back to the parent and split it *again*, allowing for additive models such as
  $\hat{y} = 6 + 5[-(x_1 - 4)]_+ - 2[+(x_2 - 3)]_+$

> Can you do this with a tree?

- The generalized form is actually $[\pm(x - t)]_+^q$ where q is usually 1 but *can be* 2 or 3

> What if $q = 0$?

- At each step of adding to $B_m(X)$ only variables which have not yet participated in this path are added

> 💡 How is it different from trees and why do you think this was added?

- For "growing" and "pruning" the $LOF(g)$ used by MARS is GCV (Generalized Cross Validation) which is a penalized version of SSE which is out of scope
- Easily extensible to GLM via the proper link function, e.g. for Logistic Regression: $logit(p) = \sum_k \beta_k B_k$
- And more...

"Pruning" MARS

Another advantage of MARS is that once the full model has been "grown", **each and every** term is candidate for pruning from the formula, not whole pairs:

Algorithm 3 (MARS—backwards stepwise)
$J^* = \{1, 2, \ldots, M_{\max}\}$; $K^* \leftarrow J^*$
$\text{lof}^* \leftarrow \min_{\{a_j | j \in J^*\}} \text{LOF}(\sum_{j \in J^*} a_j B_j(\mathbf{x}))$
For $M = M_{\max}$ to 2 do: $b \leftarrow \infty$; $L \leftarrow K^*$
  For $m = 2$ to M do: $K \leftarrow L - \{m\}$
    $\text{lof} \leftarrow \min_{\{a_k | k \in K\}} \text{LOF}(\sum_{k \in K} a_k B_k(\mathbf{x}))$
    if $\text{lof} < b$, then $b \leftarrow \text{lof}$; $K^* \leftarrow K$ end if
    if $\text{lof} < \text{lof}^*$, then $\text{lof}^* \leftarrow \text{lof}$; $J^* \leftarrow K$ end if
  end for
end for
end algorithm

The forward pass adds terms in pairs, but the pruning (backward pass) typically discards one side of the pair and so terms are often not seen in pairs in the final model.

Predicting OKCupid Income

```
mod_mars <- earth(income ~ ., data = okcupid2_imp_mice_train, degr

summary(mod_mars, digits = 2)
```

```
## Call: earth(formula=income~., data=okcupid2_imp_mice_train, degree=1)
##
##                              coefficients
## (Intercept)                        -0.011
## sexm                                0.062
## body_typeathletic                   0.042
## body_typejacked                     0.157
## body_typeused up                    0.251
## body_type_not_perfectTRUE          -0.058
## diet2kosher                         0.194
## diet2other                          0.065
## drinksnot at all                   -0.350
## drinksoften                        -0.235
## drinksrarely                       -0.330
## drinkssocially                     -0.272
## drinksvery often                   -0.199
## drugssometimes                     -0.049
## religion2christian                 -0.042
## education2degree2                   0.078
## education2degree3                   0.114
## education2high school               0.166
```

```
job3health                      0.07
job3legalservices               0.18
job3management                  0.22
job3marketing                   0.09
job3retired                     0.20
job3student                    -0.10
job3tech                        0.15
job3unemployed                 -0.19
orientationstraight             0.06
h(age-30)                       0.04
h(32-age)                      -0.01
h(age-32)                      -0.04
h(height_cm-203.2)              0.02
h(1.39029-essay4_len)          -0.32
h(essay4_len-1.39029)          -0.01
h(essay5_len-7.29167)           0.36
h(1.10393-essay6_len)           0.50

Selected 40 of 44 terms, and 36 of 114 predictors
Termination condition: RSq changed by less than 0.001 at 44 terms
Importance: age, body_type_not_perfectTRUE, job3hardware, job3tech, job3mana
Number of terms at each degree of interaction: 1 39 (additive model)
GCV 0.1    RSS 948    GRSq 0.2    RSq 0.3
```

```
pred_mars <- predict(mod_mars, okcupid2_imp_mice_valid)
report_rmse_and_cor(okcupid2_imp_mice_valid$income, pred_mars)
```

```
## RMSE: 0.357
## CORR: 0.481
```

Pros and Cons

Pros:

- Flexible, built-in interactions
- Highly interpretable
- Little pre-processing of predictors needed
- Handle all types of predictors (continuous, categorical)
- Feature selection built-in (but when predictors are correlated...)
- Good variance-bias tradeoff

Cons:

- Greedy
- The resulting fitted function may not be smooth (not differentiable along hinges)
- No built-in CI for variables params

B-Splines

We've seen that if we know the knot we can fit MARS via `lm()`

```
lm_manual <- lm(y ~ I(pmax(0, 3.1 - x)) + I(pmax(0, x - 3.1)))
head(model.matrix(lm_manual), 4)
```

```
##   (Intercept) I(pmax(0, 3.1 - x)) I(pmax(0, x - 3.1))
## 1           1                 3.1                   0
## 2           1                 3.0                   0
## 3           1                 2.9                   0
## 4           1                 2.8                   0
```

```
lm_manual
```

```
##
## Call:
## lm(formula = y ~ I(pmax(0, 3.1 - x)) + I(pmax(0, x - 3.1)))
##
## Coefficients:
##         (Intercept)  I(pmax(0, 3.1 - x))  I(pmax(0, x - 3.1))
##              26.902               -8.404                -1.995
```

Another function could do this, slightly different.

```
library(splines)

lm_bs <- lm(y ~ bs(x, degree = 1, knots = 3.1))
mm_bs <- model.matrix(lm_bs)
colnames(mm_bs) <- c("intercept", "base1", "base2")
head(mm_bs, 4)
```

```
##   intercept      base1 base2
## 1         1 0.00000000     0
## 2         1 0.03225806     0
## 3         1 0.06451613     0
## 4         1 0.09677419     0
```

```
lm_bs
```

```
##
## Call:
## lm(formula = y ~ bs(x, degree = 1, knots = 3.1))
##
## Coefficients:
##                       (Intercept)  bs(x, degree = 1, knots = 3.1)1
##                            0.8512                          26.0511
## bs(x, degree = 1, knots = 3.1)2
##                           16.2733
```

