STATS 415 Lab 8: Dimension Reduction

comparing these methods

Nick Seewald

9 March 2018

Partially adapted from http://www-bcf.usc.edu/~gareth/ISL/Chapter%206%20Labs.txt

1 Today's Objectives

1. Review principal components analysis and regression methdos
2. Learn how to implement PCA/PCR in R
3. Learn how to implement PLS in R
4. Compare dimension reduction methods to ridge regression and the LASSO

2 Review of Principal Components

Dimension reduction methods *transform* the original predictors in an effort to control variance. The main objective of principle components analysis (PCA) is to reduce the dimensionality of the data.

- p *original* variables are replaced with k ($k < p$) *linear combinations* of the original variables that are a "good representation" of the data
- Because we replace the original variables with linear combinations thereof, this is a *linear* dimension reduction method

Goal of PCA: Find *direction vectors* that maximize the variance of the linear combination of predictors $w^\top \boldsymbol{X}$.

2.1 Mathematical formulation of the principal components problem

- Assume original variables $X_1, X_2, \ldots, X_p$ have been centered (i.e., they've been algebraically manipulated so they have mean zero).
- Denote the covariance matrix of X by Σ.
- Find p *new* variables $Z_1, Z_2, \ldots, Z_p$ such that $Z_i = \sum_{j=1}^p w_{ij} X_j$ and the weights w maximize

$$w_i^\top \Sigma w_i \quad \text{subject to} \quad w_i^\top w_i = 1,\ w_i^\top w_j = 0.$$

Let's break this down:

- We're creating p linear combinations, denoted by Z_i, $i = 1, \ldots, p$.
- For *each* Z_i, we're choosing w_i that maximizes the variance of $w_{i1}X_1 + \ldots w_{ip}X_p$.
- Note that we're actually creating a $p \times p$ *matrix* of weights: one column per Z_i. We call this weight matrix W.
- In matrix form, the problem is equivalent to finding $Z = XW$ where W solves

p*p

$$\max_{W:W^\top W = I} W^\top \Sigma W.$$

PCA Solution: The columns of W which maximize $W^\top \Sigma W$ such that $W^\top W = I$ are the *eigenvectors* of Σ

Question: Isn't the goal of dimension reduction to reduce variance? Why are we picking direction vectors which maximize variance?

2.2 Computing Principal Components in R

To perform PCA in R, we'll use a function called `prcomp`. This is available in base R: you don't need to install any packages for it!

We'll continue using the `Hitters` data set, which contains information on baseball salaries in 1986-1987. Recall that this data set has 20 variables (one of which is the response, `Salary`), and 263 complete observations. It'd be helpful to do some dimension reduction here.

```
library(ISLR)
data(Hitters)

# Eliminate all cases with missing data (this is not always best practice, but
# we're ignoring that for the purposes of this lab)
Hitters <- na.omit(Hitters)
```

```
# Create a matrix of all predictors to be used for PCA (without an intercept)
X <- model.matrix(Salary ~ ., data = Hitters)[, -1]
# Run PCA
hitPCA <- prcomp(x = X, center = T, scale = F)
```

- `x` is a matrix of original predictors
- `center` is a logical (`TRUE`/`FALSE`) variable indicating whether to center the data (transform to mean zero)
- `scale` is a logical (`TRUE`/`FALSE`) variable indicating whether to scale the data (transform to SD 1)

```
summary(hitPCA)
```

```
## Importance of components:
##                              PC1       PC2       PC3       PC4       PC5
## Standard deviation     2430.8229 285.55683 173.27486 129.81952 113.99990
## Proportion of Variance    0.9744   0.01345   0.00495   0.00278   0.00214
## Cumulative Proportion     0.9744   0.98780   0.99275   0.99553   0.99767
##                             PC6      PC7      PC8      PC9     PC10
## Standard deviation     89.29916 64.52883 36.45192 13.55489 12.65830
## Proportion of Variance  0.00131  0.00069  0.00022  0.00003  0.00003
## Cumulative Proportion   0.99898  0.99967  0.99989  0.99992  0.99995
##                            PC11     PC12    PC13  PC14  PC15  PC16   PC17
## Standard deviation     11.74609 10.68498 6.35188 4.443 2.876 1.588 0.6389
## Proportion of Variance  0.00002  0.00002 0.00001 0.000 0.000 0.000 0.0000
## Cumulative Proportion   0.99997  0.99999 0.99999 1.000 1.000 1.000 1.0000
##                          PC18   PC19
## Standard deviation     0.4819 0.1811
## Proportion of Variance 0.0000 0.0000
## Cumulative Proportion  1.0000 1.0000
```

```r
names(hitPCA)
```

```
## [1] "sdev"     "rotation" "center"   "scale"    "x"
```

