

Ciencia de Datos para Políticas Públicas

Clase 07 - Selección de Modelos

Pablo Aguirre Hormann

23/09/2020

Comentario sobre tarea

- **OJO:** `install.packages(" ... ")` una vez
 - `library(...)` en cada nueva sesión de `R`

¿Qué hemos visto las últimas semanas?

- Regresión y clasificación (inferencia)
 - Dentro de muestra
- Predicción/Machine Learning
 - Fuera de muestra
- Sesgo vs varianza (*bias-variance*)
 - Complejidad/Flexibilidad
- Validación cruzada (*cross-validation*)

¿Qué NO hemos discutido?

- **¿Qué hacer cuando tenemos muchas variables predictoras (X 's)?**
- ¿Cuáles incluir en un modelo y cuáles no?
- ¿Qué criterio usamos para incluir o no una variable?
- ¿Hasta que punto agregamos variables?

¿Qué veremos hoy?

- Regresión *stepwise*
- Regularización de modelos lineales
 - LASSO
- Reducción de dimensiones (PCA)

Pero antes...

Estimaciones del error de predicción

Ajuste de datos dentro de muestra

- R^2
- Devianza

Error de predicción fuera de muestra

- Estimación directa: set de validación o *cross-validation*
- Estimación indirecta: C_p , AIC , BIC , y R_{adj}^2

Estimación indirecta

Mallow (C_p):

$$C_p = \frac{1}{n} (SCE + 2d\hat{\sigma}^2)$$

Akaike Information Criterion (AIC)

$$AIC = \frac{1}{n\hat{\sigma}^2} (SCE + 2d\hat{\sigma}^2)$$

Bayesian Information Criterion (BIC)

$$AIC = \frac{1}{n} (SCE + \log(n)d\hat{\sigma}^2)$$

d = número de predictores (x 's)

Regresión *stepwise*

Mejor subset

Dado un número p de variables, se estiman todas las combinaciones posibles de modelos. El algoritmo es el siguiente:

1. Estimar modelo nulo (M_0) correspondiente a un modelo sin predictores (x).
2. Estimar todos los posibles modelos que contienen **un solo predictor** y seleccionar el con menor SCE o mayor R^2 . Este modelo se denomina M_1 .
3. Repetir **paso 2** para modelos con dos predictores (M_2) y así sucesivamente hasta llegar al modelo con todos los predictores (M_k , **full model**).
4. De todos los modelos seleccionados ($M_0, M_1, M_2, \dots, M_k$) identificar el mejor esta vez usando *cross-validation*, C_p , AIC , BIC , o R^2_{adj} .

Limitaciones:

- Se deben calcular 2^p modelos (costo computacional)
- Sobreajuste y varianza: por azar se pueden encontrar buenos resultados (no se recomienda con más de 10 predictores)

Regresión *stepwise*

Computacionalmente más eficiente que "mejor subset". Existen distintos algoritmos: *forward*, *backward*, *hybrid*. El algoritmo del **forward stepwise** corresponde a:

1. Estimar modelo nulo (M_0) correspondiente a un modelo sin predictores (x).
2. Estimar todos los posibles modelos que contienen **un solo predictor** y seleccionar el con menor SCE o mayor R^2 . Este modelo se denomina M_1 . Hasta este punto es exactamente lo mismo que *mejor subset*.
3. Estimar todos los posibles modelos que contienen **un predictor más que M_1** y seleccionar el con menor SCE o mayor R^2 . Este modelo se denomina M_2 . Repetir hasta llegar al modelo con todos los predictores (M_k).
4. De todos los modelos seleccionados ($M_0, M_1, M_2, \dots, M_k$) identificar el mejor esta vez usando *cross-validation*, C_p , AIC , BIC , o R^2_{adj} .

Limitaciones:

- No se prueban todos los modelos por lo que puede no obtener *el mejor posible*
- Sin embargo, modelos obtenidos suelen cumplir el objetivo de evitar el *sobreajuste* y tiene buen rendimiento computacional.

Backward e Hybrid stepwise regression

Backward

- Se comienza desde el modelo M_k (*full model*) y no de M_0 .
- Se generan todos los modelos posibles eliminando una variable (M_{k-1}) y se selecciona el con menor SCE o mayor R^2 .
- Se repite el procedimiento hasta llegar al *modelo nulo* (M_0) y se identifica el mejor de todos esta vez usando *cross-validation*, C_p , AIC , BIC , o R^2_{adj}

Hybrid

- Comienza de forma similar al *forward* pero tras cada nueva incorporación se hace una prueba de extracción de variables
- Aproxima más a *mejor subset* pero sigue siendo más eficiente a nivel computacional