But wait, the `model.matrix()` looks different and so are the coefficients! And yet...

```
head(lm_manual$fit)
```

```
##         1         2         3         4         5         6
## 0.8511751 1.6915318 2.5318884 3.3722450 4.2126016 5.0529582
```

```
head(lm_bs$fit)
```

```
##         1         2         3         4         5         6
## 0.8511751 1.6915318 2.5318884 3.3722450 4.2126016 5.0529582
```

```
head(mm_bs, 5)
```

```
##   intercept      base1 base2
## 1         1 0.00000000     0
## 2         1 0.03225806     0
## 3         1 0.06451613     0
## 4         1 0.09677419     0
## 5         1 0.12903226     0
```

The `splines::bs()` will parametrize `x` slightly differently:

```
mm_bs_manual <- as.matrix(
  cbind(1,
        (x - 0) / (3.1 - 0) * ifelse(x > 0 & x <= 3.1, 1, 0) +
          (8 - x) / (8 - 3.1) * ifelse(x > 3.1 & x <= 8, 1, 0),
        (x - 3.1) / (8 - 3.1) * ifelse(x > 3.1 & x <= 8, 1, 0)))

head(mm_bs_manual, 5)
```

```
##      [,1]       [,2] [,3]
## [1,]    1 0.00000000    0
## [2,]    1 0.03225806    0
## [3,]    1 0.06451613    0
## [4,]    1 0.09677419    0
## [5,]    1 0.12903226    0
```

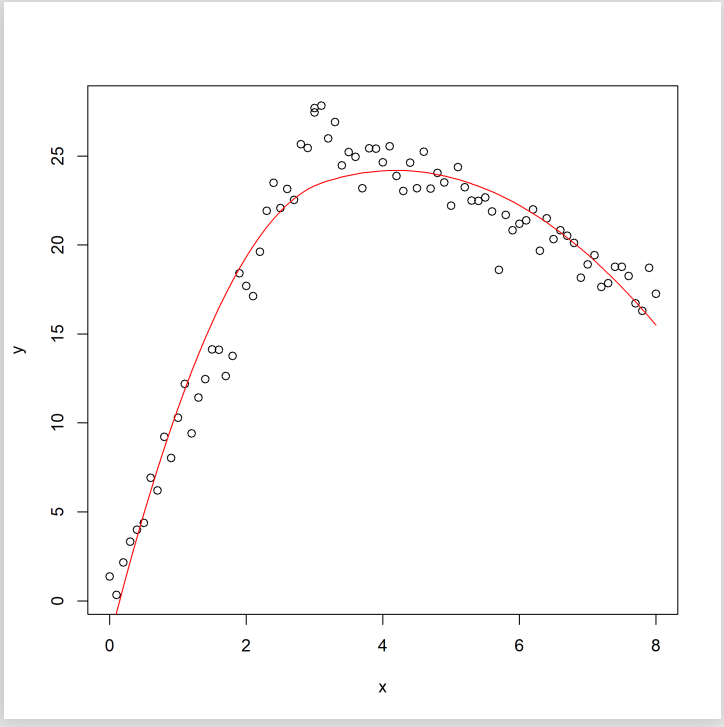
But `splines::bs()` is much more flexible

```
lm_bs2 <- lm(y ~ bs(x, degree = 2, knots = 3.1))
mm_bs2 <- model.matrix(lm_bs2)
colnames(mm_bs2) <- c("intercept", "base1", "base2", "base3")
head(mm_bs2, 4)
```

```
##   intercept      base1        base2 base3
## 1         1 0.00000000 0.0000000000     0
## 2         1 0.06307232 0.0004032258     0
## 3         1 0.12325702 0.0016129032     0
## 4         1 0.18055411 0.0036290323     0
```

```
lm_bs2
```

```
##
## Call:
## lm(formula = y ~ bs(x, degree = 2, knots = 3.1))
##
## Coefficients:
##                       (Intercept)  bs(x, degree = 2, knots = 3.1)1
##                            -2.184                           23.648
## bs(x, degree = 2, knots = 3.1)2  bs(x, degree = 2, knots = 3.1)3
##                            28.850                           17.682
```



Now that I got your attention...

Splines

We've seen a tree or MARS could be generalized to this form:

$\hat{y} = g(X) = \sum_m \beta_m B_m(X)$

In general, $B_m(X)$ are sometimes called *basis functions* or *basis expansions* of X.

> What would $B_m(X)$ be for polynomial regression?

For splines, **for a univariate** X, define a set of knots $\tau_1, \ldots, \tau_K$ in the range of X. Let each $B_k(X)$ be a polynomial of degree d in the range $\tau_{k-1} < X \leq \tau_k$ (where for the first and last knots we'll use the minimum and maximum of X for now)

And we can write

$$\hat{y} = g(X) = \sum_{k=1}^{K+1} \beta_k B_k(X) =$$

$$\sum_{k=1}^{K+1} \beta_k [\alpha_{0k} + \alpha_{1k} x + \ldots + \alpha_{dk} x^d] I(\tau_{k-1} < x \leq \tau_k)$$

