APPLICATIONS

OF DATA SCIENCE

Intro. to Multivariate Methods

Applications of Data Science - Class 14

Giora Simchoni

gsimchoni@gmail.com and add #dsapps in subject

Stat. and OR Department, TAU

2022-12-26

PCA

(Principle Components Analysis)

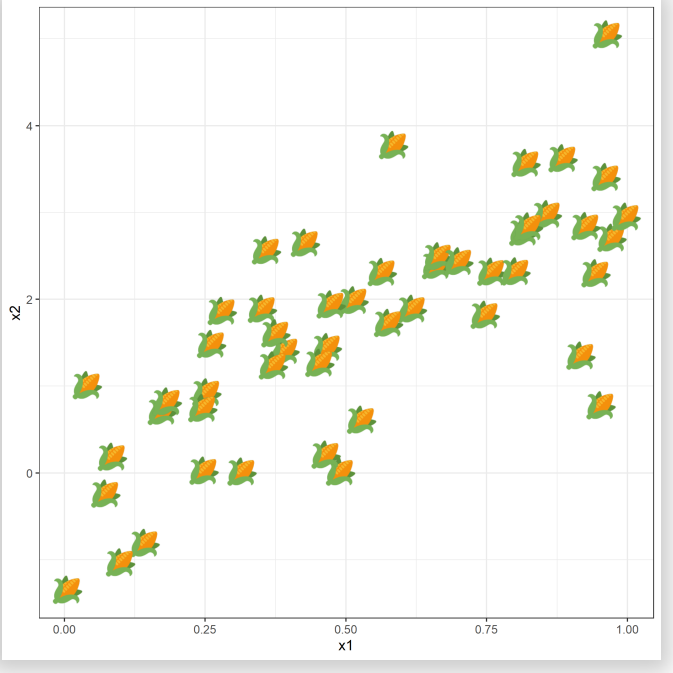
Dimensionality reduction - why?

- EDA:
  - High dimensionality data $X_{n \times p}$
  - Assume some features are correlated
  - Hard to see latent/hidden structure in high-dimensional space
  - Transform the data matrix into a set of linearly uncorrelated variables called Principal Components
  - The first few PCs usually account for most of the variability in the data
  - So by visualizing only these PCs, the basic internal structure of the data can be revealed

- Performance in ML:
  - (Especially for old-school models, old-school computation)
  - p is so large we cannot apply our ML model of choice
  - Features decorrelation: Models which suffer from multicollinearity (which?)
  - Clustering (e.g. K-Means, community detection)
- Feature Engineering!
- Compression (images)

A Metaphor.

You're in a corn field. You have no drone. Where do you stand to watch most corns?



Finding the First PC

We wish to find a vector t_1, which is a linear transformation of X:

$$t_1 = Xw_1$$

Such that it will have maximum variance
$V(t_1) = E[t_1 - E(t_1)]^2 = E[Xw_1 - E(Xw_1)]^2$

We can assume $E(X) = 0$ and if not we will subtract the mean off every column of X ("centering"), therefore $E(Xw_1) = 0$.

We are maximizing $E(Xw_1)^2$, which in the sample sense becomes:

$$V(t_1) = \frac{1}{n} w_1' X' X w_1$$

Or really what we're after: $\max_{w_1} w_1' X' X w_1 \text{ s.t.} ||w_1|| = 1$

> 💡 Why s.t. $||w_1|| = 1$?

Finding the First PC - Cont.

Using Lagrange multiplier λ_1: $\max_{w_1} w_1' X' X w_1 + \lambda_1 (1 - w_1' w_1)$

Take derivative with respect to w_1, compare to 0:

$$2X'Xw_1 - 2\lambda_1 w_1 = 0$$
$$X'Xw_1 = \lambda_1 w_1$$

So w_1 must be eigenvector of the square $X'X$ matrix, and λ_1 its eigenvalue!

> 💡 Which eigenvalue and eigenvector?

> 💡 Many confuse w_1 and t_1.
>
> w_1 is a weight vector, "loadings", "rotation". t_1 is the PC.

Finding the next orthogonal PCs

Turns out we're looking for the set of $W_{p\times p}$ eigenvectors of $X'X$ with their corresponding eigenvalues $\lambda_1, \ldots, \lambda_p$ ordered from largest to smallest.

One can also show the λ_i themselves are the variances of the PCs!

> 💡 Do it!

Once we get W, we project X to get the PCs: $T = XW$

But we can only compute the first few K loadings accounting for most variability in the data, $W^*_{p\times K}$.

Then $T^* = XW^*$ of dimensions $n \times K$ where $K << p$ is X with *reduced* dimensionality.