Forward stepwise en R

```
library(ISLR)
glimpse(Hitters)
```

```
## Rows: 322
## Columns: 20
## $ AtBat      <int> 293, 315, 479, 496, 321, 594, 185, 298, 323, 401, 574, 20 ...
## $ Hits       <int> 66, 81, 130, 141, 87, 169, 37, 73, 81, 92, 159, 53, 113, ...
## $ HmRun      <int> 1, 7, 18, 20, 10, 4, 1, 0, 6, 17, 21, 4, 13, 0, 7, 3, 20, ...
## $ Runs       <int> 30, 24, 66, 65, 39, 74, 23, 24, 26, 49, 107, 31, 48, 30, ...
## $ RBI        <int> 29, 38, 72, 78, 42, 51, 8, 24, 32, 66, 75, 26, 61, 11, 27 ...
## $ Walks      <int> 14, 39, 76, 37, 30, 35, 21, 7, 8, 65, 59, 27, 47, 22, 30, ...
## $ Years      <int> 1, 14, 3, 11, 2, 11, 2, 3, 2, 13, 10, 9, 4, 6, 13, 3, 15, ...
## $ CAtBat     <int> 293, 3449, 1624, 5628, 396, 4408, 214, 509, 341, 5206, 46 ...
## $ CHits      <int> 66, 835, 457, 1575, 101, 1133, 42, 108, 86, 1332, 1300, 4 ...
## $ CHmRun     <int> 1, 69, 63, 225, 12, 19, 1, 0, 6, 253, 90, 15, 41, 4, 36, ...
## $ CRuns      <int> 30, 321, 224, 828, 48, 501, 30, 41, 32, 784, 702, 192, 20 ...
## $ CRBI       <int> 29, 414, 266, 838, 46, 336, 9, 37, 34, 890, 504, 186, 204 ...
## $ CWalks     <int> 14, 375, 263, 354, 33, 194, 24, 12, 8, 866, 488, 161, 203 ...
## $ League     <fct> A, N, A, N, N, A, N, A, N, A, A, N, N, A, N, A, N, A, A, ...
## $ Division   <fct> E, W, W, E, E, W, E, W, W, E, E, W, E, E, E, W, W, W, W, ...
## $ PutOuts    <int> 446, 632, 880, 200, 805, 282, 76, 121, 143, 0, 238, 304, ...
## $ Assists    <int> 33, 43, 82, 11, 40, 421, 127, 283, 290, 0, 445, 45, 11, 1 ...
## $ Errors     <int> 20, 10, 14, 3, 4, 25, 7, 9, 19, 0, 22, 11, 7, 6, 8, 0, 10 ...
```

Forward stepwise en R (cont)

```
library(leaps)
mejores_modelos_forward <- regsubsets(Salary~., data = Hitters, nvmax = ncol(Hitters)-
                                     method = "forward")
summary(mejores_modelos_forward)$which
```



##	(Intercept)	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun
## 1	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
## 2	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
## 3	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
## 4	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
## 5	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
## 6	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
## 7	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
## 8	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
## 9	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE
## 10	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE
## 11	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE
## 12	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE
## 13	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE
## 14	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE
## 15	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE
## 16	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE	TRUE	FALSE

Forward stepwise en R (cont)

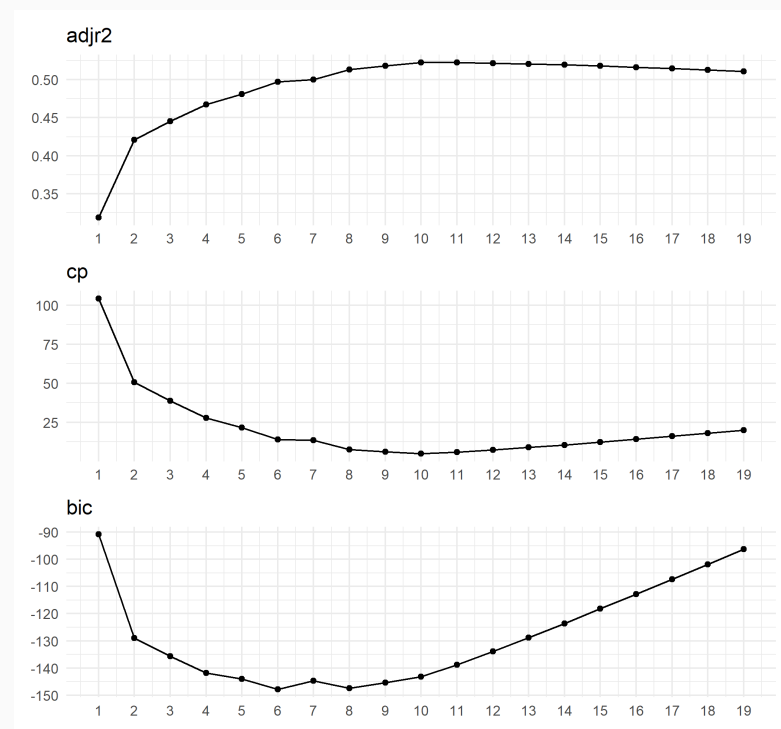
```
summary(mejores_modelos_forward) %>% names
```

```
## [1] "which" "rsq" "rss" "adjr2" "cp" "bic" "outmat" "obj"
```

```
resumen_forward <- summary(mejores_model
```

```
graf_metricas <- function(x){  
  data.frame(n_predictores = 1:19,  
             metrica = resumen_forward[[  
ggplot(aes(x = n_predictores, y = metr  
geom_line() +  
geom_point() +  
scale_x_continuous(breaks = c(0:19)) +  
theme_minimal() +  
labs(x = NULL, y = NULL, title = x)  
}
```

```
graf_forward <- map(names(resumen_forwar
```