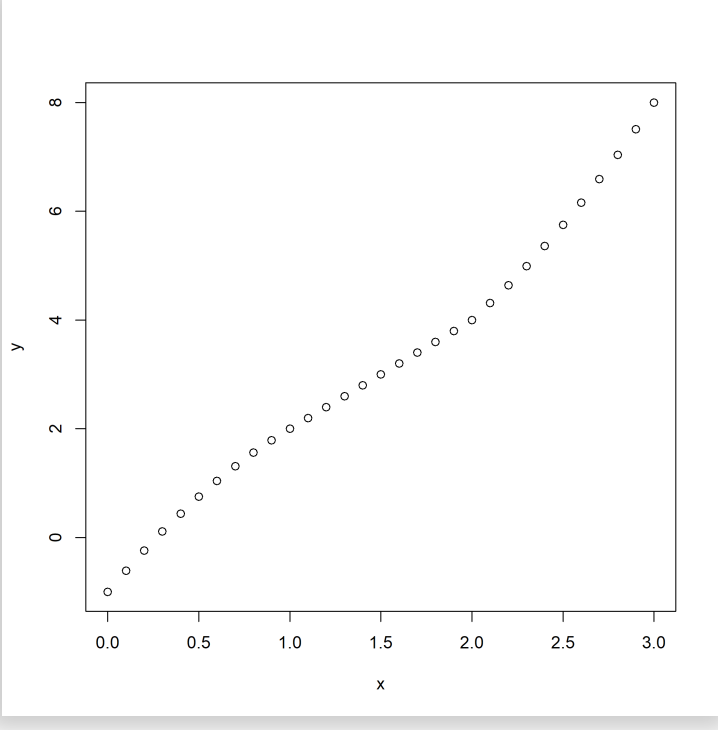
Such that $\beta_{k-1} B_{k-1}(\tau_k) = \beta_k B_k(\tau_k)$

(i.e. $g(X)$ is continuous, it is a sequence of polynomials tied together at the knots)

Spline with $K = 2$ knots, $d = 2$ and $0 < X \leq 3$:

$$B_1(X) = -1 + 4X - X^2, 0 < X \leq 1$$
$$B_2(X) = 4X, 1 < X \leq 2$$
$$B_3(X) = 2 - X + X^2, 2 < X \leq 3$$
$$y = 1 \cdot B_1(X) + \frac{1}{2} \cdot B_2(X) + 1 \cdot B_3(X)$$

```
x1 <- seq(0, 1, 0.1); x2 <- seq(1, 2, 0.1); x3 <- seq(2, 3, 0.1)
y1 <- -1 + 4 * x1 - x1^2; y2 <- 4 *x2; y3 <- 2 - x3 + x3^2
x_all <- c(x1, x2, x3)
y_all <- c(1 * y1, 0.5 * y2, 1 * y3)
plot(x_all, y_all, xlab = "x", ylab = "y")
```



But we do not have $(K+1)\cdot(d+1)$ free params! It can be shown we actually have $K\cdot d$ constraints, so $(K+1)\cdot(d+1)-K\cdot d = K+d+1$ degrees of freedom.

Our model matrix will look like this:

$$\begin{bmatrix} 1 & x & x^2 & \dots & x^d & (x-\tau_1)_+^d & (x-\tau_2)_+^d & \dots & (x-\tau_K)_+^d \end{bmatrix}$$

Starting with a "basic" polynomial of degree d, deviations from the basic polynomial are successively added to the spline function to the right of each of K knots, where remember the $(x-c)_+$ is the hinge function $max(0, x-c)$.

We see this in the no. of columns of the the matrix returned by `bs()`:

```
ncol(bs(x, degree = 1, knots = 3.1))
```

```
## [1] 2
```

```
ncol(bs(x, degree = 2, knots = 3.1))
```

```
## [1] 3
```

```
ncol(bs(x, degree = 2, knots = c(3.1, 5)))
```

```
## [1] 4
```

```
ncol(bs(x, degree = 3, knots = c(3.1, 5)))
```

```
## [1] 5
```

But the matrix itself is slightly different than what we would have expected.

```
head(mm_bs2)
```

```
##   intercept      base1        base2 base3
## 1         1 0.00000000 0.0000000000     0
## 2         1 0.06307232 0.0004032258     0
## 3         1 0.12325702 0.0016129032     0
## 4         1 0.18055411 0.0036290323     0
## 5         1 0.23496358 0.0064516129     0
## 6         1 0.28648543 0.0100806452     0
```

We saw this for $d = 1; K = 1$ and were able to reconstruct the matrix.

In general, `bs()` fits B-splines, which is a special paramterization of the simple spline defined above.

Expressions for the polynomial pieces can be derived by means of the Cox–de Boor recursion formula[6]

$$B_{i,0}(x) := \begin{cases} 1 & \text{if} \quad t_i \leq x < t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

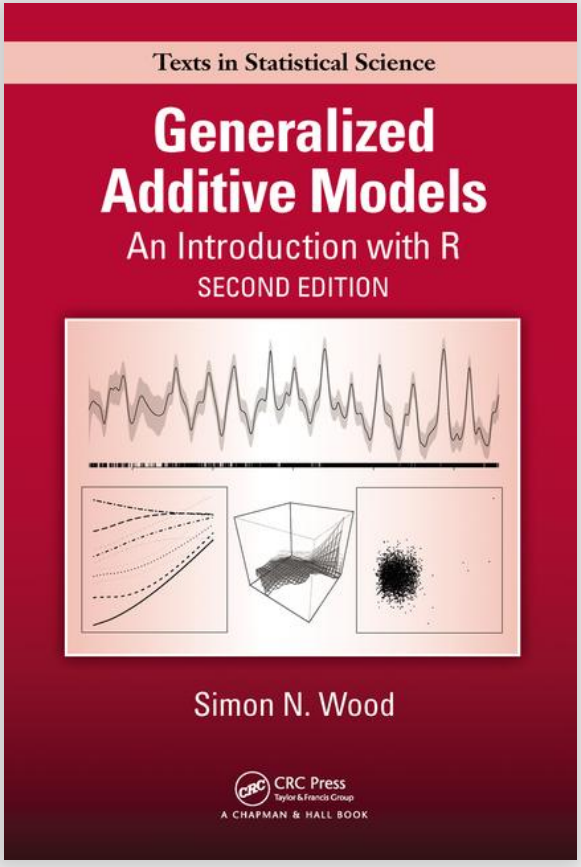
$$B_{i,k}(x) := \frac{x - t_i}{t_{i+k} - t_i} B_{i,k-1}(x) + \frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}} B_{i+1,k-1}(x).$$[7]

(de Boore (1978))

Splines - this was just a taste

- It looks like there are many params to tune here, however in practice `bs()` defaults work quite well with few knots (1-3) to avoid overfitting
- By default `bs()` will fit a cubic spline (`degree = 3`)
- It is custom to control the `df` parameter, e.g.:
  - `df = 4` would produce `degree = 3` and one knot, the median
  - `df = 6` would produce `degree = 3` and 3 knots, Q1, Q2 and Q3
- There are other splines: Natural splines (see `splines::ns()`), Penalized splines
- For retrieving the regression formula for a `lm()` object fitted on `bs()` see the SplinesUtils package and this SO answer
- What is it good for, really? See next week.

