

Lec4: Regularización

Isidoro Garcia Urquieta

2021

Agenda

- ▶ In sample (IS) vs Out of Sample (OOS)
- ▶ Trade-off Sesgo Varianza
- ▶ Criterios de Información (AIC, BIC)
- ▶ Error cuadrático medio
- ▶ Binomial Deviance
- ▶ Least Absolute Selection and Shrinkage Operator (LASSO)
- ▶ Elastic Nets

Estadística moderna

La clase pasada vimos que la Regresión lineal es BLUE bajo las condiciones Gauss-Markov. Esto significa que la regresión es el estimador con menor Error Cuadrático Medio **dentro de los estimadores insesgados**.

$$ECM = E[(y_i - \hat{y})^2] = var(\hat{y}) + sesgo^2(\hat{y})$$

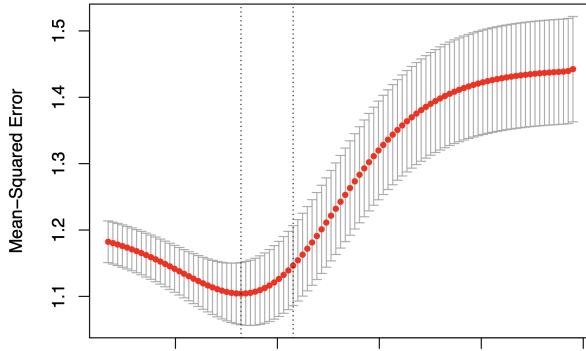
Más aún, vimos que FDR nos puede permitir seleccionar que variables son las más relevantes (señales).

En la estadística moderna se reconocen 2 variaciones a la mirada clásica de BLUE:

- ▶ Las predicciones importantes son **fuera de la muestra**. Esto genera la necesidad de encontrar algoritmos de re-muestreo para simular el comportamiento del modelo con datos no vistos.
- ▶ Podemos sacrificar un poco de riesgo por (ojalá) poca varianza. Así terminaremos con un mejor estimador.

Bias Variance Tradeoff

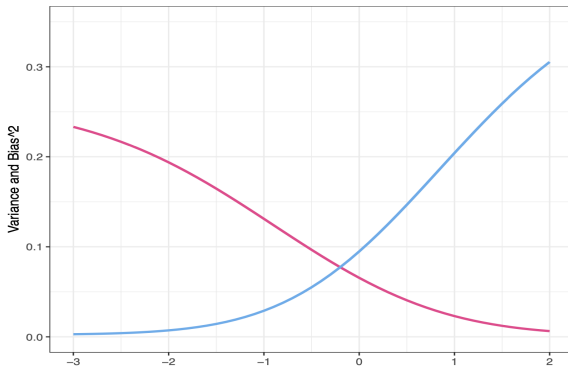
EL Error Cuadrático Medio (y) normalmente tiene una relación de U vs la sencillez del modelo.



El eje de las abcisas es la **sencillez** del modelo

Bias Variance Tradeoff

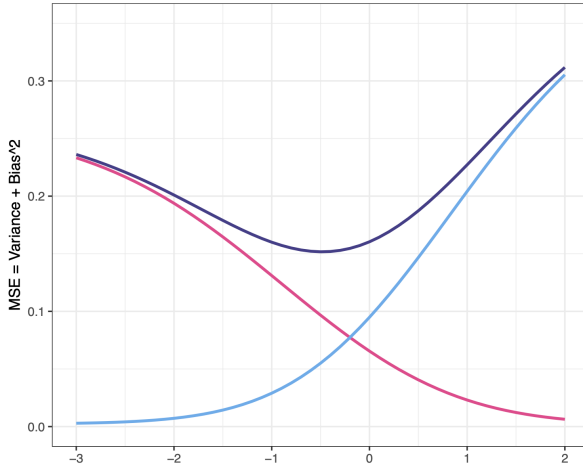
Resulta que este MSE es producto de $var(\hat{y})$ y del $sesgo^2(\hat{y})$



Rosa: $var(\hat{y})$, azul: $sesgo^2(\hat{y})$

Bias Variance Tradeoff

Podemos sacrificar un poco de sesgo para disminuir el error de predicción.