Forward stepwise en R (cont)

```
coef(object = mejores_modelos_forward,  
      id = which.min(resumen_forward$bic))
```

```
## (Intercept)          AtBat          Hits          Walks          CRBI      DivisionW  
##  91.5117981    -1.8685892    7.6043976    3.6976468    0.6430169  -122.9515338  
##           PutOuts  
##    0.2643076
```

¿*stepwise* para inferencia?

- **Ojo:** inferencia es más que definir una métrica y encontrar el mejor modelo según esta
- Puede ser que *stepwise* deje afuera variables que teóricamente tienen sentido

Regularización de modelos lineales

Métodos de regularización

- Métodos de *subset* usan un subconjunto de las variables disponibles
 - Cada modelo se estima a través de MCO
- Métodos de regularización se estiman con todas las variables disponibles
 - Estimación es similar a MCO pero se agrega un componente que fuerza los coeficientes hacia cero
- Los métodos de regularización más usados son *LASSO* y *Ridge*
 - Nos concentraremos en *LASSO* (*Least Absolute Shrinkage and Selection Operator*)

MCO vs LASSO

Optimización MCO

$$\min_{\beta_j} \sum_{i=1}^n (y_i - \hat{\beta}_0 - \sum_{j=1}^p \hat{\beta}_j x_{ij})^2$$

$$\min_{\beta_j} SCE$$

Optimización LASSO

$$\min_{\beta_j} \sum_{i=1}^n (y_i - \hat{\beta}_0 - \sum_{j=1}^p \hat{\beta}_j x_{ij})^2 + \lambda \sum_{j=1}^p |\beta_j|$$

$$\min_{\beta_j} SCE + \lambda \sum_{j=1}^p |\beta_j|$$

- $|\beta_j|$ se conoce como *regularización L1*
- Al reemplazar *L1* por β_j^2 (*L2*) obtenemos la regularización **ridge**
- *LASSO* fuerza los coeficientes a cero (Selección de variables)

LASSO

$$\min_{\beta_j} \sum_{i=1}^n (y_i - \hat{\beta}_0 - \sum_{j=1}^p \hat{\beta}_j x_{ij})^2 + \lambda \sum_{j=1}^p |\beta_j|$$

- Si $\lambda = 0$, LASSO es equivalente a MCO
- Si $\lambda = \infty$, todos los coeficientes (β) son igual a cero y obtenemos el *modelo nulo*
- Existe un set de coeficientes, $\hat{\beta}$, para cada valor de λ
- λ nos permite "jugar" con el dilema *sesgo-varianza*
- **¿pero cómo elegir el valor de λ ?**
- *Cross-validation*