Few Excellent Books



CCA

(Canonical Correlation Analysis)

Applications

Of Data Science

What is Linear Regression, really?

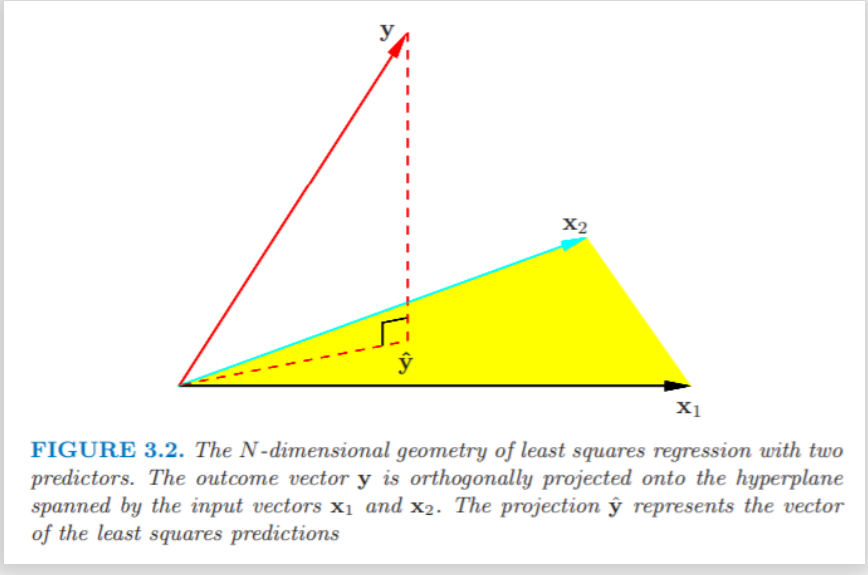


FIGURE 3.2. *The N-dimensional geometry of least squares regression with two predictors. The outcome vector* $\mathbf{y}$ *is orthogonally projected onto the hyperplane spanned by the input vectors* $\mathbf{x}_1$ *and* $\mathbf{x}_2$*. The projection* $\hat{\mathbf{y}}$ *represents the vector of the least squares predictions*

(ESL II)

$$y = X\beta + \epsilon \rightarrow \hat{y} = X\hat{\beta} = X(X'X)^{-1}X'y = Hy$$

"We project y to the space spanned by X..."

"We seek a linear combination of X that would be closest to y..."

What if y is Y?

If Y is multi-dimensional (not a single vector, but a matrix of nxq dependent variables), we get the same result with a $\hat{B}_{pxq}$ matrix:

$$Y = XB + E \rightarrow \hat{Y} = X\hat{B} = X(X'X)^{-1}X'Y = HY$$

And if the columns of X are linearly independent this is equivalent to q separate linear regressions:

```
n <- 1000
x1 <- rnorm(n)
x2 <- rnorm(n)
x3 <- rnorm(n)

y1 <- 0 + 1 * x1 + 2 * x2 + 3 * x3 + rnorm(n)
y2 <- 1 + 2 * x1 + 3 * x2 + 4 * x3 + rnorm(n)
Y <- cbind(y1, y2)

lm(Y ~ x1 + x2 + x3)
```

```
##
## Call:
## lm(formula = Y ~ x1 + x2 + x3)
##
## Coefficients:
##              y1         y2
## (Intercept)  -0.05197   1.00172
## x1            1.02041   2.02414
## x2            2.07436   3.01918
## x3            2.98838   3.96748
```

Though harder to interpret geometrically, it isn't that different:

"We project each of Y's columns to the space spanned by X..."

"We seek a few linear combinations of X that would be closest to each of Y's columns..."

$\boxed{\rightarrow}$ But in case of multi-dimensional Y could we not find linear combinations **of** Y that would be closest to linear combinations of X ?

What is Pearson correlation coefficient, really?

$$r = Corr(x, y) = \frac{Cov(x, y)}{\sqrt{Var(x)}\sqrt{Var(y)}} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2}\sqrt{\sum(y_i - \bar{y})^2}}$$

If we assume x and y are standardized (centered and scaled) to have mean 0 and variance $\frac{1}{n-1}$ (they have a L2 norm of 1), we could write:

$$r = \sum x_i y_i \ s.t. \ \sum x_i = 0; \sum y_i = 0; \sum x_i^2 = 1; \sum y_i^2 = 1$$

Or:

$$r = x'y \ s.t. \ \sum x_i = 0; \sum y_i = 0; x'x = 1; y'y = 1$$

```r
center_vec <- function(x) x - mean(x)
norm_vec <- function(x) sqrt(sum(x^2))
x_scaled <- center_vec(x1) / norm_vec(center_vec(x1))
y_scaled <- center_vec(y1) / norm_vec(center_vec(y1))

glue::glue("pearson: {format(cor(x1, y1), digits = 3)}
x'y: {format(t(x_scaled) %*% y_scaled, digits = 3)}")
```

```
## pearson: 0.246
## x'y: 0.246
```

So the Pearson's r can be viewed as the dot product between x and y , when scaled.

And the dot product between two vectors with L2 norm of 1 is actually the cosine of the angle between them. The higher the correlation, the larger the cosine, the smaller the angle between them, which gives another interpretation for correlation.