Métricas del desempeño del modelo

Hasta ahora hemos hablado del *ECM* de una manera algo informal. En realidad, el *ECM* se inserta en una variedad de estadísticos de prueba dependiendo del problema a resolver:

La variable a predecir es **continua**

- Error Cuadrático Medio: $ECM = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y})^2$
- Deviance $dev(\hat{y}) = \sum_{i=1}^N (y_i - \hat{y})^2$.

Noten como minimizar deviance es equivalente a Maximizar la Máxima Verosimilitud:

$$\text{Verosimilitud: } \max \sum_{i=1}^N \ln(f(x_i|\beta)) \iff \min dev(\hat{\beta}) = -2 \sum_{i=1}^N \ln(f(x_i|\beta))$$

Noten como en el caso de una regresión, $R^2 = 1 - \frac{dev(\hat{y})}{dev(\beta=0)}$.

Como dijimos, sólo nos va a importar el R^2 , $dev(\hat{y})$, $ECM(\hat{y})$ FUERA de la muestra (OOS).

Métricas de desempeño del modelo

Cuando el problema es de **clasificación** o y es categórica:

- ▶ Binomial Deviance: $\sum_{i=1}^N -2[y_i \ln(\hat{p}_i) + (1 - y_i) \ln(1 - \hat{p}_i)]$
- ▶ Matriz de confusion: En la siguiente clase!
- ▶ Área bajo la curva ROC (AUC): En la siguiente clase!

Selección del modelo adecuado

Hasta ahora vimos que la regresión lineal puede servir mucho para hacer predicciones. No obstante, seleccionar al modelo adecuado es complicado:

1. Y si usamos FDR como seleccionador de variables?

Ejemplo Libro Taddy: Semiconductores

Veamos un ejemplo de la manufactura de semiconductores. Queremos predecir si el semiconductor va a fallar. El problema es complicado porque un semiconductor fallido puede generar problemas en toda una planta.

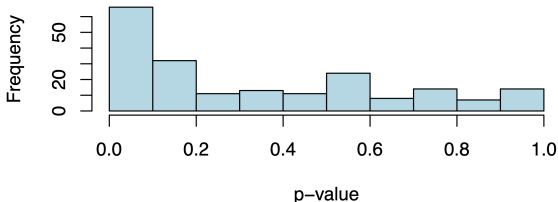
Tenemos una base de x_i de 200 columnas y una y_i binaria de (*pass*, *fail*). En la base de datos hay 100/1477 casos de semiconductores fallidos.

La regresión logística de la falla es:

$$p_i = P(\text{fail}_i | x) = \frac{e^{\beta_0 + x_i' \beta}}{1 + e^{\beta_0 + x_i' \beta}}$$

Semiconductores, FDR control

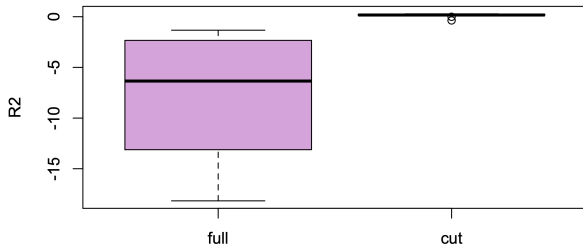
Si incluimos las 200 columnas en el modelo, nos da una $R^2 = 0.56$. Así se ven los p-values de las 200 variables:



Si corremos FDR al $q = 0.1$, nos arroja $\alpha = 0.0122$ de punto de corte. Esto genera 25 variables significativas, de las cuales 22-23 son verdaderas señales y 2-3 son ruido.

Semiconductores, FDR control

La $R^2_{FDR, cut} = 0.18$ vs $R^2 = 0.56$. Ambas *In-sample*. Como se ven en OOS?



Para el modelo FDR, cut la $R^2_{OOS} = 0.09$. La mitad de lo que observamos *In-sample*. Vean la R^2_{OOS} !! Qué significa que sea negativa?