LASSO en R

```
library(glmnet)

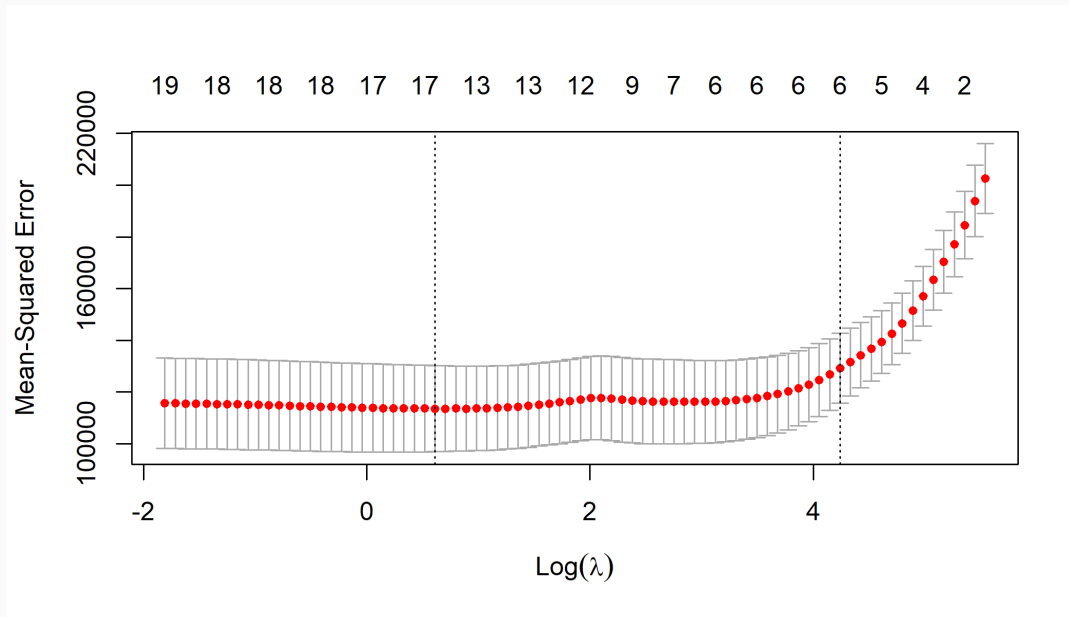
datos ← Hitters %>% filter(!is.na(Salary))

x ← model.matrix(Salary~., data = datos)[,-1]
y ← datos %>% pull(Salary)

modelos_lasso ← glmnet(x = x, y = y, alpha = 1)
plot(modelos_lasso, xvar = "lambda", label = TRUE)
```

LASSO en R (cont)

```
set.seed(1)
cv_error_lasso <- cv.glmnet(x = x, y = y, alpha = 1, nfolds = 10)
plot(cv_error_lasso)
```



```
c("min" = cv_error_lasso$lambda.min,
  "1se" = cv_error_lasso$lambda.1se)
```

```
##      min      1se
## 1.843343 69.400691
```

LASSO en R (cont)

```
modelo_final_lasso <- glmnet(x = x,  
                             y = y,  
                             alpha = 1,  
                             lambda = cv  
coef(modelo_final_lasso)
```

El modelo solo deja 6 coeficientes (más el intercepto), todos los demás son "forzados" hacia 0

```
## 20 x 1 sparse Matrix of class "dgCMatrix"  
##  
## s0  
## (Intercept) 128.19631883  
## AtBat .  
## Hits 1.42438631  
## HmRun .  
## Runs .  
## RBI .  
## Walks 1.58454271  
## Years .  
## CAtBat .  
## CHits .  
## CHmRun .  
## CRuns 0.15369557  
## CRBI 0.34300137  
## CWalks .  
## LeagueN .  
## DivisionW -8.16797602  
## PutOuts 0.08359601  
## Assists .  
## Errors .  
## NewLeagueN .
```

¿LASSO para inferencia?

- λ -por construcción- nos introduce sesgo en los $\hat{\beta}$
- Existe una corrección que se ha descrito hace relativamente poco tiempo (2006) que podría corregir esto
- *Adaptive LASSO*: se introduce una "penalidad" diferenciada a cada coeficiente y esto permitiría corregir el sesgo
 - *Oracle property*