PCA in R

```
X <- matrix(rnorm(5 *  2), nrow = 5)
pca <- prcomp(X) # see default scaling
```

The loadings, rotation W:

```
pca$rotation
```

```
##              PC1       PC2
## [1,] -0.9300713 0.3673791
## [2,]  0.3673791 0.9300713
```

The PCs:

```
pca$x
```

```
##              PC1        PC2
## [1,] -0.58032309  0.3166279
## [2,] -1.05943800 -0.1812795
## [3,]  0.84344476 -0.1061266
## [4,]  0.73126676  0.1247415
## [5,]  0.06504958 -0.1539632
```

The variances:

```
pca$sdev ** 2 #compare with apply(pca$x, 2, var)
```

```
## [1] 0.67739134 0.04591085
```

Check R:

```
X_scaled <- scale(X, center = T, scale = F)

X_scaled %*% pca$rotation
```

```
##               PC1         PC2
## [1,] -0.58032309  0.3166279
## [2,] -1.05943800 -0.1812795
## [3,]  0.84344476 -0.1061266
## [4,]  0.73126676  0.1247415
## [5,]  0.06504958 -0.1539632
```

PCA DIY (don't really...)

```
eigen(t(X_scaled) %*% X_scaled)$vectors
```

```
##             [,1]       [,2]
## [1,] -0.9300713 -0.3673791
## [2,]  0.3673791 -0.9300713
```

```
eigen(t(X_scaled) %*% X_scaled)$values / (5 - 1)
```

```
## [1] 0.67739134 0.04591085
```

> 💡 Though that's not what `prcomp()` does. `prcomp()` uses SVD.
>
> But we're good for now.

PCA on Netflix data

```
con <- url("http://www.tau.ac.il/~saharon/StatsLearn2022/train_rat
netflix <- as_tibble(
  read.table(con, colClasses = c(rep("integer", 14), rep("NULL", 9
)

con <- url("http://www.tau.ac.il/~saharon/StatsLearn2022/movie_tit
titles <- read.table(con, sep = ",", nrows = 14)
colnames(netflix) <- janitor::make_clean_names(str_sub(titles[,2],

netflix
```

```
## # A tibble: 10,000 × 14
##    independence_da the_patriot the_day_after_t pirates_of_the pretty_wom
##              <int>       <int>           <int>          <int>       <in
##  1               2           4               4              4
##  2               4           5               3              3
##  3               5           5               5              5
##  4               3           3               3              4
##  5               4           4               4              5
##  6               4           3               3              4
##  7               5           4               5              5
##  8               3           5               3              4
##  9               5           5               5              5
## 10               4           4               4              4
## #   with 9,990 more rows, and 9 more variables: forrest_gump <int>
```

What do the first PCs "mean"?

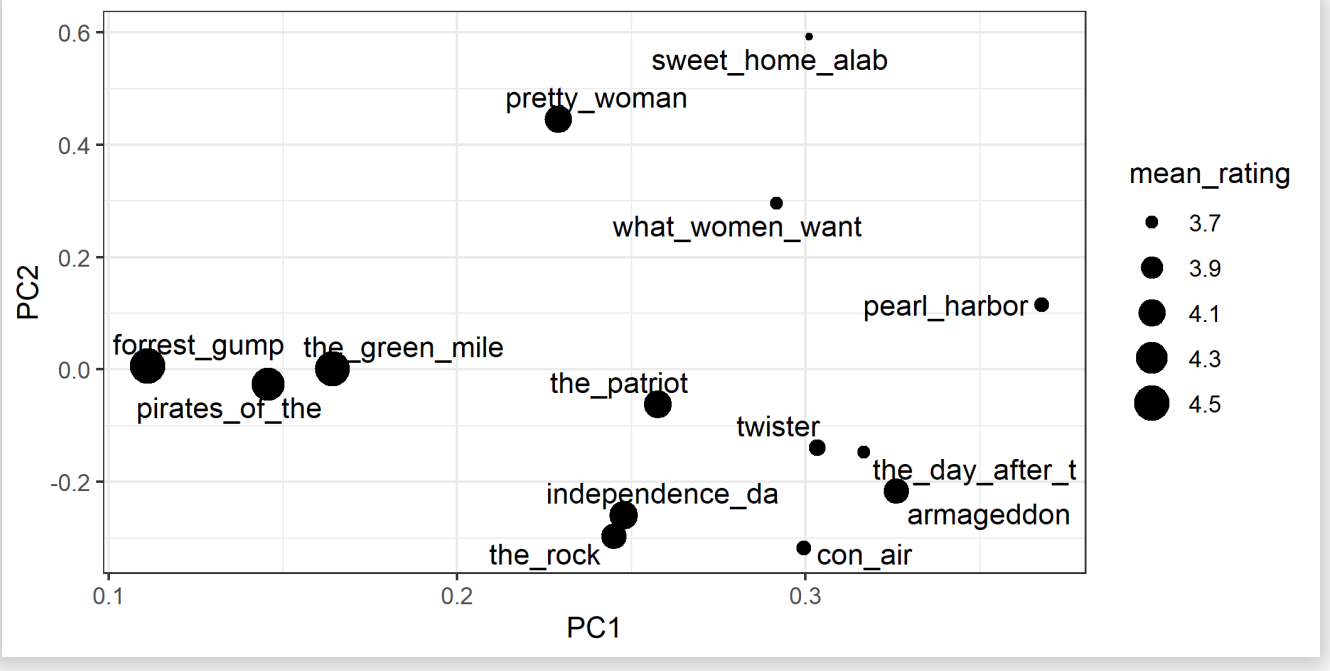
Look at first loadings vs. mean popularity of movie:

```
pca <- prcomp(netflix)

movies <- enframe(colMeans(netflix))
colnames(movies) <- c("title", "mean_rating")
W2 <- bind_cols(movies, as_tibble(pca$rotation[,1:2]))

W2
```

```
## # A tibble: 14 × 4
##    title           mean_rating   PC1      PC2
##    <chr>                 <dbl> <dbl>    <dbl>
##  1 independence_da        4.15 0.248 -0.259
##  2 the_patriot            4.11 0.258 -0.0618
##  3 the_day_after_t        3.70 0.317 -0.147
##  4 pirates_of_the         4.35 0.146 -0.0266
##  5 pretty_woman           4.07 0.229  0.447
##  6 forrest_gump           4.51 0.111  0.00623
##  7 the_green_mile         4.46 0.164  0.00184
##  8 con_air                3.73 0.299 -0.317
##  9 twister                3.75 0.303 -0.139
## 10 sweet_home_alab        3.67 0.301  0.593
## 11 pearl_harbor           3.73 0.368  0.116
## 12 armageddon             4.02 0.326 -0.216
## 13 the_rock               4.02 0.245 -0.297
```