FDR como seleccionador

En general, es mejor usar FDR que no usar nada de control de falsos positivos. Sin embargo para la selección de modelos puede tener algunas limitantes:

- ▶ FDR asume independencia entre las pruebas.
- ▶ Si tenemos más columnas que filas $p > n$, FDR ni siquiera es posible de correr.

Necesitamos otros algoritmos para seleccionar variables.

Training y Validation (Test) Sets

Antes de entrar a estos algoritmos hablemos un poco algunos tips prácticos de la división de las bases:

1. Dividimos los datos **aleatoriamente** entre training y validation sets. En algunos modelos (Neural Networks), se requiere un tercer set llamado test set. Esto esta fuera del scope de esta clase.
2. Estimamos el modelo en el training set.
3. Validamos en el validation set.

Pero, que porcentajes utilizamos? DEPENDE.



Training y Validation (Test) Sets

Típicamente queremos tener la mayoría de los datos en el training set. Esto hará que la varianza de nuestros estimadores sea baja. Por otro lado, un validation set muy pequeño puede generar tests muy ruidosos. Les dejo unas reglas prácticas de dedo:

- ▶ Empieza por (70/75/80 training, 30/25/20 validation).

Training y Validation (Test) Sets

Típicamente queremos tener la mayoría de los datos en el training set. Esto hará que la varianza de nuestros estimadores sea baja. Por otro lado, un validation set muy pequeño puede generar tests muy ruidosos. Les dejo unas reglas prácticas de dedo:

- ▶ Empieza por (70/75/80 training, 30/25/20 validation).
- ▶ Si tienes pocas observaciones, sube el número de observaciones en el training set. Esto hará que los estimadores converjan.

Training y Validation (Test) Sets

Típicamente queremos tener la mayoría de los datos en el training set. Esto hará que la varianza de nuestros estimadores sea baja. Por otro lado, un validation set muy pequeño puede generar tests muy ruidosos. Les dejo unas reglas prácticas de dedo:

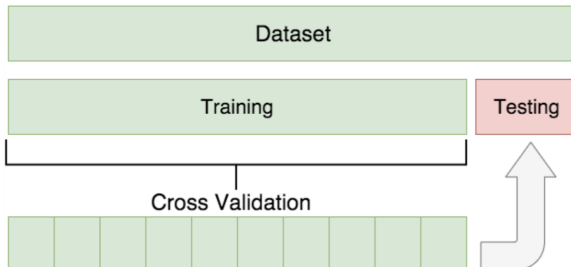
- ▶ Empieza por (70/75/80 training, 30/25/20 validation).
- ▶ Si tienes pocas observaciones, sube el número de observaciones en el training set. Esto hará que los estimadores converjan.
- ▶ Toma en cuenta que tan intensivo en poder de cómputo es tu algoritmo. Uno muy pesado (i.e. un Random Forest) te puede invitar a bajar un poco el tamaño del training set.

Resampling methods

Hoy vamos a ver *Cross Validation* como primer método de prueba.

La idea es la siguiente:

- ▶ Divide los datos en muestra de entrenamiento y validación.
- ▶ Hay alguna forma de utilizar mi muestra de entrenamiento varias veces para escoger el mejor modelo antes de ir a la validación? si!

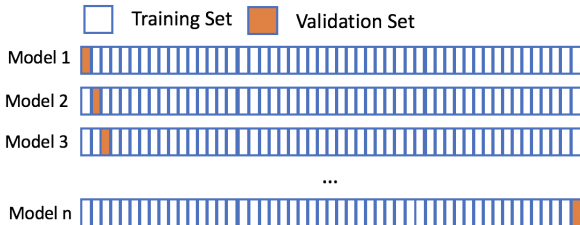


Cross Validation

Cross-validation te dejan repetir para cada candidato a modelo la estimación para elegir al mejor. Existen dos tipos formas de hacer Cross-Validation:

- ▶ **Leave One out Cross Validation:** Estimás el modelo en todas las observaciones de entrenamiento excepto una. Validas en la observación restante. Repites el ejercicio n veces.
- ▶ **K-fold cross validation:** Divides la muestra de k -partes (usualmente 5). Estimás el modelo en $k - 1$ partes, validas en la parte restante. Repites el ejercicio k veces.
- ▶ **K-Cross Fitting:** Fundamento de causal machine learning!