- `hitPCA$sdev` contains the standard deviations of each principal component (i.e., $\mathrm{Var}(w_i^\top X)$).
- `hitPCA$rotation` contains the PC direction vectors w_i. The elements of these vectors, denoted w_{ij} are the loadings.

To check the math, we can also ask R for the eigenvectors of $\Sigma = \mathrm{Var}(X)$:

```r
X.centered <- apply(X, 2, function(x) x - mean(x))
hitEigen <- eigen(var(X.centered))
cbind("eigen" = hitEigen$vectors[, 1], "prcomp" = hitPCA$rotation[, 1])
```

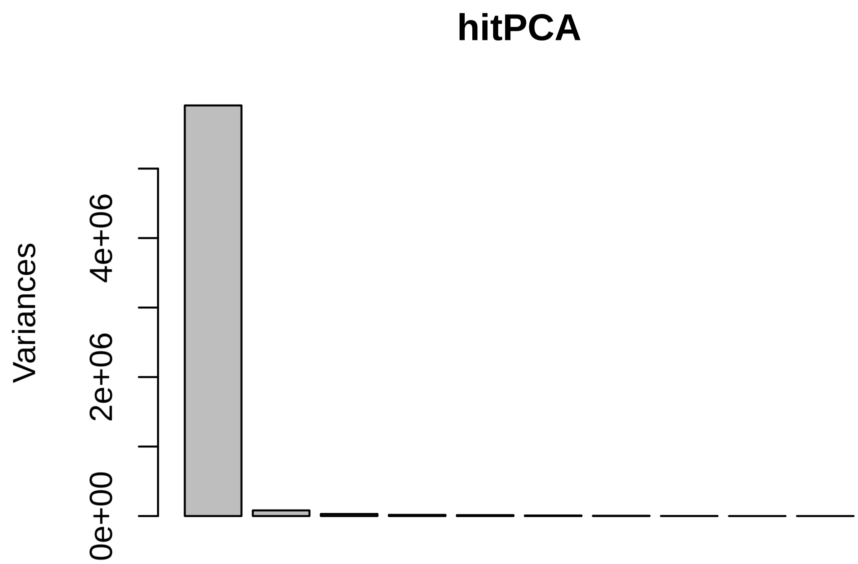
```
##                   eigen        prcomp
## AtBat      -1.272955e-02  1.272955e-02
## Hits       -3.901918e-03  3.901918e-03
## HmRun      -7.998722e-04  7.998722e-04
## Runs       -1.852363e-03  1.852363e-03
## RBI        -3.011122e-03  3.011122e-03
## Walks      -2.453800e-03  2.453800e-03
## Years      -1.802397e-03  1.802397e-03
## CAtBat     -9.405882e-01  9.405882e-01
## CHits      -2.655136e-01  2.655136e-01
## CHmRun     -2.724251e-02  2.724251e-02
## CRuns      -1.341274e-01  1.341274e-01
## CRBI       -1.267818e-01  1.267818e-01
## CWalks     -9.873265e-02  9.873265e-02
## LeagueN     5.138500e-06 -5.138500e-06
## DivisionW   4.123144e-06 -4.123144e-06
## PutOuts    -6.499663e-03  6.499663e-03
## Assists     6.517803e-04 -6.517803e-04
## Errors      1.958859e-04 -1.958859e-04
## NewLeagueN  1.111984e-06 -1.111984e-06
```

Notice that the results are *identical* up to the sign. The sign is essentially arbitrary. For more, see, e.g., https://stats.stackexchange.com/a/88882.