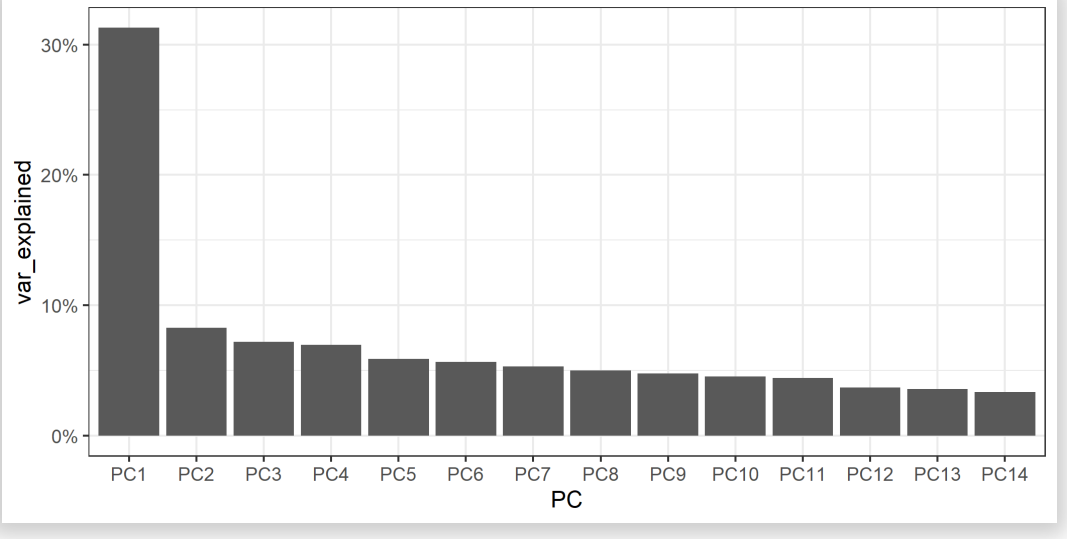
Or visualize:

```
W2 %>%
  ggplot(aes(PC1, PC2)) +
  geom_point(aes(size = mean_rating)) +
  ggrepel::geom_text_repel(aes(label = title)) +
  theme_bw()
```



How much variance is "explained" by the PCs? (Scree plot)

```
ve <- tibble(PC = str_c("PC", 1:14),
             var_explained = summary(pca)$importance[2, ])

ve %>%
  ggplot(aes(PC, var_explained)) +
  geom_bar(stat = "identity") +
  scale_x_discrete(limits=ve$PC) +
  scale_y_continuous(labels = scales::percent_format()) +
  theme_bw()
```

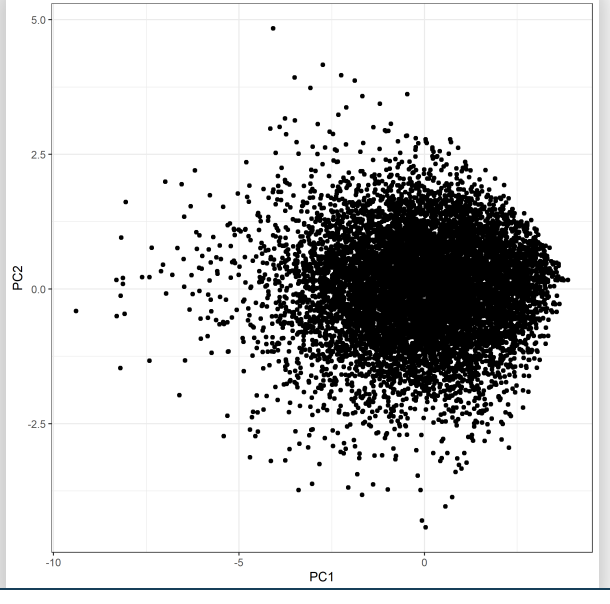


Now those PCs

Unfortunately we know nothing else of our users to add to this map.

You can develop your own theories about viewers who are low/high on PC1/PC2.

```
as_tibble(pca$x[, 1:2]) %>%
  ggplot(aes(PC1, PC2)) + geom_point() + theme_bw()
```



How did Feature Engineering slip in?