Reducción de dimensiones

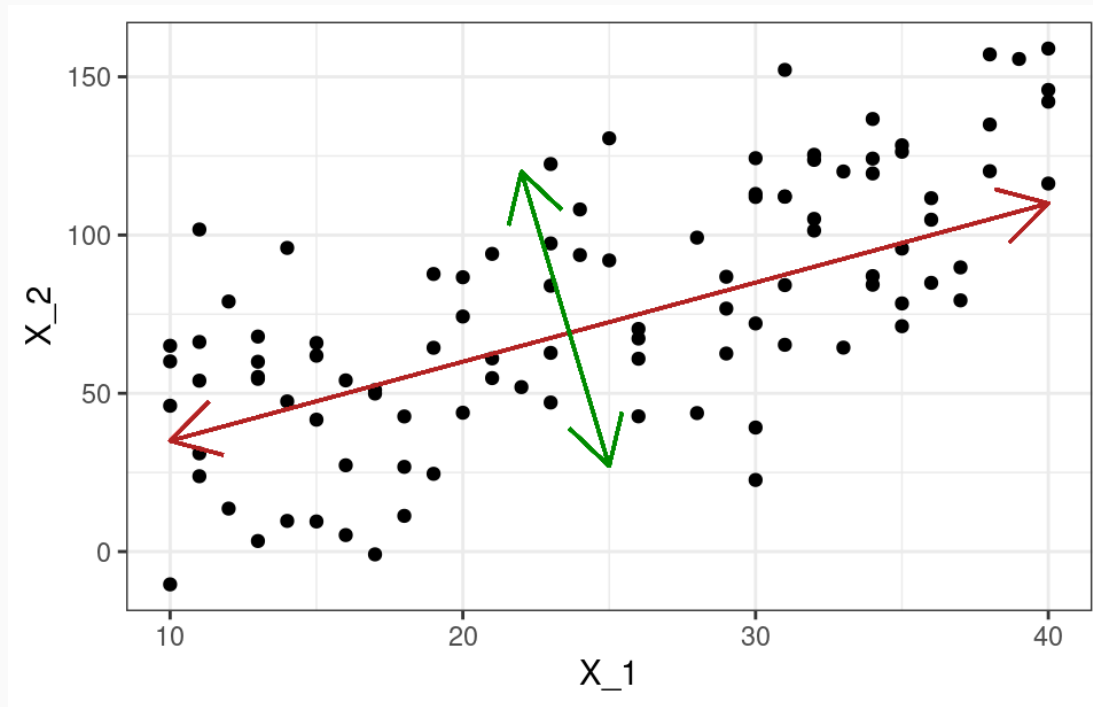
Análisis de Componentes Principales

- p variables (X_1, X_2, \dots, X_p)
- PCA nos permite encontrar $z < p$ variables que explican aproximadamente lo mismo que p
- Cada z se denomina un **componente principal**
- "Concentramos" la información de p en z
- **TODOS LOS z SON ORTOGONALES** (no están correlacionados)

Álgebra lineal de nuevo: *eigenvectors* y *eigenvalues*

El análisis de componentes principales corresponde a lo conocido como *aprendizaje no supervisado (próxima clase)* pero tiene usos en algoritmos supervisados también.

Interpretación geométrica



- La línea roja corresponde al **primer componente principal**, Z_1
- La línea verde corresponde al **segundo componente principal**, Z_2
- La proyección de los puntos en cada componente corresponde a los *principal component scores*, z_{1i} y z_{2i}

¿Qué está pasando?

Cada componente principal (Z_j) es una combinación lineal de las variables originales

Por ejemplo, la primera componente principal es la combinación lineal normalizada que captura la mayor varianza:

$$Z_1 = \phi_{11}X_1 + \phi_{12}X_2 + \dots + \phi_{1p}X_p$$

$$\sum_{j=1}^p \phi_{j1}^2 = 1$$

Los términos ϕ se conocen como *loadings* y su conjunto definen a cada componente principal. Se pueden interpretar como el peso/importancia de cada variable en el componente

- Centralizar las variables (PCA depende de varianza de cada variable)
- Buscar Z_1 a través de optimización para encontrar *loadings* que maximizan varianza (*eigenvector-eigenvalue*)
- Se repite el procedimiento para encontrar Z_2 sumando la restricción de que Z_1 y Z_2 no pueden estar correlacionados (ortogonales o perpendiculares)
- Se repite hasta encontrar p componentes principales (o $n - 1$)

Principal Component Regression

Calcular los componentes principales es un algoritmo no supervisado. Sin embargo, podemos estimar un modelo utilizando componentes principales como \mathbf{X} 's y no las variables originales (supervisado).

