Class Notes

Statistical Computing & Machine Learning

Class 17

Review

Predicting `Salary` in the `ISLR::Hitters` data:

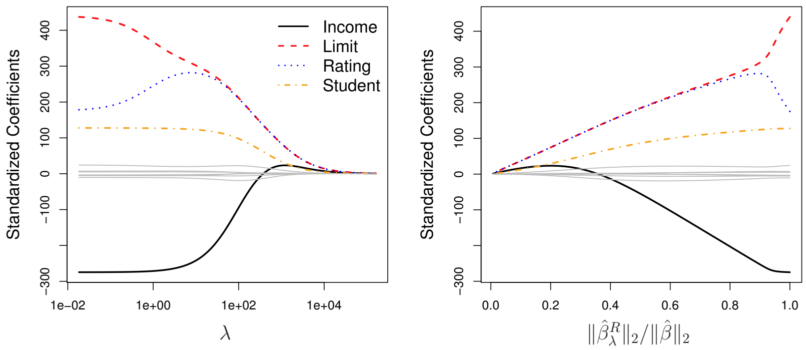
Ridge: There's other concerns in addition to fitting, e.g. the size of the coefficients.

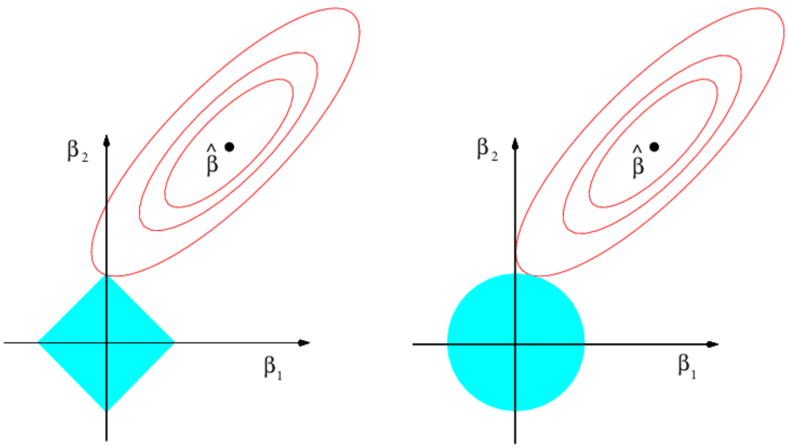
```
Without_NA <- na.omit(ISLR::Hitters)
inds <- sample(nrow(Without_NA), size = nrow(Without_NA)/2)
Train <- Without_NA[inds,]
Test <- Without_NA[-inds,]
y_all <- Without_NA$Salary
x_all <- model.matrix(Salary ~ ., data=Without_NA)
y_train <- Train$Salary
x_train <- model.matrix(Salary ~ ., data=Train)
y_test <- Test$Salary
x_test <- model.matrix(Salary ~ ., data=Test)
ridge_mod <- cv.glmnet(x_train, y_train, alpha = 0)
ridge_mod$lambda.min
ridge_pred <- predict(ridge_mod, s=0, newx = x_test, exact=TRUE)
mean((ridge_pred - y_test)^2)
final <- glmnet(x_all, y_all, alpha=0)
predict(final, type="coefficients", s=ridge_mod$lambda.min)
```

Lasso: Do we really need all of those variables?

```
lasso_mod <- cv.glmnet(x_train, y_train, alpha = 1)
lasso_mod$lambda.min
lasso_pred <- predict(lasso_mod, s=0, newx = x_test, exact=TRUE)
mean((lasso_pred - y_test)^2)
final <- glmnet(x_all, y_all, alpha=1)
predict(final, type="coefficients", s=lasso_mod$lambda.min)
```

ISLR Figure 6.4.





ISLR Figure 6.7.

Multi-collinearity

The SAT story.

```
summary(lm(sat ~ expend, data=mosaicData::SAT))$coef
```

```
##                Estimate Std. Error   t value
## (Intercept) 1089.29372  44.389950 24.539197
## expend       -20.89217   7.328209 -2.850925
##                 Pr(>|t|)
## (Intercept) 8.168276e-29
## expend      6.407965e-03
```

```
summary(lm(sat ~ expend + ratio, data=mosaicData::SAT))$coef
```

```
##                 Estimate Std. Error
## (Intercept) 1136.335547 107.803485
## expend       -22.307944   7.955544
## ratio         -2.294539   4.783836
##                 t value     Pr(>|t|)
```