Suppose your job is to predict the viewers rating of [Miss Congeniality]().

How would viewers low/high on PC2 rate it?

```
con <- url("http://www.tau.ac.il/~saharon/StatsLearn2022/train_y_r
mc <- as_tibble(read.table(con))
colnames(mc) <- "mc_score"

PC2_top100 <- order(pca$x[, 2], decreasing = T)[1:100]
PC2_bottom100 <- order(pca$x[, 2])[1:100]
```

```
mc[PC2_top100,] %>% count(mc_score)
```

```
## # A tibble: 4 × 2
##   mc_score     n
##      <int> <int>
## 1        2    12
## 2        3    26
## 3        4    28
## 4        5    34
```

```
mc[PC2_bottom100,] %>% count(mc_score)
```

```
## # A tibble: 5 × 2
##   mc_score     n
##      <int> <int>
## 1        1    17
## 2        2    27
## 3        3    35
## 4        4    12
## 5        5     9
```

> 💡 I know, shocker. But what a feature.

FA

(Exploratory Factor Analysis)

APPLICATIONS

OF DATA SCIENCE

Excuse me, where is your model?

In PCA we wrote: $T = XW$ and *calculated* that projection $W_{p \times K}$ where $K << p$.

This is very similar to modeling our observed data $X_{n \times p}$ as a linear combination of K latent *factors* $T_{n \times K}$, where $K << p$, and:

$X = TW'$, where $W_{p \times K}$ is again a loadings matrix.

Think Psychology Department, where a researcher makes $n = 100$ subjects fill out a questionnaire with $p = 50$ questions assessing their self-efficacy. She assumes self-efficacy is the product of $K = 3$ factors (concepts, constructs): previous experience, support from environment and perceived task difficulty. She might use FA to test that.

The FA Model

Still, if two individuals share the same level of factors (e.g. past experience) they cannot be expected to perform/answer/measure the same.

Enter error $\varepsilon_{n \times p}$:

$$X = TW' + \varepsilon$$

Simplify a bit:

- X of dimensions $n \times p$ is n realizations of a p-dimensional RV x_i
  .
- Factors matrix T of dimensions $n \times K$ is n realizations of a K-dimensional RV t_i.
- Error matrix ε of dimensions $n \times p$ is n realizations of a p-dimensional RV ε_i.
- Assume $\varepsilon_i \sim N(0, \Psi_{p \times p})$ where Ψ is assumed diagonal, and $t_i \sim N(0, I_K)$ (meaning...)
- Let μ be the p-dimensional mean vector for all observations
- then write the FA model: $x_i = Wt_i + \mu + \varepsilon_i$

So x_i is a linear combinations of K factors, plus a mean, plus some normal error.

FA does not calculate. It estimates.

Conditionally, $x_i|t_i \sim N(Wt_i + \mu, \Psi)$.

Marginally, $x_i \sim N(\mu, V)$, where $V_{p\times p} = WW' + \Psi$ (communality + uniqueness).

Write the marginal log-likelihood:
$L(\mu, W, \Psi|X) = -\frac{n}{2}[p\ln(2\pi) + \ln|V| + \mathrm{tr}(V^{-1}S)]$,

where S is the covariance matrix of X:
$S = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)(x_i - \mu)'$

And find the MLE for W, Ψ and predict T.

FA in R

```
X <- matrix(rnorm(10 *  5), nrow = 10)
fa <- factanal(X, factors = 2, scores = "regression")

fa
```

```
##
## Call:
## factanal(x = X, factors = 2, scores = "regression")
##
## Uniquenesses:
## [1] 0.683 0.010 0.828 0.005 0.228
##
## Loadings:
##      Factor1 Factor2
## [1,]  0.563
## [2,] -0.405   0.909
## [3,]  0.174   0.377
## [4,] -0.974   0.214
## [5,]  0.856   0.198
##
##                Factor1 Factor2
## SS loadings      2.193   1.053
## Proportion Var   0.439   0.211
## Cumulative Var   0.439   0.649
##
```

What is what?

```
W <- fa$loadings
Psi <- diag(fa$uniquenesses)
T <- fa$scores # or: scale(X) %*% solve(cor(X)) %*% W
```