Ventaja es que nos libramos de cualquier correlación existente entre variables

Desventaja es que perdemos interpretabilidad

PCA en R

```
library(pls)
pca <- x %>%
  prcomp(scale = TRUE)

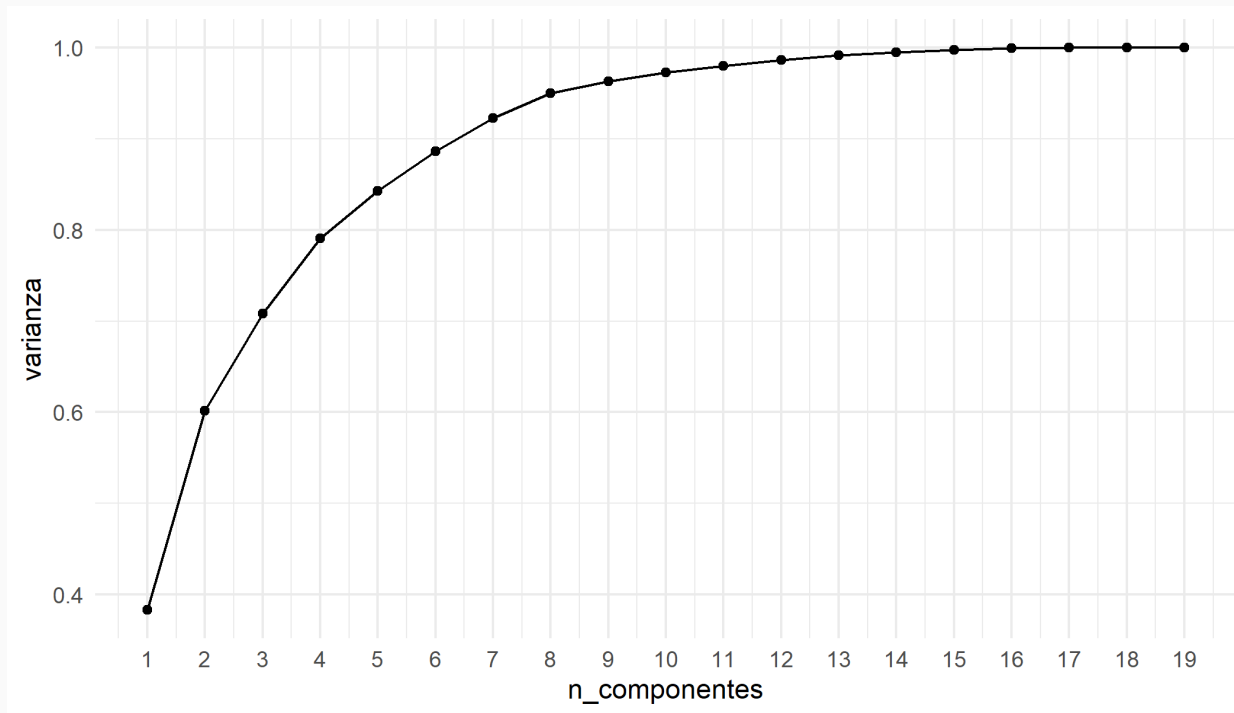
summary(pca)
```

```
## Importance of components:
```

```
##           PC1      PC2      PC3      PC4      PC5      PC6      PC7
## Standard deviation  2.6981 2.0371 1.4249 1.24763 0.99933 0.90855 0.83027
## Proportion of Variance 0.3831 0.2184 0.1069 0.08193 0.05256 0.04345 0.03628
## Cumulative Proportion 0.3831 0.6016 0.7084 0.79034 0.84290 0.88635 0.92263
##           PC8      PC9      PC10     PC11     PC12     PC13     PC14
## Standard deviation  0.7163 0.5007 0.42990 0.37047 0.35704 0.30917 0.24706
## Proportion of Variance 0.0270 0.0132 0.00973 0.00722 0.00671 0.00503 0.00321
## Cumulative Proportion 0.9496 0.9628 0.97255 0.97978 0.98649 0.99152 0.99473
##           PC15     PC16     PC17     PC18     PC19
## Standard deviation  0.22798 0.16735 0.11871 0.06973 0.03446
## Proportion of Variance 0.00274 0.00147 0.00074 0.00026 0.00006
## Cumulative Proportion 0.99747 0.99894 0.99968 0.99994 1.00000
```

PCA en R (cont)

```
prop_varianza <- cumsum(pca$sdev^2 / sum(pca$sdev^2))  
  
data.frame(n_componentes = 1:19,  
           varianza = prop_varianza) %>%  
  ggplot(aes(x = n_componentes, y = varianza)) +  
  geom_line() +  
  geom_point() +  
  theme_minimal() + scale_x_continuous(breaks = 0:19)
```



Comparar métodos

Métodos a comparar y datos

- MCO
- *Forward stepwise*
- *LASSO*
- *Principal Component Regression*

```
library(faraway)
names(meatspec)
```

```
## [1] "V1" "V2" "V3" "V4" "V5" "V6" "V7" "V8" "V9" "V10"
## [11] "V11" "V12" "V13" "V14" "V15" "V16" "V17" "V18" "V19" "V20"
## [21] "V21" "V22" "V23" "V24" "V25" "V26" "V27" "V28" "V29" "V30"
## [31] "V31" "V32" "V33" "V34" "V35" "V36" "V37" "V38" "V39" "V40"
## [41] "V41" "V42" "V43" "V44" "V45" "V46" "V47" "V48" "V49" "V50"
## [51] "V51" "V52" "V53" "V54" "V55" "V56" "V57" "V58" "V59" "V60"
## [61] "V61" "V62" "V63" "V64" "V65" "V66" "V67" "V68" "V69" "V70"
## [71] "V71" "V72" "V73" "V74" "V75" "V76" "V77" "V78" "V79" "V80"
## [81] "V81" "V82" "V83" "V84" "V85" "V86" "V87" "V88" "V89" "V90"
## [91] "V91" "V92" "V93" "V94" "V95" "V96" "V97" "V98" "V99" "V100"
## [101] "fat"
```

```
summary(meatspec$fat)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      0.90   7.30   14.00   18.14   28.00   49.10
```