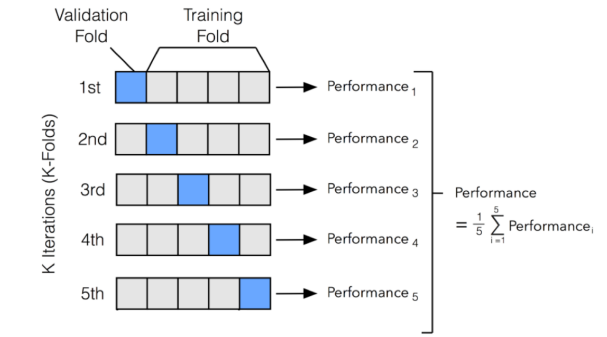
LOOCV



- ▶ Ventaja: Es un método muy general. Se puede utilizar en cualquier modelo predictivo.
- ▶ Desventaja: Puede ser computacionalmente muy complicado, ya que estimas un modelo n veces.

$$CV_{(n)} = \sum_{i=1}^n \text{error rate}$$

K-fold Cross Validation



$$CV_{(k)} = \sum_{i=1}^k \text{error rate}$$

K-fold vs LOOCV

En la práctica, K-fold con $k = 5, 10$ ha mostrado ser muy bueno identificando en **que** flexibilidad del modelo se minimiza el error rate (*ECM, binomial deviance, etc*) igual de bien que *LOOCV* pero siendo mucho más fácil de estimar computacionalmente. Esto es, el *k-fold CV* logra balancear bien el bias-variance trade-off

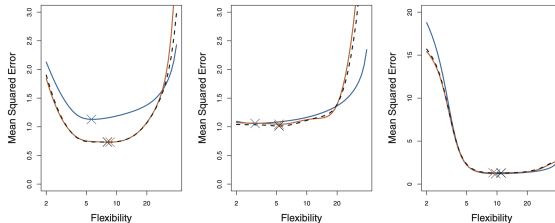


FIGURE 5.6. True and estimated test MSE for the simulated data sets in Figures 2.9 (left), 2.10 (center), and 2.11 (right). The true test MSE is shown in blue, the LOOCV estimate is shown as a black dashed line, and the 10-fold CV estimate is shown in orange. The crosses indicate the minimum of each of the MSE curves.

Regularización

Al fin llegamos a regularización! La idea general es:

- ▶ Imponer **restricciones pertinentes** a las $\hat{\beta}$ para hacerlos *estables y significantes*:

A qué nos referimos con restricciones pertinentes?

- ▶ Que $\hat{\beta}$ sea menos variable
- ▶ Que $\hat{\beta}$ ayude a arrojar mejores predicciones de y_i
- ▶ Que $\hat{\beta}$ tenga $\hat{\beta} = 0$ donde las variables sean ruido

En todos los métodos se intenta minimizar una función así:

$$\min \sum_{i=1}^n (y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij})^2 + \lambda \sum_{j=1}^p c(\beta_k)$$

Noten como esta función es parecida a la de mínimos cuadrados, sólo que hay un término nuevo. A qué se parece?