What does correlation have to do with regression?

$$y = a + bx + \epsilon \rightarrow y = X\beta + \epsilon$$

And we're intereseted in $\hat{b}$ which is $\hat{\beta}_2$.

$$\hat{\beta} = (X'X)^{-1}X'y =$$

$$(\begin{bmatrix} 1 & \dots & 1 \\ x_1 & \dots & x_n \end{bmatrix} \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix})^{-1} \begin{bmatrix} 1 & \dots & 1 \\ x_1 & \dots & x_n \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

It's not that hard to get that:

$$\hat{b} = \hat{\beta}_2 = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

But that means...

$$\hat{b} = \hat{\beta}_2 = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})\sqrt{\sum(y_i - \bar{y})^2}}{\sum(x_i - \bar{x})^2\sqrt{\sum(y_i - \bar{y})^2}}$$

$$= \frac{\sum(x_i - \bar{x})(y_i - \bar{y})\sqrt{\sum(y_i - \bar{y})^2}}{\sqrt{\sum(x_i - \bar{x})^2}\sqrt{\sum(y_i - \bar{y})^2}\sqrt{\sum(x_i - \bar{x})^2}}$$

$$= \frac{Cov(x, y)}{\sqrt{Var(x)}\sqrt{Var(y)}}\frac{s_y}{s_x} = r\frac{s_y}{s_x}$$

So the Pearson's r is the regression coefficient when regressing y against x, and vice versa, times a different factor! (And the same if they are standardized to have the same variance)

```
lm_yx <- lm(y1 ~ x1); lm_xy <- lm(x1 ~ y1); s_x = sd(x1); s_y = sd

glue::glue("pearson: {format(cor(x1, y1), digits = 3)}
b_y * s_x / s_y: {format(lm_yx$coef[2] * s_x/s_y, digits = 3)}; b_
```

```
## pearson: 0.246
## b_y * s_x / s_y: 0.246; b_x * s_y / s_x: 0.246
```

And now, CCA

- Let X_{nxp} be a matrix of n observations with p random variables and Y_{nxq} matrix of the same n observations with q different random variables.
- For example, X represents n people answering p questions in questionnaire A measuring stress, and Y represents the same n people answering q in questionnaire B also measuring stress.
- Canonical Correlation Analysis seeks to find linear combinations of Y and of X that would be "closest" in the common space.
- Whether you understand "closest" via correlation (small angle) or via "regression" (small sum of squares) it is the same thing.
- A typical result would be $Corr(x_1, \frac{y_1+y_2+y_3}{3}) = 0.8$, meaning question 1 of questionnaire A is "very" correlated with the average of questions 1, 2 and 3 of questionnaire B.

We want a "viewpoint" $u \in R^p$ to look at high-dimensional X, and a "viewpoint" $v \in R^q$ to look at high-dimensional Y, such that:

$$(u, v) = \text{argmax}_{u,v} Corr(Xu, Yv) = \text{argmax}_{u,v} \frac{Cov(Xu, Yv)}{\sqrt{Var(xu)}\sqrt{Var(Yv)}}$$

Writing $Corr(Xu, Yv)$ as we did:

$$(0)\ (u, v) = \text{argmax}_{u,v} Corr(Xu, Yv) = \text{argmax}_{u,v} u'X'Yv$$

$$\text{s.t. } \sum_i X_{ij} = 0\ \forall j; \sum_i Y_{il} = 0\ \forall l; u'X'Xu = 1; v'Y'Yv = 1$$

Applying the Lagrange multipliers technique to this optimization gives:

$$L = u'X'Yv - \frac{\theta}{2}(u'X'Xu - 1) - \frac{\tau}{2}(v'Y'Yv - 1) \rightarrow max_{u,v}$$

(where the $\sum_i X_{ij} = 0$ terms are dropped since they won't have an effect on u and v)

Taking derivatives with respect to u and v and comparing to 0 we obtain the equations:

$$(1)\ X'Yv - \theta X'Xu = 0\ \ (2)\ Y'Xu - \tau Y'Yv = 0$$

Multiplying equation (1) by u, and equation (2) by v, and subtracting equation (2) from equation (1) we get: $\theta u'X'Xu = \tau v'Y'Yv$

However, $u'X'Xu = v'Y'Yv = 1$, so we get:

$$\theta = \tau = u'X'Yv = Corr(Xu, Yv) = r$$

Equations (1) and (2) now become:

$$(3)\ X'Yv - rX'Xu = 0\ \ (4)\ Y'Xu - rY'Yv = 0$$

If we assume $Y'Y$ and $X'X$ are invertible we can write:

$$(5)\ v = \frac{1}{r}(Y'Y)^{-1}Y'Xu\ \ (6)\ u = \frac{1}{r}(X'X)^{-1}X'Yv$$

Substituting for v in equation (3):

$$(X'X)^{-1}X'Y(Y'Y)^{-1}Y'Xu = r^2u$$

If we write $A = (X'X)^{-1}X'Y(Y'Y)^{-1}Y'X$ we see that:

$$Au = r^2u$$

is a simple eigenproblem.

Here the first k canonical correlations $r_1, \ldots, r_k$ are the square roots of the first ordered k eigenvalues of the matrix $A = (X'X)^{-1}X'Y(Y'Y)^{-1}Y'X$ and the first k weight vectors $u_1, \ldots, u_k$ are the associated eigenvectors, where $k = \min(p, q)$.

Similarly, $v_1, \ldots, v_k$ are the eigenvectors of the matrix $B = (Y'Y)^{-1}Y'X(X'X)^{-1}X'Y$, or we can put in (5) any u_i and r_i, to give v_i.