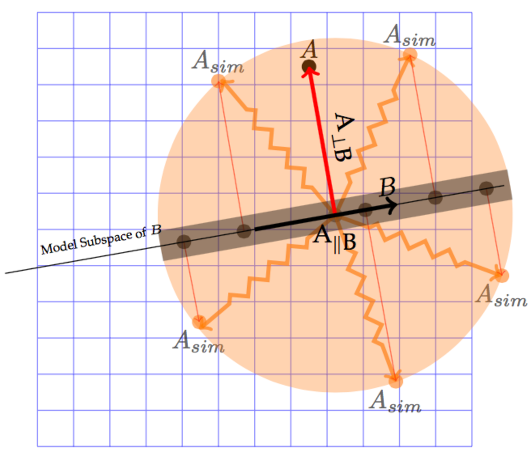
```
## (Intercept) 10.5408053 5.693212e-14
## expend      -2.8040751 7.313013e-03
## ratio       -0.4796442 6.337049e-01
```

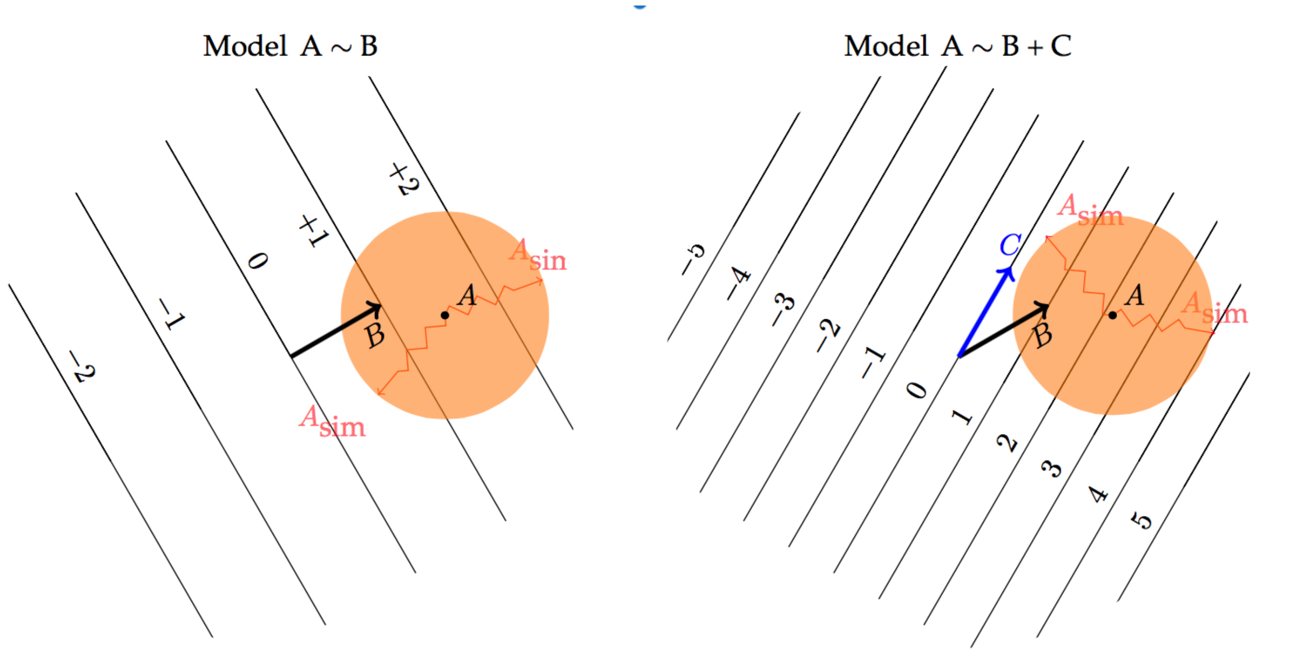
```
summary(lm(sat ~ expend + ratio + salary, data=mosaicData::SAT))$coef
```

```
##                Estimate Std. Error
## (Intercept) 1069.234168 110.924940
## expend        16.468866  22.049899
## ratio          6.330267   6.542052
## salary        -8.822632   4.696794
##                 t value     Pr(>|t|)
## (Intercept)  9.6392585 1.292219e-12
## expend       0.7468907 4.589302e-01
## ratio        0.9676272 3.382908e-01
## salary      -1.8784372 6.666771e-02
```

```
rsquared(lm(expend ~ ratio + salary, data=mosaicData::SAT))
```

```
## [1] 0.893476
```






```
load("../Daily-Programming/mona.rda")
rankMatrix(mona)
```

```
## [1] 191
## attr(,"method")
## [1] "tolNorm2"
## attr(,"useGrad")
## [1] FALSE
## attr(,"tol")
## [1] 5.551115e-14
```

```
# pick a vector at random; column 151 versus the first 131 columns
rsquared(lm(t(mona)[,151] ~ t(mona)[,-151]))
```

```
## [1] 0.9999636
```