Preparar los datos

```
library(tidymodels)
set.seed(1)
split <- initial_split(data = meatspec, prop = 0.8)
datos_train <- training(split)
datos_test <- testing(split)
```

```
dim(datos_train)
```

```
## [1] 173 101
```

```
dim(datos_test)
```

```
## [1] 42 101
```

MCO

```
modelo_mco <- lm(fat ~ ., data = datos_train)
glance(modelo_mco)$adj.r.squared
```

```
## [1] 0.993278
```

```
tidy(modelo_mco) %>%
  filter(abs(statistic) ≥ 1.96) %>%
  pull(term)
```

```
## [1] "(Intercept)" "V1"          "V2"          "V12"         "V15"
## [6] "V17"           "V45"         "V51"         "V52"         "V55"
## [11] "V71"           "V85"
```

```
(ecm_mco_train <- mean((modelo_mco$fitted.values - datos_train$fat)^2))
```

```
## [1] 0.4313586
```

```
(ecm_mco_test <- mean((predict(modelo_mco, newdata = datos_test) - datos_test$fat)^2))
```

```
## [1] 14.80932
```

Forward stepwise

```
datos_train_cv <- datos_train %>%  
  mutate(g = rep(1:10, length.out = nrow(datos_train)))  
  
predict.forward <- function(object, newdata) {  
  form <- as.formula(object$call[[2]])  
  mat <- model.matrix(form, newdata)  
  coefi <- coef(object, id = id)  
  xvars <- names(coefi)  
  mat[, xvars] %*% coefi  
}  
  
error_matrix <- matrix(data = NA, nrow = 10, ncol = 100,  
  dimnames = list(letters[1:10], 1:100))
```

```
for (k in 1:10) {  
  train <- datos_train_cv %>% filter(g == k)  
  mejores_modelos <- regsubsets(fat ~  
    data = train[, -1],  
    nvmax = 100,  
    method = "cv")  
  
  for (i in 1:100) {  
    test <- datos_train_cv %>% filter(g != k)  
    predicciones <- predict.forward(object = mejores_modelos[[k]],  
      newdata = test[, -1])  
  
    error_matrix[k,i] <- mean((test$fat - predicciones)^2)  
  }  
}  
  
mean_cv_error <- apply(X = error_matrix, MARGIN = 2, FUN = mean)
```

Forward stepwise (cont)

```
(mejor_modelo <- error_matrix %>%  
  as_tibble() %>%  
  summarise_all(mean) %>%  
  which.min())
```

```
## 14
```

```
## 14
```

```
modelo_final <- regsubsets(fat ~ ., data = datos_train, nvmax = 100,  
                          method = "forward")  
coef(object = modelo_final, mejor_modelo)
```

```
## (Intercept)          V1          V13          V18          V20          V30  
##      8.702953 -361.054469 2174.805558 -3621.488424 1753.760014 1182.197615  
##           V35          V37          V40          V41          V43          V47  
## -7740.039713 8537.500001 2647.196141 -6756.535402 2204.236608 3189.120522  
##           V48          V55          V57  
## -3514.026157 452.254632 -148.009624
```

```
(ecm_forward_test <- mean((predict.forward(modelo_final, datos_test, id = mejor_modelo
```

```
## [1] 9.13784
```

LASSO

```
set.seed(1)

datos_train_x ← model.matrix(fat~., data = datos_train)[,-1]
datos_train_y ← datos_train %>% pull(fat)
datos_test_x ← model.matrix(fat~., data = datos_test)[,-1]
datos_test_y ← datos_test %>% pull(fat)

cv_error_lasso ← cv.glmnet(x = datos_train_x, y = datos_train_y, alpha = 1,
                           nfolds = 10,
                           type.measure = "mse")

modelo_lasso ← glmnet(x = datos_train_x, y = datos_train_y, alpha = 1,
                      lambda = cv_error_lasso$lambda.1se)

predicciones ← predict(object = modelo_lasso, newx = datos_test_x,
                        s = cv_error_lasso$lambda.1se, exact = TRUE)

(ecm_lasso_test ← mean((predicciones - datos_test_y)^2))

## [1] 16.91051
```