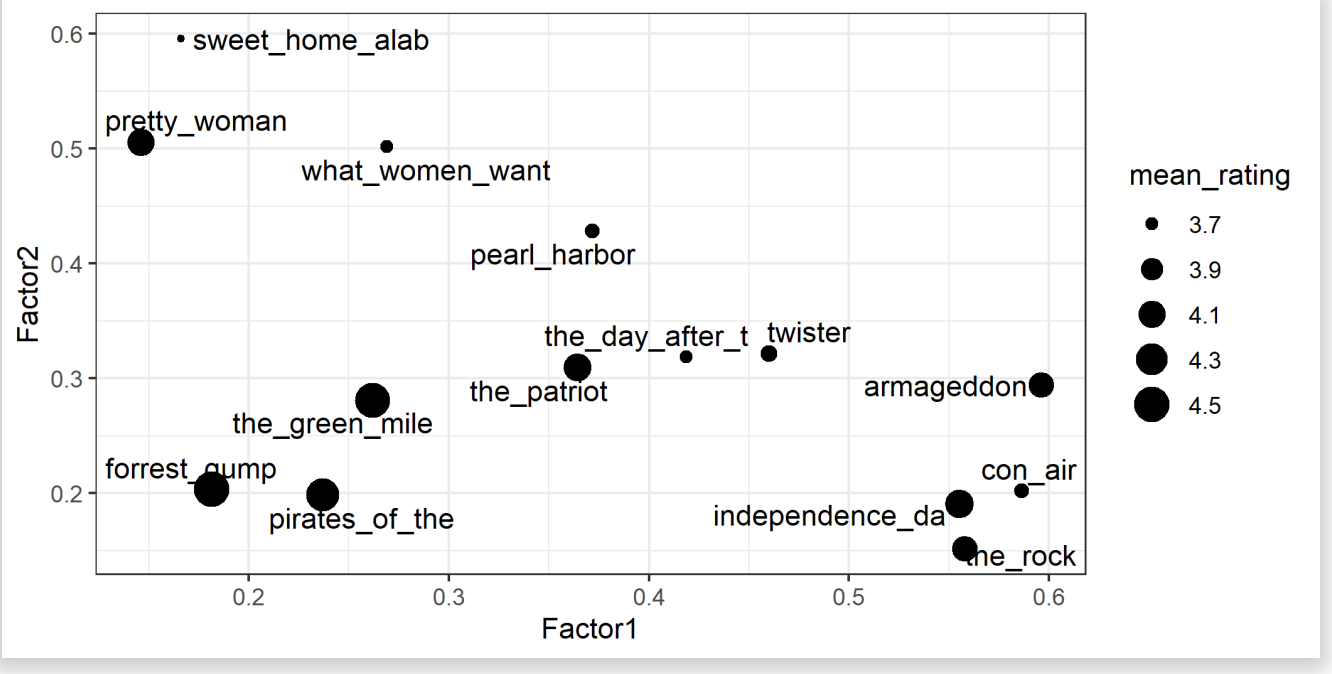
FA on Netflix Data

```
fa <- factanal(netflix, factors = 2, scores = "regression")
W2 <- bind_cols(movies, as_tibble(as.matrix(fa$loadings)[1:14,]))
W2
```

```
## # A tibble: 14 × 4
##    title           mean_rating Factor1 Factor2
##    <chr>                 <dbl>   <dbl>   <dbl>
##  1 independence_da        4.15   0.555   0.191
##  2 the_patriot            4.11   0.365   0.309
##  3 the_day_after_t        3.70   0.419   0.319
##  4 pirates_of_the         4.35   0.237   0.198
##  5 pretty_woman           4.07   0.146   0.506
##  6 forrest_gump           4.51   0.181   0.203
##  7 the_green_mile         4.46   0.262   0.281
##  8 con_air                3.73   0.586   0.202
##  9 twister                3.75   0.460   0.322
## 10 sweet_home_alab        3.67   0.166   0.596
## 11 pearl_harbor           3.73   0.372   0.428
## 12 armageddon             4.02   0.596   0.294
## 13 the_rock               4.02   0.558   0.152
## 14 what_women_want        3.71   0.269   0.501
```

Say isn't that...

```
W2 %>%
  ggplot(aes(Factor1, Factor2)) +
  geom_point(aes(size = mean_rating)) +
  ggrepel::geom_text_repel(aes(label = title)) +
  theme_bw()
```



Probabilistic PCA

Tipping and Bishop show that if $\Psi = \sigma_e^2 I_p$ (i.e. "isotropic" covariance matrix), then:

$$W_{MLE} = U_K(\Lambda_K - \sigma_e^2 I_K)^{1/2} R,$$

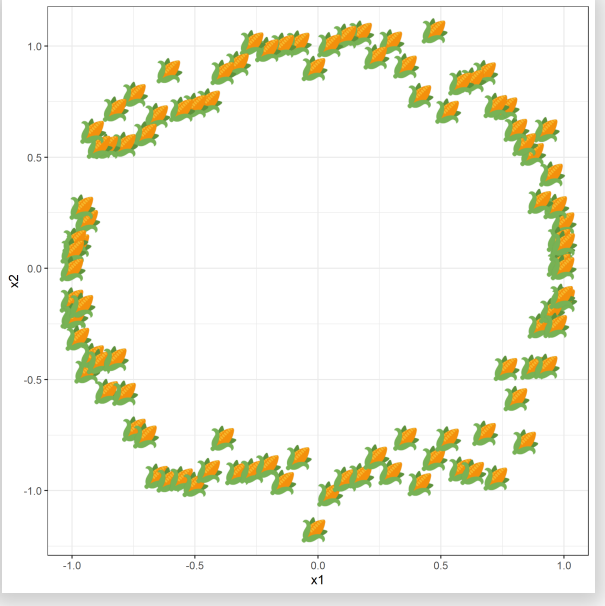
where the columns of U_K are eigenvectors of S, with corresponding K largest eigenvalues in the diagonal matrix Λ_K, and R is a $K \times K$ arbitrary rotation matrix.