The beauty of CCA

CCA can be viewed as a unified framework for dimensionality reduction in multivariate data analysis, and it generalizes other methods:

- If we let $X = Y$ and change the constraints to be $u'u = 1$ we get Principal Components Analysis (PCA), i.e. finding directions of maximum variance in X (or let X predict itself)
- If we let $Y'Y = I$ we get a result equivalent to multivariate linear regression
- If $X'X = I$ and $Y'Y = I$ we get symmetrical Partial Least Squares (PLS), i.e. u and v would maximize the sample covariance $u'X'Yv$

CCA: A Simple Example

We have 20 middle-aged men, some data regarding their physique and some data regarding their fitness results:

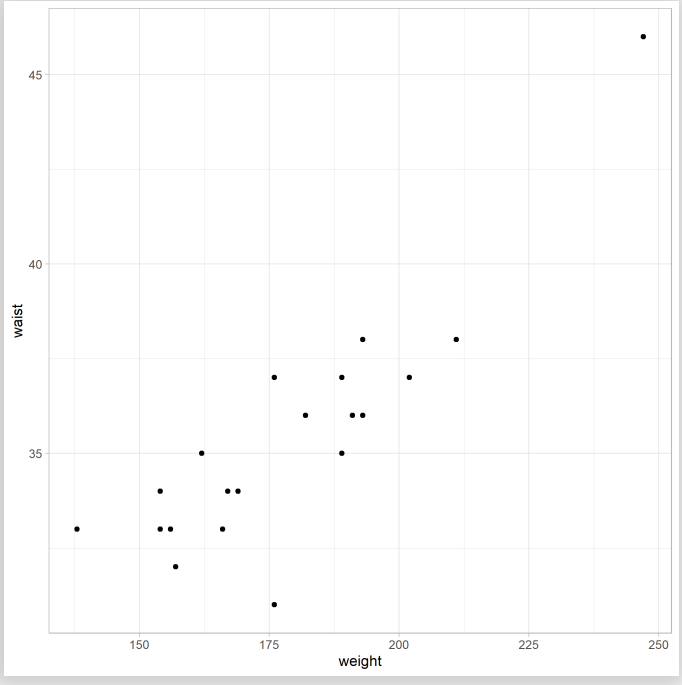
```
fitness <- read_csv("../data/fitness.csv")

head(fitness)
```

```
## # A tibble: 6 x 6
##   weight waist pulse chins situps jumps
##    <dbl> <dbl> <dbl> <dbl>  <dbl> <dbl>
## 1    191    36    50     5    162    60
## 2    189    37    52     2    110    60
## 3    193    38    58    12    101   101
## 4    162    35    62    12    105    37
## 5    189    35    46    13    155    58
## 6    182    36    56     4    101    42
```

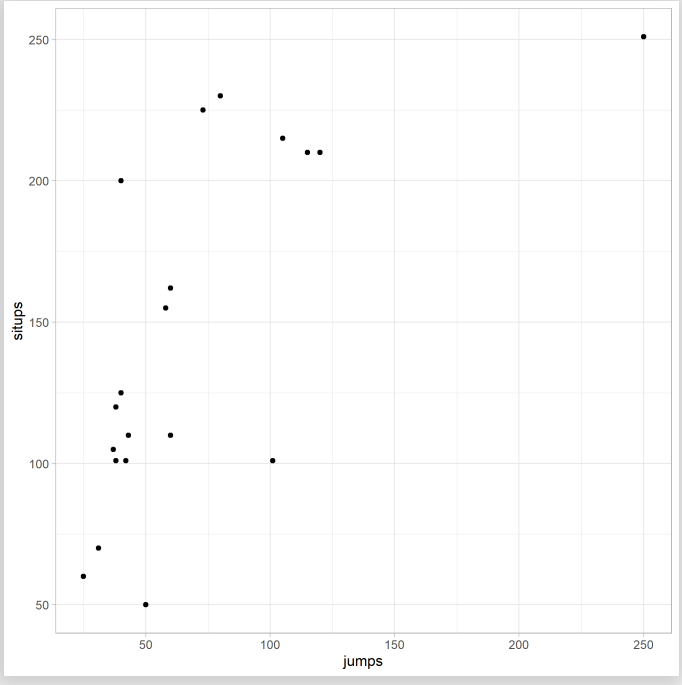
What is the relationship between a man's physique and his fitness results?

We have a *really* heavy guy, and a guy who *really* likes to jump:

```
ggplot(fitness, aes(weight, wa
  geom_point() +
  theme_light()
```




```
ggplot(fitness, aes(jumps, situ
  geom_point() +
  theme_light()
```



```
fitness <- fitness %>% slice(-14, -20)
X <-  fitness %>%
  select(weight, waist, pulse) %>% as.matrix()
Y <- fitness %>%
  select(chins, situps, jumps) %>% as.matrix()

mod_cca <- cancor(X, Y)

mod_cca
```

```
## $cor
## [1] 0.81405809 0.29671748 0.06902653
##
## $xcoef
##                [,1]         [,2]         [,3]
## weight -0.004532796 -0.021071104 -0.000489462
## waist   0.138548292  0.115809822 -0.045769011
## pulse  -0.004648447  0.003889603 -0.036050107
##
## $ycoef
##                [,1]         [,2]         [,3]
## chins  -0.017960992  0.019265549 -0.060928131
## situps -0.004276666 -0.002437772  0.003750079
## jumps   0.003124631  0.005168287  0.001372880
##
## $xcenter
##    weight     waist     pulse
## 177.05556  34.94444  55.77778
##
```

What does this mean?