PCA

```
pca <- datos_train %>% prcomp(scale = TRUE)
summary(pca)$importance[, 1:15]
```

```
##              PC1      PC2      PC3      PC4      PC5
## Standard deviation  9.938559 1.087727 0.9162659 0.4123008 0.1594651
## Proportion of Variance 0.977970 0.011710 0.0083100 0.0016800 0.0002500
## Cumulative Proportion 0.977970 0.989680 0.9980000 0.9996800 0.9999300
##              PC6      PC7      PC8      PC9      PC10
## Standard deviation  0.06192057 0.04262364 0.02770417 0.01915117 0.007838513
## Proportion of Variance 0.00004000 0.00002000 0.00001000 0.00000000 0.00000000
## Cumulative Proportion 0.99997000 0.99999000 0.99999000 1.00000000 1.00000000
##              PC11      PC12      PC13      PC14
## Standard deviation  0.006148035 0.004277114 0.003535942 0.002014743
## Proportion of Variance 0.000000000 0.000000000 0.000000000 0.000000000
## Cumulative Proportion 1.000000000 1.000000000 1.000000000 1.000000000
##              PC15
## Standard deviation  0.00129544
## Proportion of Variance 0.00000000
## Cumulative Proportion 1.00000000
```

PCA (cont)

```
set.seed(1)
modelo_pcr <- pcr(formula = fat ~ ., data = datos_train,
                  scale. = TRUE, validation = "CV")
validationplot(modelo_pcr, val.type = "MSE")
```

PCA (cont)

```
modelo_pcr$validation$PRESS %>% which.min()
```

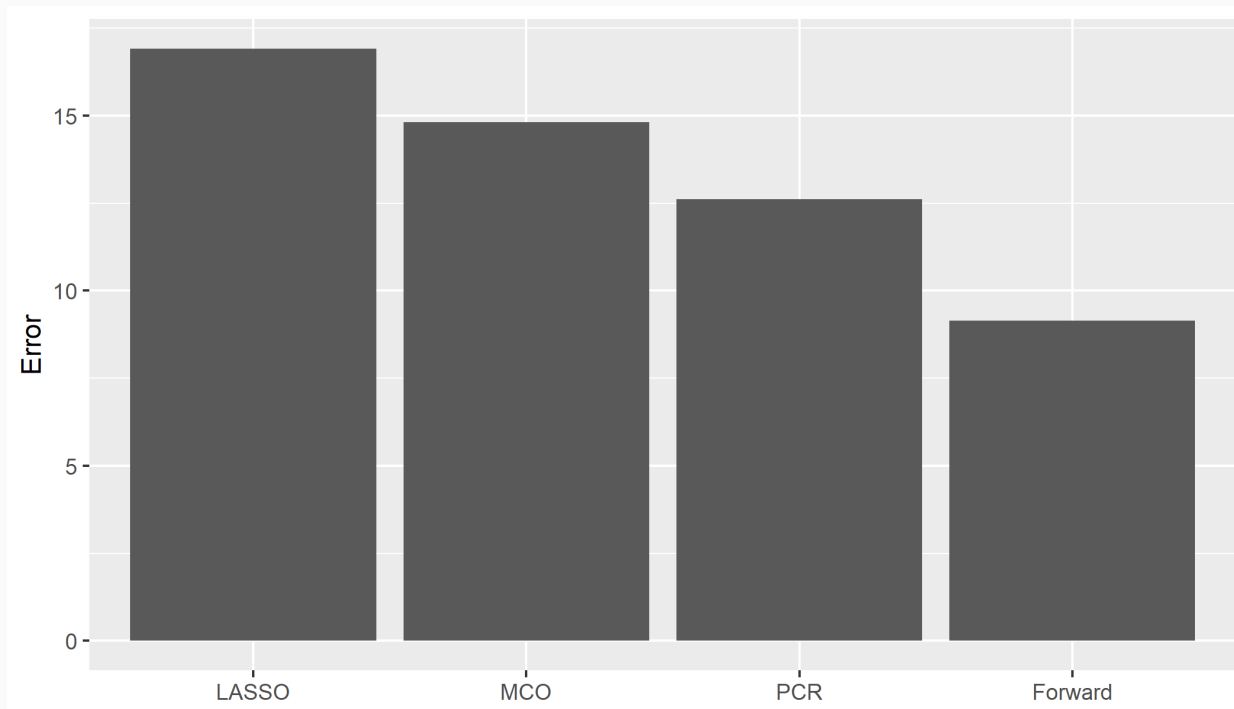
```
## [1] 20
```

```
(ecm_pcr_test ← mean((predict(modelo_pcr, newdata = datos_test,  
                             ncomp = 20) - datos_test$fat)^2))
```

```
## [1] 12.62108
```


Comparación

```
data.frame(Error = c(ecm_mco_test, ecm_forward_test, ecm_lasso_test, ecm_pcr_test),  
           Modelo = c("MCO", "Forward", "LASSO", "PCR")) %>%  
  ggplot(aes(x = reorder(Modelo, -Error), y = Error)) +  
  geom_col() +  
  labs(x = NULL)
```



Mensajes importantes

- Inferencia vs predicción
- Probar distintos algoritmos
- Train/Test/CV

Siguientes clases

- Aprendizaje no supervisado
- Árboles de decisión