These U_K are of course the principal eigenvectors of S, so in the limit $\sigma_e^2 \to 0$ this solution is equivalent to PCA.

We got a probabilistic interpretation to PCA!

Before you go

Question: Where do you "stand" now?



💡 Right answers: Kernel PCA, Auto-Encoders

CCA

(Canonical Correlation Analysis)

APPLICATIONS

OF DATA SCIENCE

What is Linear Regression, really?

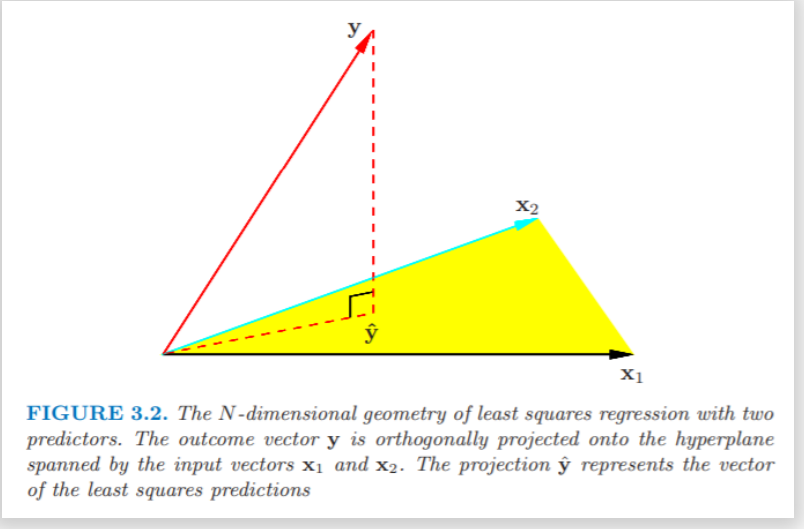


FIGURE 3.2. *The N-dimensional geometry of least squares regression with two predictors. The outcome vector* $\mathbf{y}$ *is orthogonally projected onto the hyperplane spanned by the input vectors* $\mathbf{x}_1$ *and* $\mathbf{x}_2$*. The projection* $\hat{\mathbf{y}}$ *represents the vector of the 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)  -0.03955   0.99723
## x1            0.96308   1.97034
## x2            1.99257   3.02369
## x3            2.98476   4.02250
```

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..."

➡️ 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: 0.268
## x'y: 0.268
```

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: 0.268
## b_y * s_x / s_y: 0.268; b_x * s_y / s_x: 0.268
```

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 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:

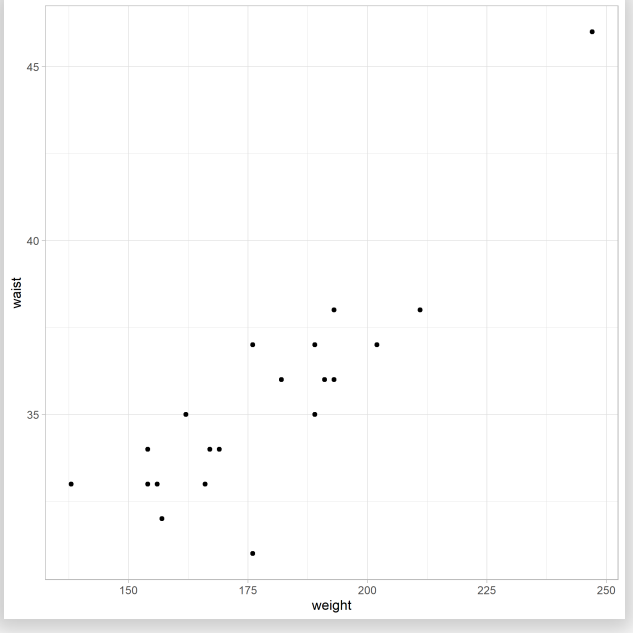
```
fitness <- read_csv("../data/fitness.csv")