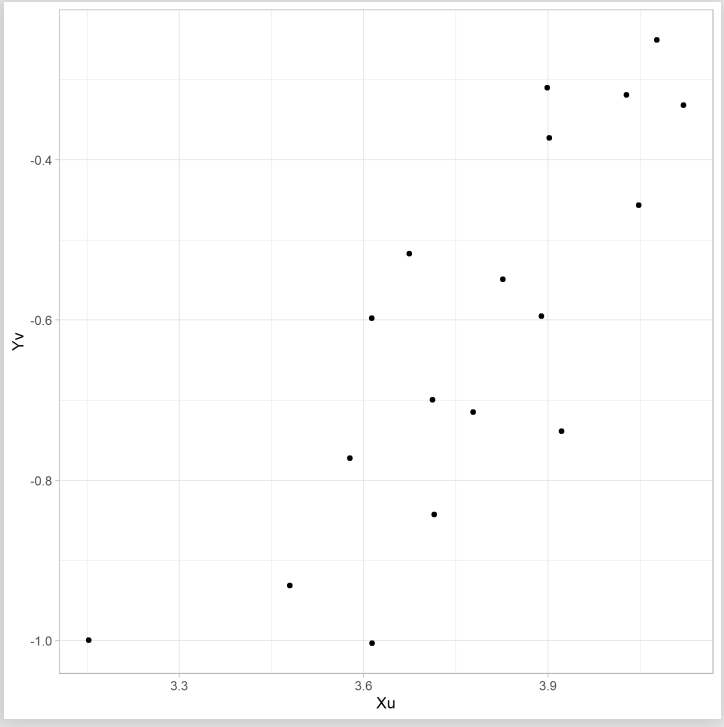
- The first canonical pair shows a strong negative relationship between waist size and no. of chins (the larger the waist, the less chins)
- The effect waist size has on chins is mildly suppressed by weight and pulse
- But this DOES NOT mean weight is positively correlated with chins, only that it is LESS negatively correlated (check this)
- Anyway 20 observations... don't cite this.

Also make sure you get the same correlation on the projected X and projected Y:

```
Xu = X %*% mod_cca$xcoef[, 1]
Yv = Y %*% mod_cca$ycoef[, 1]
cor(Xu, Yv)[1, 1]
```

```
## [1] 0.8140581
```

```
tibble(Xu, Yv) %>%
  ggplot(aes(Xu, Yv)) +
  geom_point() +
  theme_light()
```



CCA: A Problem

What if p and/or q are very large?

Example: a researcher in order to understand a complex disease, would want to find relations between the expressions of many genes (like, tens of thousands) and the variations in many DNA-markers for this disease

- all multiplications, inversions and eigendecompositions of matrices (e.g. $(Y'Y)^{-1}$) are long and unstable
- as these Regression-like solutions tend to be, u and v will not be sparse, will probably have all elements different from zero, making interpretation very difficult

Sparse CCA

Witten, Tibshirani and Hastie (2009) developed one possible approach, Diagnoal Penalized CCA (or DP-CCA)

General approach for decomposing matrices with penalties, "Penalized Matrix Decomposition" (PMD). They show how Sparse CCA can be achieved by essentially performing what they call $PMD(L1, L1)$ on the covariance matrix $X'Y$, with some modifications to (0):

- Treat $X'X$ and $Y'Y$ as the identity matrix I, a step that has proven to be beneficial in high-dimensional problems where we expect sparsity
- Loosen up the variance constraint, the L2 norm constraint, to be ≤ 1 rather than $= 1$, to make the problem convex
- Add the Lasso penalty, controlling the L1 norm of u and v, which tends to bring sparse solutions

(0) then becomes:

$$(u, v) = \text{argmax}_{u,v} Corr(Xu, Yv) \approx \text{argmax}_{u,v} u'X'Yv$$

$$\text{s.t. } u'u \leq 1; v'v \leq 1; ||u||_1 \leq c_1; ||v||_1 \leq c_2$$

With c_1 and c_2 being penalty parameters which can be chosen via cross-validation.

And from here the problem is solved iteratively:

Assuming some v vector, the problem becomes very similar to the LASSO problem:

$$u = \text{argmax}_u u' X' Y v \text{ s.t. } u'u \leq 1; ||u||_1 \leq c_1$$

We are "regressing" X on Yv (which is now a given vector!) to get an update for vector u, with this u "regressing" Y on Xu to get an update for vector v, and so on until convergence.

Using Regression also means high speed!

These are u_1 and v_1. To get the next pair of u_2 and v_2, they repeat the process on the "remainder" of the $X'Y$ covariance matrix, once you subtract the resulting $u_1' X' Y v_1 \cdot u_1 \cdot v_1'$ matrix from it. And so on:

$$X'Y^{k+1} \leftarrow X'Y^k - u_1' X' Y v_1 \cdot u_1 \cdot v_1'$$

CCA: A Modern Example

Using the `chorrrds` package I scraped the Brazilian site CifraClub for the chords and lyrics of thousands of Rock songs:

```
chords_lyrics <-  read_rds(url(chords_url, "rb"))
X <- chords_lyrics$X
Y <- chords_lyrics$Y

dim(X)
```

```
## [1] 3297   79
```

```
dim(Y)
```

```
## [1] 3297  784
```

for the script including some basic pre-processing see here.

Are there interesting relations between chords and lyrics of Rock songs?

What's in X?

```
X[1:5, 1:3]
```

```
##                                  Terms
## Docs                                    A5        D5         E5
##   ac dc back in black            1.38518557 1.4010408 1.47338388
##   ac dc big balls                0.00000000 0.0000000 0.00000000
##   ac dc cant stop rock n roll    1.36621043 0.6909242 0.00000000
##   ac dc for those about to rock 0.03710318 0.0000000 0.03946564
##   ac dc highway to hell          0.00000000 0.0000000 0.14258554
```

What's in Y?

```
Y[1:5, 1:3]
```

```
##                                  Terms
## Docs                                 black     catch        die
##   ac dc back in black            0.9662521 0.1348891 0.09439266
##   ac dc big balls                0.0000000 0.0000000 0.00000000
##   ac dc cant stop rock n roll    0.0000000 0.0000000 0.00000000
##   ac dc for those about to rock 0.0000000 0.0000000 0.00000000
##   ac dc highway to hell          0.0000000 0.0000000 0.00000000
```

DP-CCA in R

```
library(PMA)

K <- 10
scca <- CCA(X, Y, K = 10, niter = 100, trace = FALSE)

print(scca)
```

```
## Call: CCA(x = X, z = Y, K = 10, niter = 100, trace = FALSE)
##
##
## Num non-zeros u's:  9 9 14 22 16 18 37 30 32 27
## Num non-zeros v's:  206 144 221 262 177 279 277 199 268 212
## Type of x:  standard
## Type of z:  standard
## Penalty for x: L1 bound is  0.3
## Penalty for z: L1 bound is  0.3
## Cor(Xu,Zv):  0.4481368 0.3475206 0.4145097 0.4594632 0.4311564 0.4451745
## 0.489829 0.4203822 0.463351 0.4276246
```

The first pair of u and v has 9 non-zero chords (out of 79!) correlated with 206 non-zero words (out of 784!), and a correlation of 0.448.