Regularización

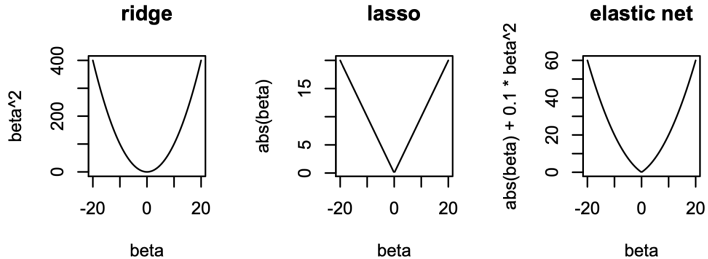
Existen 3 métodos de regularización principales:

1. **Ridge**: $\min \sum_{i=1}^n (y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij})^2 + \lambda \sum_{j=1}^p \beta_j^2$
2. **Least Absolute Selector and Shrinkage Operator (LASSO)**:
 $\min \sum_{i=1}^n (y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij})^2 + \lambda \sum_{j=1}^p |\beta_j|$
3. **Elastic Nets**: $\min \sum_{i=1}^n (y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij})^2 + \lambda \sum_{j=1}^p (\alpha \beta_j^2 + |\beta_j|)$

Cada una tiene una función de penalización de inclusión de variables distinta. λ es el tuning parameter. Esto es lo primero que es machine learning. Hay un parámetro que no se encuentra en la estimación. Tenemos que tunearlo con alguna técnica de re-muestreo.

Funciones de penalización

Graficamente así se ven estas funciones de penalización:



Escala importa

En todos los algoritmos de regularización la escala importa mucho. Es por ello que en todas las aplicaciones de R hay una opción `standardize=TRUE` que iguala las escalas para todas las variables.

$$x' = \frac{x - \bar{x}}{sd(x)}$$

Ridge Regression

$$\min \sum_{i=1}^n (y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij})^2 + \lambda \sum_{j=1}^p \beta_j^2$$

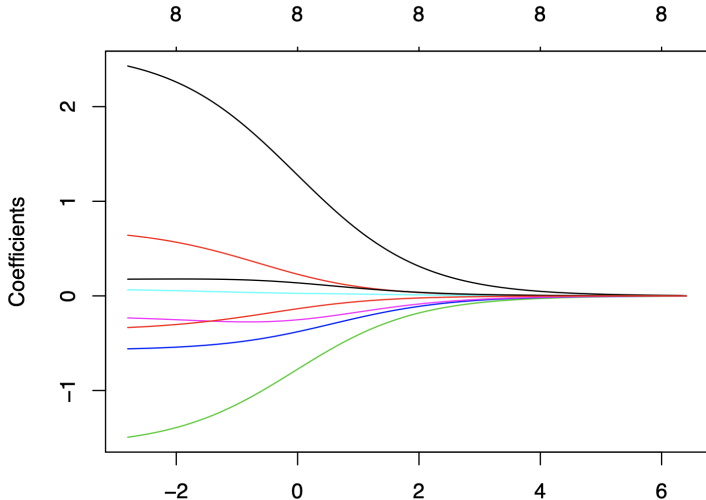
Noten como si $\lambda = 0$ estamos en OLS. Al incluir más variables (algunas ruido), los estimadores ganan más varianza.

En el escenario donde $\lambda > 0$ tenemos una penalidad por el *tamaño* de los coeficientes a estimar. Esto es, el algoritmo debe balancear entre minimizar los residuales y el costo de incluir variables.

Por qué esto es relevante? Por que Ridge cambia el tamaño de los estimadores (aumentando el sesgo) en pro de mejorar el *ECM* total. Así el Ridge regression ayuda a mejorar el poder predictivo de OLS.

Ridge Regression

Esto es el el Ridge path. Vean como mientras más sube λ , o castigamos la complejidad del modelo, los coeficientes tienden a cero.



Ridge Regression

Pero, ridge te hace los coeficientes exactamente cero (Selección de variables)? No! Es costo marginal de incluir una variable cuando tenemos que $\beta = 0$ es $\frac{\partial}{\partial \beta_k} \sum_{j=1}^p \beta_k^2 = 2\lambda \beta_k$. Es decir, no hay costo de incluir variables. Simplemente las regulariza!