head(fitness)
```

```
## # A tibble: 6 × 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
```

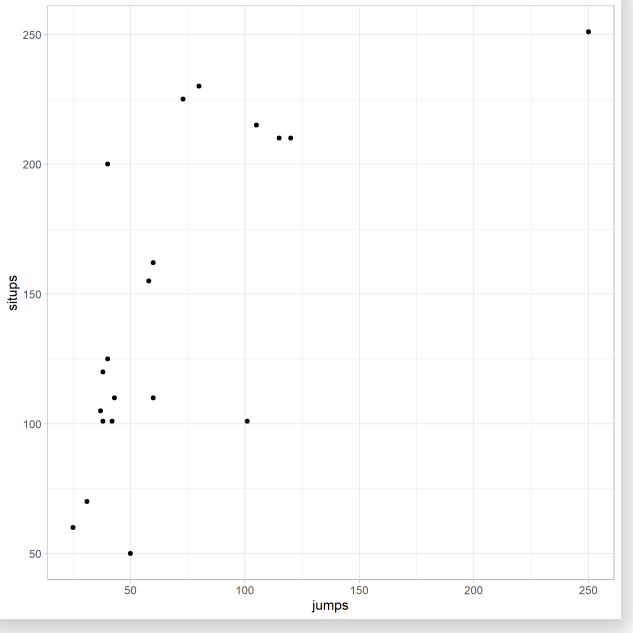
What is the relationship between a man's physique and his fitness results?

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?

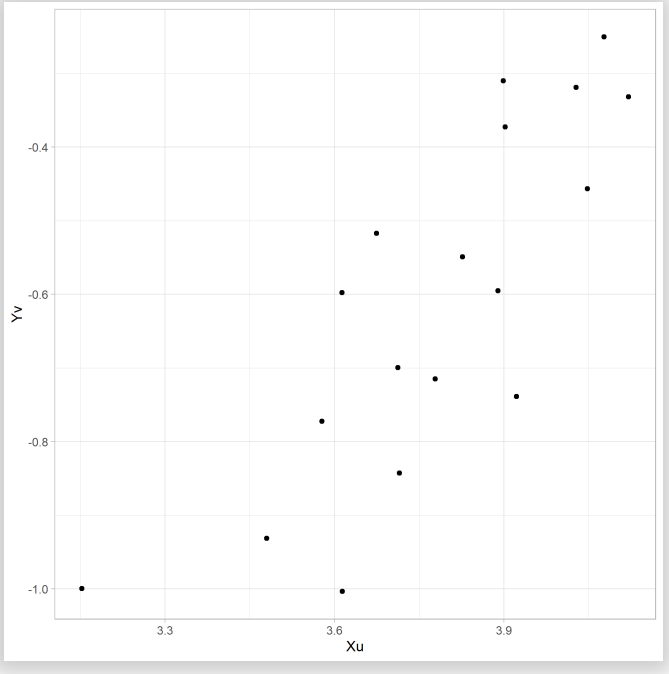
- The first canonical pair shows a strong negative relationship between waist size and no. of chins (the larger the waist, the less chins)
- The effect waist size has on chins is mildly suppressed by weight and pulse
- But this DOES NOT mean weight is positively correlated with chins, only that it is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

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

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

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("~/chords_lyrics.rds")
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]
```

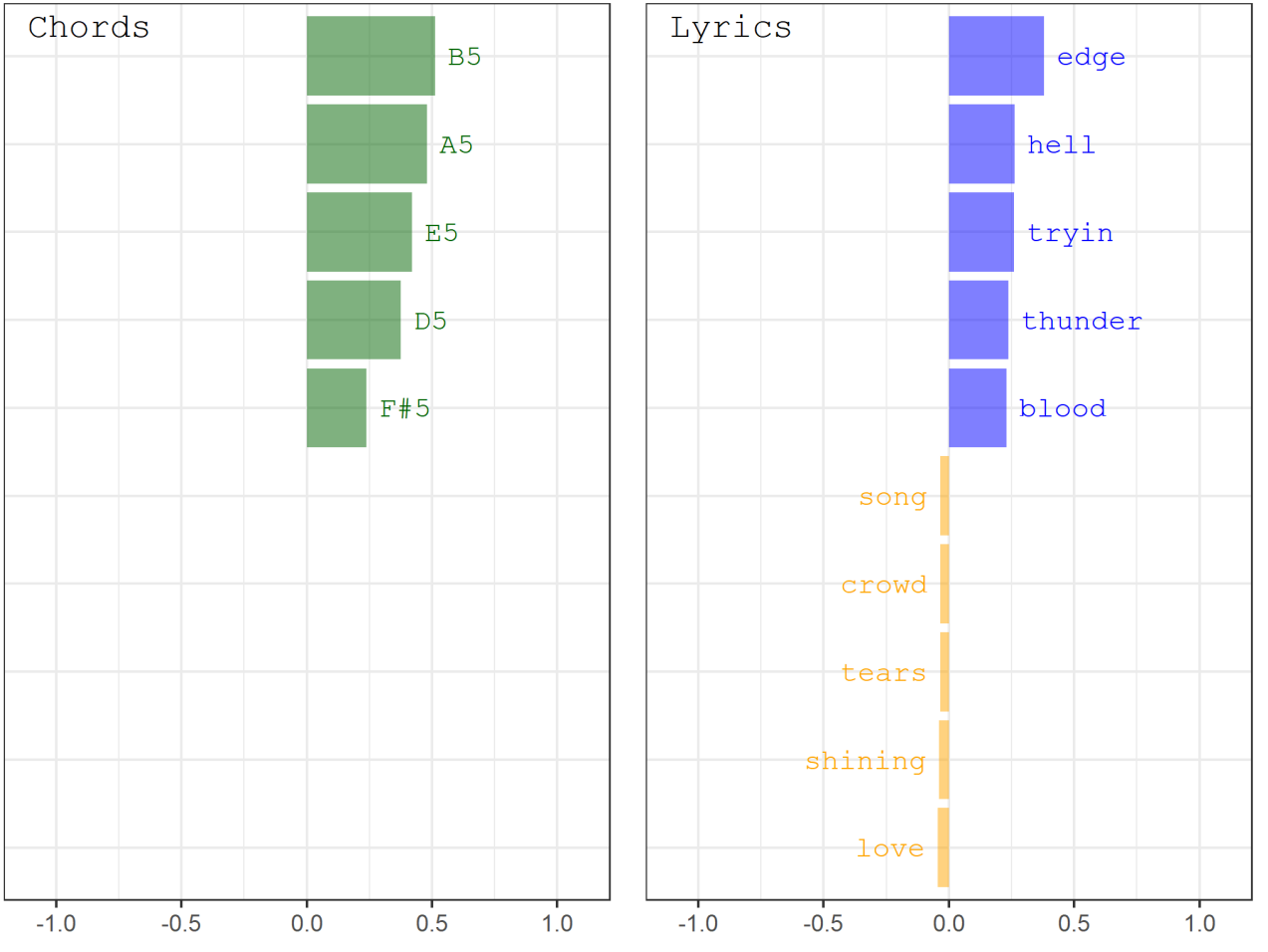
```
## [1] "edge"    "hell"    "tryin"   "thunder" "blood"
```