The top 5 chords in terms of absolute weights and corresponding weights are:

```
k <- 1
important_id <- which(scca$u[, k] != 0)
important_id_order <- order(abs(scca$u[important_id, k]), decreasi
important_id <- important_id[important_id_order[1:5]]
important_id <- important_id[!is.na(important_id)]
colnames(X)[important_id]
```

```
## [1] "B5"  "A5"  "E5"  "D5"  "F#5"
```

```
scca$u[important_id, k]
```

```
## [1] 0.5120943 0.4797985 0.4200780 0.3749777 0.2375822
```

The top 5 words and corresponding weights are:

```
important_id <- which(scca$v[, k] != 0)
important_id_order <- order(abs(scca$v[important_id, k]), decreasi
important_id <- important_id[important_id_order[1:5]]
important_id <- important_id[!is.na(important_id)]
colnames(Y)[important_id]
```

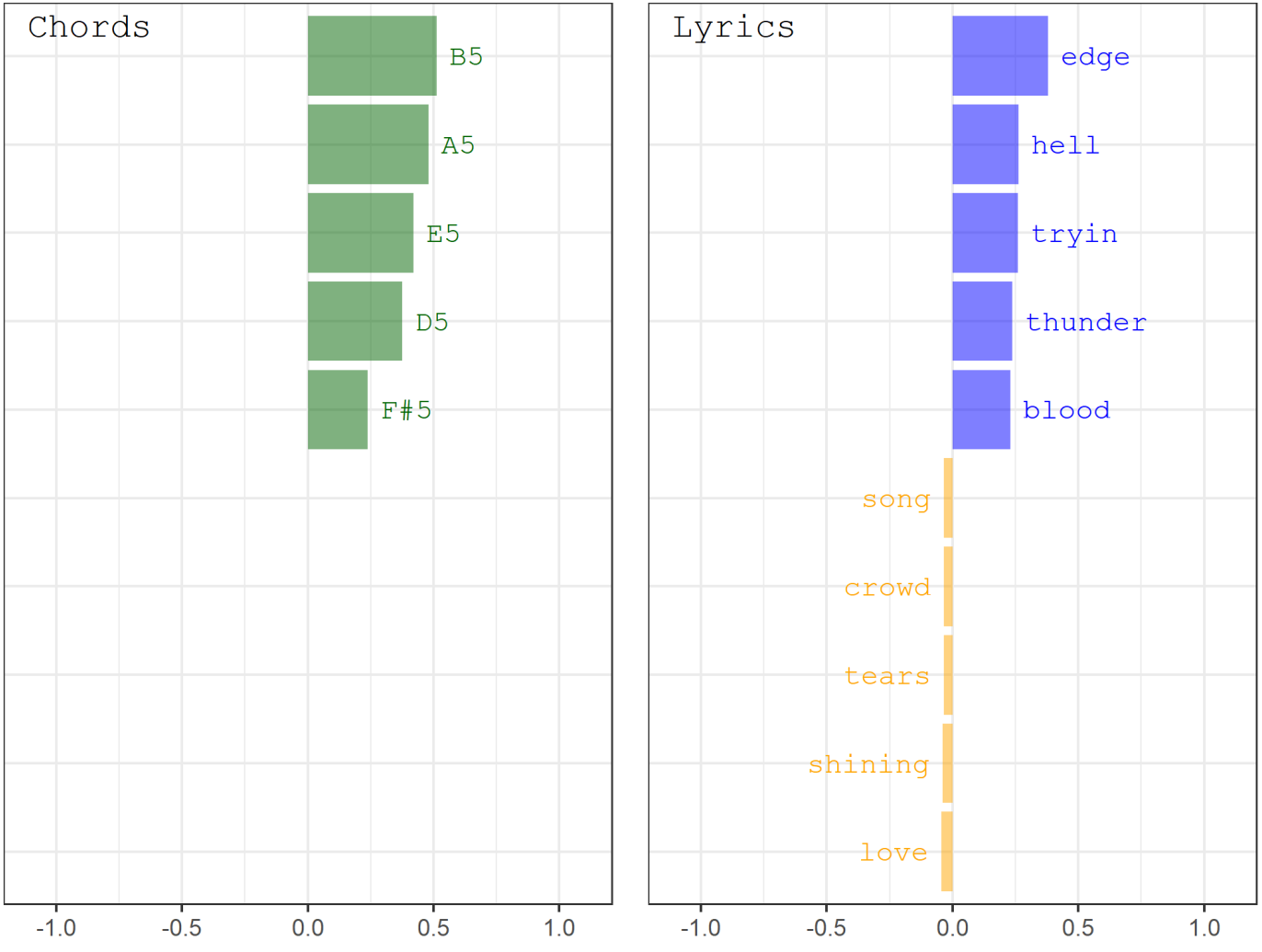
```
## [1] "edge"    "hell"    "tryin"   "thunder" "blood"
```

```
scca$v[important_id, k]
```

```
## [1] 0.3788166 0.2625134 0.2603274 0.2382183 0.2306617
```

If you know some chords this pattern makes perfect sense! The A5, B5, D5 etc. chords are also known as Power Chords, usually played with electric guitars, and are used mainly in hard rock songs (think Guns n Roses), where words such as "hell", "thunder" and... "blood" are used abundantly.

Power chords are correlated with hard rock lyrics



Major chords A, E, B are correlated with Soft Rock happy lyrics