LASSO

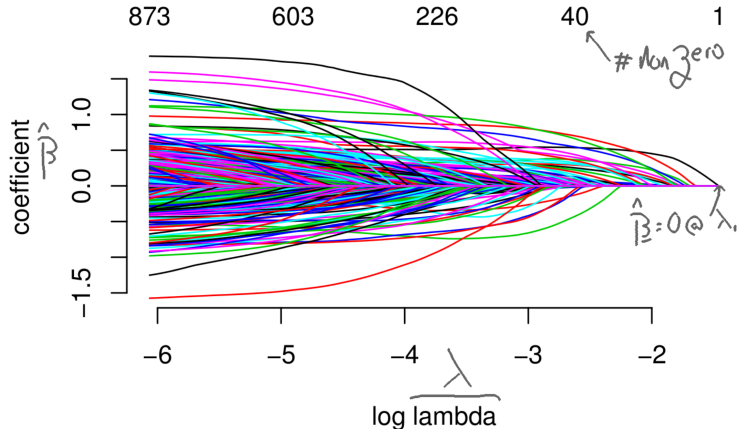
$$\min \sum_{i=1}^n (y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij})^2 + \lambda \sum_{j=1}^p |\beta_j|$$

El LASSO es cualitativamente distinto al Ridge. Al penalizar de manera lineal la inclusion de los parámetros, el costo marginal de incluir una variable a $\beta = 0$ es λ . Esto significa que LASSO **si selecciona variables. Las fija a cero.**

El hecho de que LASSO haga selección de variables lo convierte en el favorito de los métodos de regularización. Es muy interpretable y poderoso. Siempre empiecen por LASSO antes de ir a Ridge.

LASSO

Esto es el el LASSO path. Vean como mientras más sube λ , o castigamos la complejidad del modelo, los coeficientes se van a cero.



Selección de λ

Hay dos maneras de escoger el hiper parámetro λ :

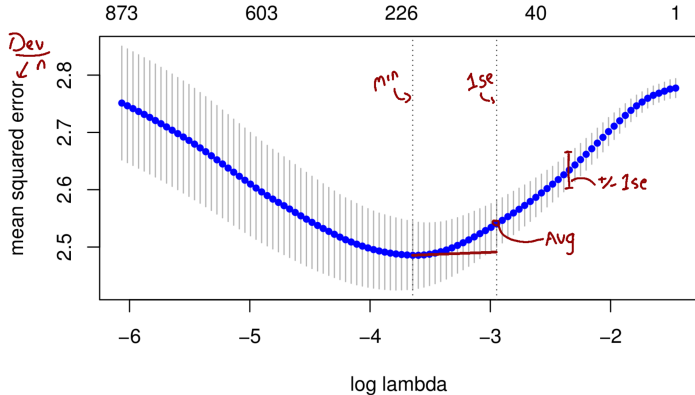
- ▶ Cross-validation
- ▶ LASSO Path con criterios de información

CV LASSO

Pasos:

1. Escoges un vector $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_L)$
2. Divides la base en k-partes de manera aleatoria
3. Para cada $\lambda_l \in \lambda$:
 - ▶ Entrenas el modelo usando toda la base menos un fold k
 - ▶ Predices en la parte k restante
 - ▶ Calculas el error de predicción: $CV_{(k)} = \sum_{i=1}^k error\ rate$
4. Escoges la λ_l que te dio mejor error de predicción CV
5. Finalmente, reestimas el modelo con λ_l^* para toda la base de entrenamiento

Opciones en el CV LASSO



min selecciona el que minimiza el error de predicción. 1se es parecido. Elige un modelo más sencillo con un error de predicción que no difiere en más de una desviación estándar.

Problemas con CV LASSO

- ▶ El cross validation puede ser muy tardado de estimar.
- ▶ Puede ser algo inestable si tenemos muestras no tan grandes. Esto es porque las k-partes no tendrán suficiente información.

LASSO Path: Information Criterion

Information criteria: Son estadísticos que aproximan cuanta información se pierde al correr nuestro modelo. Es como un estimador del error de predicción OOS. Queremos minimizar los criterios de información. Existen varios:

- **Akaike Information Criterion (AIC):** $deviance + 2df$. Donde df es el número de β 's incluidas en el modelo.

El AIC es muy buen estimador de OOS error rate si n/df es grande. Si (Big Data!) $n \simeq df$, el AIC es un pobre estimador y tiene a overfitting.