2.3 How many PCs should we choose?

Principle components analysis creates p PCs, one for each original predictor. But, we might not need all of them to explain a good amount of the variance in our response! We want to keep enough PCs to represent the data "well". We can use a **scree plot** to help with our choice.

```r
plot(hitPCA)
```



Question: How many PCs should we choose in this example?

3 Principal Components Regression

Principal components regression (PCR) replaces the regression model

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p + \epsilon$$

with

$$y = \beta_0 + \beta_1 z_1 + \cdots + \beta_k z_k + \epsilon,$$

where k is the number of PCs chosen for the model.

Question: What is the interpretation of β_0 in the PCR model?

We use a function called `pcr` from the `pls` package to do principal components regression.

First, we'll identify a training set.

```
set.seed(1)
train <- sample(1:nrow(Hitters), trunc(nrow(Hitters)/2))
```

```r
library(pls)
```

```
##
## Attaching package: 'pls'

## The following object is masked from 'package:stats':
##
##     loadings
```

```r
set.seed(1)
hitPCR <- pcr(Salary ~ ., data = Hitters, subset = train, scale = TRUE, validation = "CV")
```

`pcr` uses similar syntax to `lm`, with a few added options.

- `scale` is a logical variable indicating whether or not to scale the data. Note that this only divides predictors by their standard deviation; centering is *always* done automatically by `pcr`
- `validation = "CV"` performs 10-fold cross-validation error for each value of k (i.e., for each possible number of PCs included)

```r
names(hitPCR)
```

```
##  [1] "coefficients"  "scores"        "loadings"      "Yloadings"
##  [5] "projection"    "Xmeans"        "Ymeans"        "fitted.values"
##  [9] "residuals"     "Xvar"          "Xtotvar"       "fit.time"
## [13] "ncomp"         "method"        "scale"         "validation"
## [17] "call"          "terms"         "model"
```

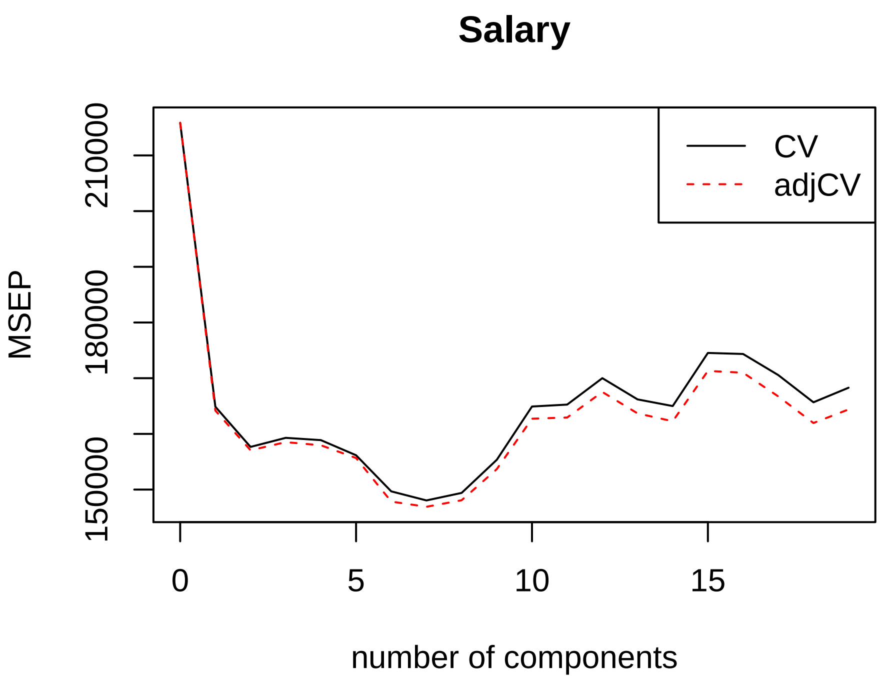
```r
summary(hitPCR)
```

```
## Data:    X dimension: 131 19
##  Y dimension: 131 1
## Fit method: svdpc
## Number of components considered: 19
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           464.6    406.1    397.1    399.1    398.6    395.2    386.9
## adjCV        464.6    405.2    396.3    398.1    397.4    394.5    384.5
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       384.8    386.5    394.1     406.1     406.5     412.3     407.7
## adjCV    383.3    384.8    392.0     403.4     403.7     409.3     404.6
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV        406.2     417.8     417.6     413.0     407.0     410.2
## adjCV     402.8     413.9     413.5     408.3     402.4     405.5
##
## TRAINING: % variance explained
##         1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X         38.89    60.25    70.85    79.06    84.01    88.51    92.61
## Salary    28.44    31.33    32.53    33.69    36.64    40.28    40.41
##         8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
## X         95.20    96.78     97.63     98.27     98.89     99.27     99.56
## Salary    41.07    41.25     41.27     41.41     41.44     43.20     44.24
##         15 comps  16 comps  17 comps  18 comps  19 comps
## X          99.78     99.91     99.97    100.00    100.00
## Salary     44.30     45.50     49.66     51.13     51.18
```