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

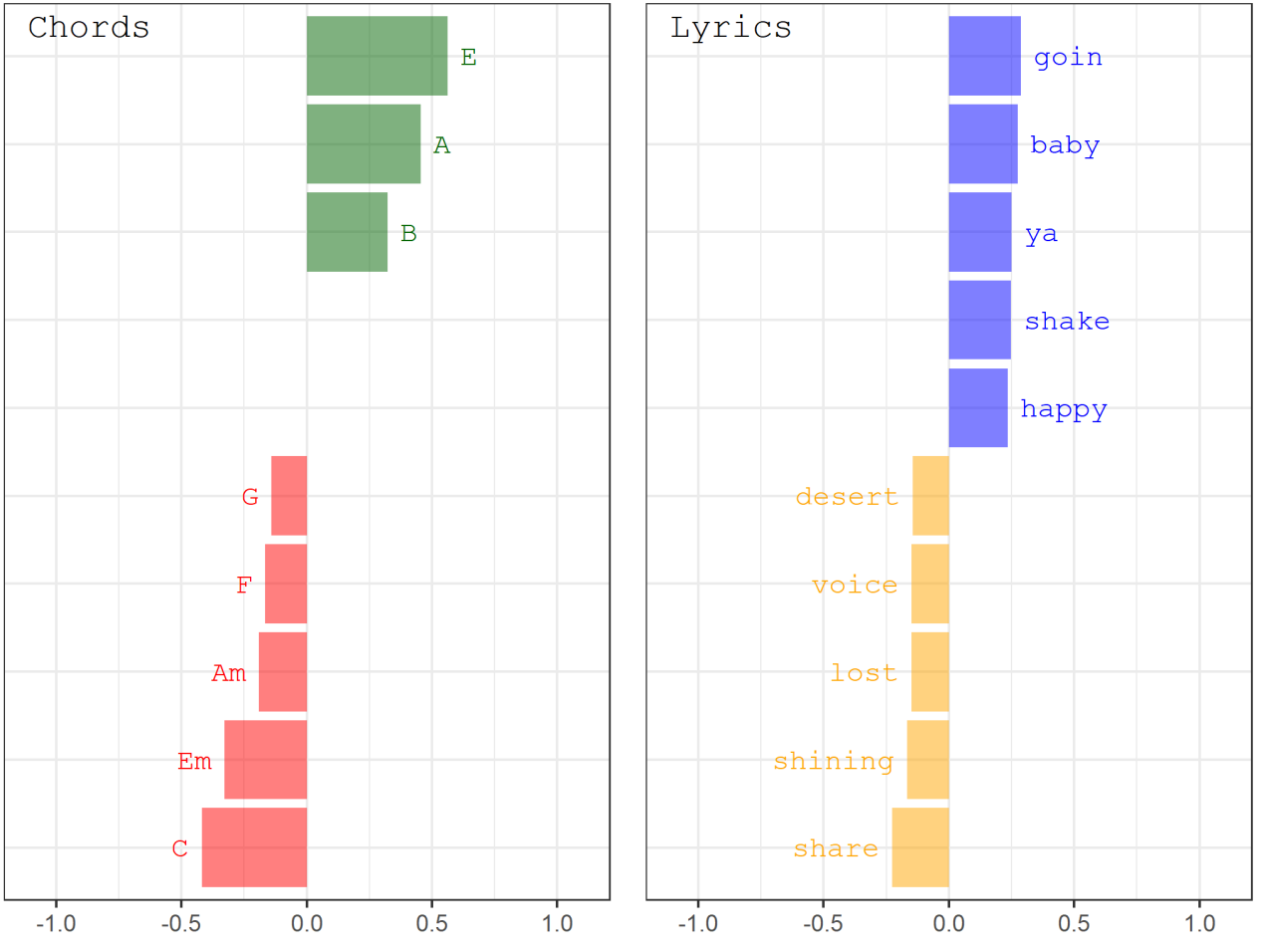
```
## [1] 0.3788166 0.2625134 0.2603274 0.2382183 0.2306617
```

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



Some more I think you should know:

- NMF (Non-negative Matrix Factorization)
- tSNE (t-distributed stochastic neighbor embedding)
- SOM (Self-Organizing Maps)
- MDS (Multidimensional Scaling)
- CA (Correspondence Analysis)
- IDC (Independent Components Analysis)
- ANOVA, ANCOVA, MANOVA, MANCOVA