Variance inflation factor
$\text{VIF}(\beta_j) = \frac{1}{1-R^2_{x_j|x_{-j}}}$

Getting rid of vectors that correlate substantially with one another can reduce the variance inflation factor.

Creating correlations

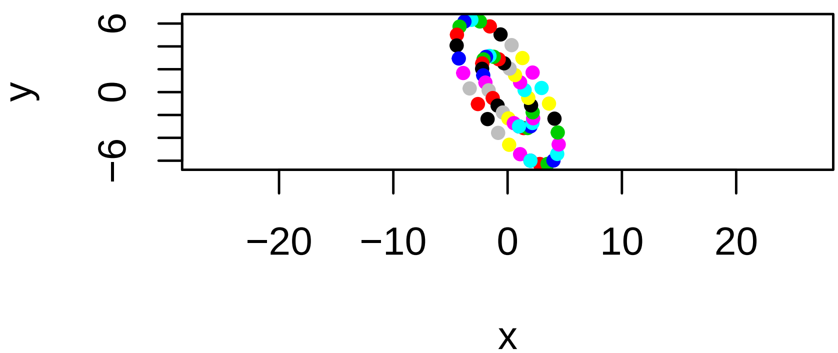
Generate points on circles of radius 1, 2, 3, . . .

```
make_circles <- function(radii = 1:2, nangs = 30) {
  theta = seq(0, 2*pi, length = nangs)
```

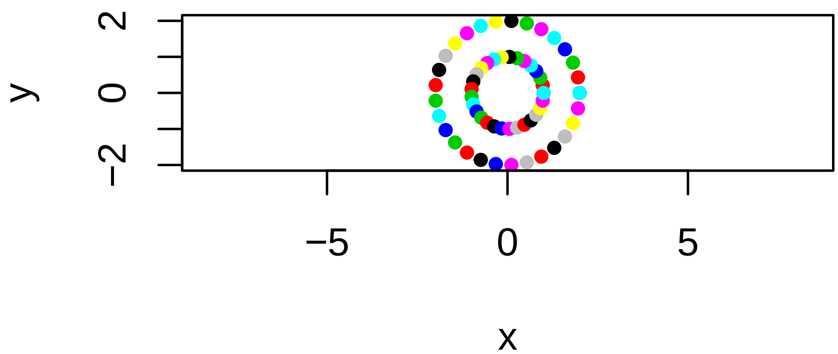
```
  x <- rep(cos(theta), length(radii))
  y <- rep(sin(theta), length(radii))
  r <- rep(radii, each = nangs)
  col <- rep(rainbow(nangs), length(radii))
  data.frame(x = x * r, y = y * r, r = r, col = col)
}
transform_circles <- function(M, circles = NULL) {
  if (is.null(circles)) circles <- make_circles()
  XY <- rbind(circles$x, circles$y)
  new <- M %*% XY
  circles$x = new[1, ]
  circles$y = new[2, ]

  circles
}
```

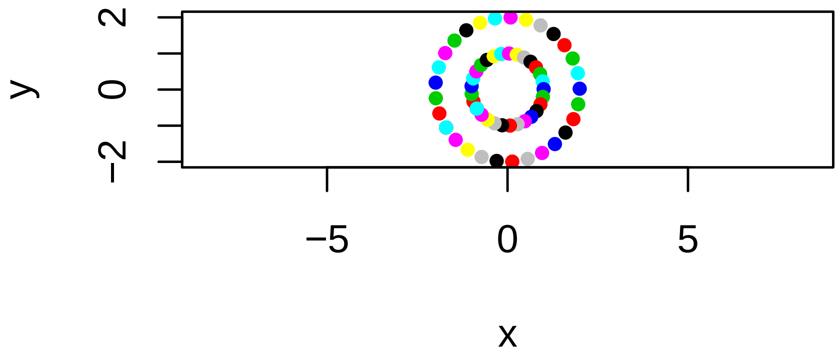
```
Trans <- matrix(c(1, 2, -3, -1), nrow = 2, byrow = TRUE)
After_trans <- transform_circles(Trans)
plot(y ~ x, data = After_trans, col = (After_trans$col), asp = 1, pch = 20)
```