In the summary, the `VALIDATION` section gives, for each k, the 10-fold CV *ROOT* MSE. The `TRAINING` section gives "the percentage of variance explained in the predictors and in the response using different numbers of components" (James, Witten, Hastie, Tibshirani; p. 257).

We can choose k based on CV error, which we can discover by plotting the CV RMSEs.

```
validationplot(hitPCR, val.type = "MSEP", legendpos = "topright")
```



- `val.type = "MSEP"` tells `validationplot` to plot the *m*ean *s*quared *e*rror of *p*rediction
- `legendpos = "topright"` tells `validationplot` to put a legend in the top right of the plot.

Question: What value of k minimizes CV-MSE?

We can find the test MSE using `predict`, as usual.

```
hitPCR.pred <- predict(hitPCR, Hitters[-train, names(Hitters) != "Salary"], ncomp = 6)
PCRTestMSE <- mean((hitPCR.pred - Hitters[-train, "Salary"])^2)
PCRTestMSE
```

```
## [1] 96587.92
```

Recall that PCR doesn't lead to interpretation and inference in terms of the original variables; rather, both are in terms of the PCs. `pcr` can automatically compute coefficient estimates for the original predictors using the loadings and PCR coefficients.

```
# Get the "original predictor" coefficients for the PCR in which k = 1. Note the
# weird syntax: coefficients are stored as an "array"
hitPCR$coefficients[, , 1]
```

```
##       AtBat        Hits       HmRun        Runs         RBI       Walks
## 19.19513578 18.71049899 18.19172975 17.99771455 21.64354657 19.00457676
##       Years      CAtBat       CHits      CHmRun       CRuns        CRBI
## 25.19068119 30.12723642 30.35198609 28.08834292 31.08912794 30.32923972
##      CWalks     LeagueN   DivisionW     PutOuts     Assists      Errors
## 28.79612365 -1.46440822  1.12134331  6.97971627  3.25461837  3.33050252
##  NewLeagueN
##  0.04069259
```

4 Partial Least-Squares

Recall from our discussion of building principal components that we never talked about the outcome y! PC methods are *unsupervised*: they don't use information in y to determine the PC directions. Unlike PC, partial least-squares (PLS) chooses zs that are good at predicting y. This is an example of a *supervised* method: y is allowed to influence/monitor/"supervise" the construction of the direction vectors.

PLS Algorithm:

1. Center y, center *and scale* each x_j
2. Regress y on each x_j *separately*. Get a coefficent estimate α_j.
3. Construct $z_1 = \sum_{j=1}^{p} \alpha_j x_j$. This is the first PLS component.
4. Regress y on z_1 to get $\hat{\beta}_1$.
5. Orthogonalize each of the predictors x_j to z_1: regress each x_j on z_1 and replace it with the residual.
6. Return to step 2 and continue until the final model is fit:

$$\hat{y} = \bar{y} + \hat{\beta}_1 z_1 + \cdots + \hat{\beta}_k z_k.$$

4.1 Some things to remember about PLS

1. We still need to select the number of components
2. There is no interpretation for PLS coefficients!

4.2 Implementation in R

We continue to use the `pls` package, this time focusing on a function called `plsr()`. The syntax is the same as `pcr()`:

```r
set.seed(1)
hitPLS <- plsr(Salary ~ ., data = Hitters, subset = train, scale = TRUE, validation = "CV")
summary(hitPLS)
```

```
## Data:    X dimension: 131 19
##  Y dimension: 131 1
## Fit method: kernelpls
## Number of components considered: 19
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           464.6    394.2    391.5    393.1    395.0    415.0    424.0
## adjCV        464.6    393.4    390.2    391.1    392.9    411.5    418.8
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
```

```
## CV       424.5    415.8    404.6     407.1     412.0     414.4     410.3
## adjCV    418.9    411.4    400.7     402.2     407.2     409.3     405.6
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV        406.2     408.6     410.5     408.8     407.8     410.2
## adjCV     401.8     403.9     405.6     404.1     403.2     405.5
##
## TRAINING: % variance explained
##         1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X         38.12    53.46    66.05    74.49    79.33    84.56    87.09
## Salary    33.58    38.96    41.57    42.43    44.04    45.59    47.05
##         8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
## X         90.74    92.55     93.94     97.23     97.88     98.35     98.85
## Salary    47.53    48.42     49.68     50.04     50.54     50.78     50.92
##         15 comps  16 comps  17 comps  18 comps  19 comps
## X          99.11     99.43     99.78     99.99    100.00
## Salary     51.04     51.11     51.15     51.16     51.18
```

Question: What k is associated with the lowest cross-validation MSE?

2

We can get test MSE similarly to before:

```
hitPLS.pred <- predict(hitPLS, Hitters[-train, names(Hitters) != "Salary"], ncomp = 2)
PLSTestMSE <- mean((hitPLS.pred - Hitters[-train, "Salary"])^2)
PLSTestMSE
```

```
## [1] 101417.5
```

5 Comparison with ridge regression and the lasso

Let's quickly re-do cross-validated ridge regression and the lasso on our training and test data to get MSEs so we can compare them to PCR.

Recall from lab 7 how to set up the data for use with `glmnet`

```
library(glmnet)
X <- model.matrix(Salary ~ ., Hitters)[, -1]
Y <- Hitters$Salary
```

Ridge regression:

```
set.seed(1)
ridgeMod <- glmnet(X[train, ], Y[train], alpha = 0)
ridgeCV <- cv.glmnet(X[train, ], Y[train], alpha = 0)
lambda <- ridgeCV$lambda.min
lambda
```

```
## [1] 211.7416
```

Cross-validation error for ridge regression is minimized when $\lambda = 211.7$

We compute test error:

```r
ridge.pred <- predict(ridgeCV, s = lambda, newx = X[-train, ])
ridgeTestMSE <- mean((ridge.pred - Y[-train])^2)
ridgeTestMSE
```

```
## [1] 95982.96
```

The lasso:

```r
set.seed(1)
lassoMod <- glmnet(X[train, ], Y[train], alpha = 1)
lassoCV <- cv.glmnet(X[train, ], Y[train], alpha = 1)
lambda <- lassoCV$lambda.min
lambda
```

```
## [1] 16.78016
```

We compute test error:

```r
lasso.pred <- predict(lassoCV, s = lambda, newx = X[-train, ])
lassoTestMSE <- mean((lasso.pred - Y[-train])^2)
lassoTestMSE
```

```
## [1] 100838.2
```

5.1 A note about `set.seed()`

You'll notice that we're re-setting the seed (to the same thing) each time we run a cross-validated model. This is because, in order for the models to be truly comparable, we want to cross-validate them on the *same folds*.

5.2 Test errors

```r
d <- data.frame("TestMSE" = c(PCRTestMSE, PLSTestMSE, ridgeTestMSE, lassoTestMSE))
rownames(d) <- c("PCR", "PLS", "Ridge", "Lasso")
knitr::kable(d)
```

	TestMSE
PCR	96587.92
PLS	101417.46
Ridge	95982.96
Lasso	100838.20

5.3 Other things to keep in mind

Question: What are some other things to remember when we're comparing these methods?

??