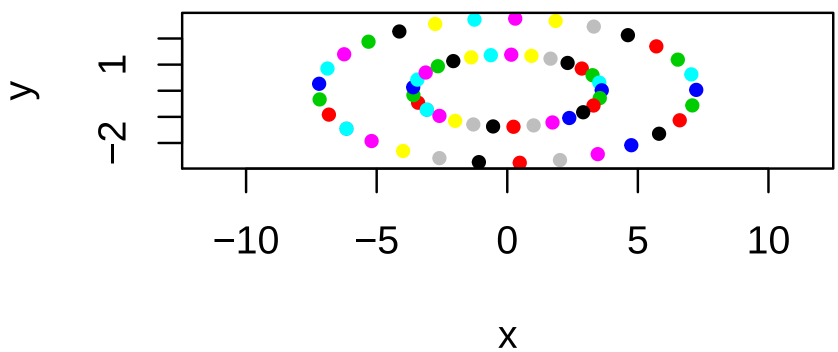

```
svals <- svd(Trans)
Start <- make_circles()
plot(y ~ x, data = Start, col = Start$col, asp = 1, pch = 20)
```



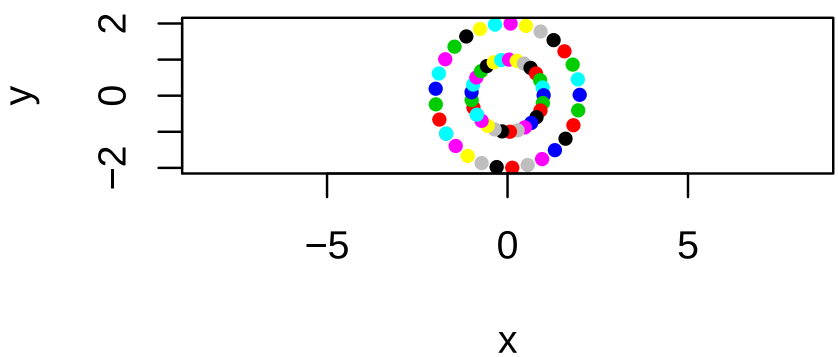
```
After_V <- transform_circles(t(svals$v), Start)
plot(y ~ x, data = After_V, col = After_V$col, asp = 1, pch = 20)
```




```
After_V_lambda <- transform_circles(diag(svals$d), After_V)
plot(y ~ x, data <- After_V_lambda, col = After_V_lambda$col, asp = 1, pch = 20)
```




```
After_V_lambda_U <- transform_circles(svals$u, After_V_lambda)
plot(y ~ x, data = After_V, col = After_V_lambda_U$col, asp = 1, pch = 20)
```

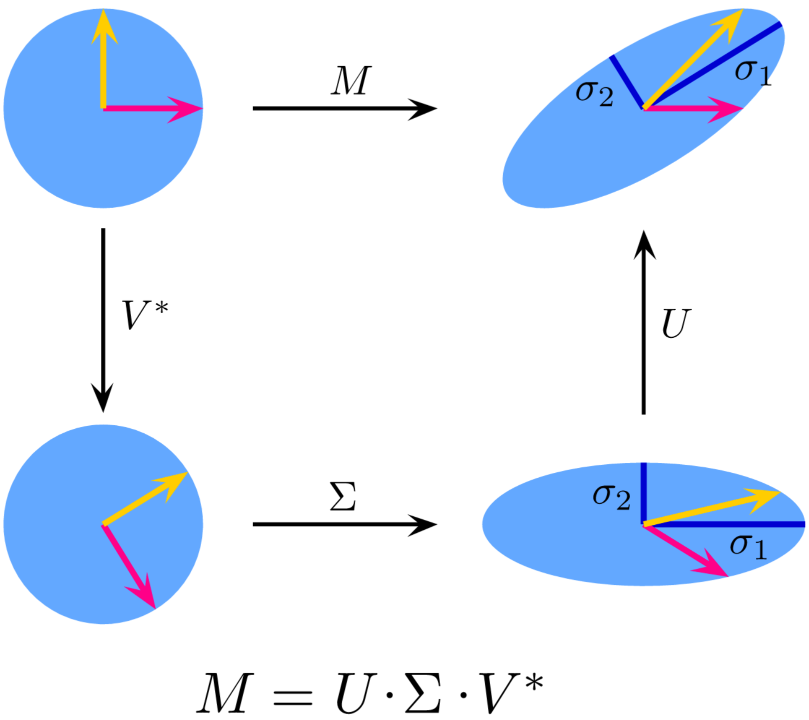


Idea of singular values.

Find orthogonal vectors to describe the ellipsoidal cloud. The singular value describes "how long" each ellipsoidal axis is.

Correlation $R^2_{x_j|x_{-j}}$ gets increased for each *direction* that overlaps between x_j and x_{-j} — it doesn't matter how big the singular value is in that direction. Only by throwing out *directions* can we reduce $R^2_{x_j|x_{-j}}$

Nice illustration:



Source

Dimension reduction

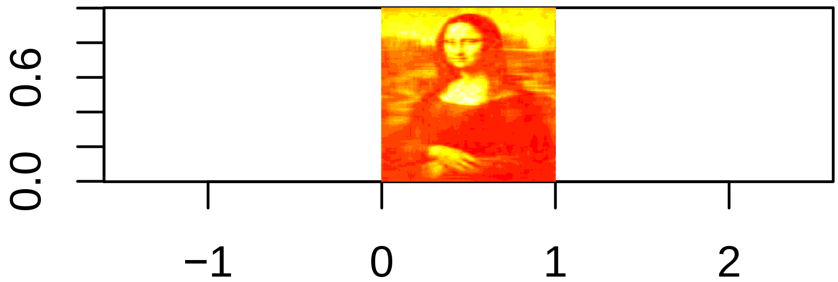
Re-arrange the variables to squeeze the juice out of them.

1. Matrix
2. Approximate matrix in a least squares sense. If that approximation includes the same column or more, we can discard the repeats.
3. Outer product
4. Rank-1 matrix constructed by creating multiples of one column.
5. Create another vector and another rank-1 matrix. Add it up and we get closer to the target.

Creating those singular vectors:

- singular value decomposition
- **D** gives information on how big they are
- orthogonal to one another
- cumulative sum of **D** components gives the amount of variance in the approximation.

```
res <- svd(mona)
approx <- 0
for (i in 1:15) {
  approx <- approx + outer(res$u[,i], res$v[,i]) * res$d[i]
}
image(approx, asp=1)
```



Picture in terms of gaussian cloud. The covariance matrix tells all that you need.

Not magic. Show the “envelope” example from Mayo.

Using `pcr()` to fit models, interpreting the output.

```
pcr.fit <- pcr(Salary ~ ., data = ISLR::Hitters, scale=TRUE, validation="CV")
summary(pcr.fit)
```

```
## Data:    X dimension: 263 19
##  Y dimension: 263 1
## Fit method: svdpc
## Number of components considered: 19
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps
## CV             452    354.0    353.0
## adjCV          452    353.6    352.6
##        3 comps  4 comps  5 comps  6 comps
## CV       352.6    350.3    346.3    344.4
## adjCV    352.1    349.7    345.6    343.6
##        7 comps  8 comps  9 comps  10 comps
## CV       344.7      348    350.4     353.2
## adjCV    343.9      347    349.2     351.7
##        11 comps  12 comps  13 comps
## CV        353.9     355.9     356.4
## adjCV     352.4     354.3     354.6
##        14 comps  15 comps  16 comps
## CV        354.4     354.8     345.6
## adjCV     352.4     352.9     343.7
##        17 comps  18 comps  19 comps
```

```
## CV         343.8     340.7     344.3
## adjCV      341.7     338.6     342.1
##
## TRAINING: % variance explained
##          1 comps  2 comps  3 comps  4 comps
## X          38.31    60.16    70.84    79.03
## Salary     40.63    41.58    42.17    43.22
##          5 comps  6 comps  7 comps  8 comps
## X          84.29    88.63    92.26    94.96
## Salary     44.90    46.48    46.69    46.75
##          9 comps  10 comps  11 comps
## X          96.28     97.26     97.98
## Salary     46.86     47.76     47.82
##          12 comps  13 comps  14 comps
## X           98.65     99.15     99.47
## Salary      47.85     48.10     50.40
##          15 comps  16 comps  17 comps
## X           99.75     99.89     99.97
## Salary      50.55     53.01     53.85
##          18 comps  19 comps
## X           99.99    100.00
## Salary      54.61     54.61
```

```
validationplot(pcr.fit, val.type = "MSEP")
```

Programming Activity

Day 15 Programming Activity. Generating data and fitting models.