- **AICc:** El AIC pero corregido. $deviance + 2df \frac{n}{n-df-1}$. Siempre usar este, ya que incluso si $n \simeq df$ es un buen estimador de el error de predicción.
- **Bayesian Information Criterion (BIC):** $deviance + \log(n)df$. Este criterio busca que encontrar al modelo 'real'. En la práctica, es más conservador. Por ende tiene a elegir modelos más simples

LASSO Path: Information Criterion

Usando AICc, podemos correr un LASSO path. Pasos:

1. Escogemos una secuencia de λ : $\lambda_1 < \lambda_2 < \dots < \lambda_{T=100}$
2. Empieza con una $\lambda_T \simeq \infty$ (no incluyes mas que el intercepto)
3. Ve con una λ_{T-1} más pequeña.

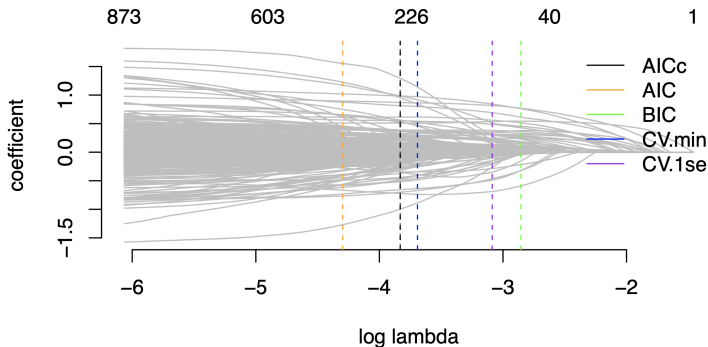
...

4. Termina en una en una lambda muy pequeña (donde incluyes casi todo).
5. Escoge la estimación que arroje el menor AICc

Qué facil!!! En el mundo de machine learning, es muy común esta forma de prueba-error. Uno construye *grids* de los hiper parámetros a tunear que cubran mucho del espacio posible (ninguna variable, todas) y partir de ahí prueba todo. Es parecido a una optimización numérica.

Comparativa entre ICs

El AICc se parece mucho a el seg.min de CV; BIC se parece un poco a seg.1se de CV (ojo! a veces demasiado a la izquierda o underfitting). El AIC tiende al overfitting.



Ejemplo de correr LASSO en R

Las librerías más comunes son `glmnet` y `gamlr`. La diferencia es que `glmnet` puede estimar elastic nets. Para el resto, son equivalentes. A mi me gusta más `gamlr` porque aprendí en ella.

En ambas, los datos deben ser cargados en una `sparse.matrix`. Para esto, se carga la librería `Matrix`.

```
library(gamlr)
library(fastDummies)

# Creamos los vectores y y la matriz de X's
y<-data$churn

# La base debe tener puras numericas
X<-sparse.model.matrix(~.+0, data = products)

# LASSO Path
lasso<-gamlr(x = X, y = Y, family = 'binomial')
```

Ejemplo LASSO en R


```
# LASSO Path
lasso<-gamlr(x = X, y = Y, family = 'binomial')

# Prediciendo
y_pred<-predict(lasso1, newdata = X1, type = 'response')
y_pred<-as.numeric(y_pred)

# Lasso path graph
plot(lasso)

# coeficientes
lasso_summary<-summary(lasso)
```


Ejemplo LASSO en R



gamr (gamr)

R Documentation

Gamma-Lasso regression

Description

Adaptive L1 penalized regression estimation.

Usage

```
gamr(x, y,
     family=c("gaussian", "binomial", "poisson"),
     gamma=0, lambda=10, lambda.start=Inf,
     lambda.min.ratio=0.01, fine=FALSE, standardize=TRUE,
     downweight=FALSE, varweight=TRUE,
     prexx=p>50),
     tol=1e-7, maxit=1e5, verb=FALSE, ...)
```

S3 method for class "gamr"

```
plot(x, against=c("pen", "dev"),
     col=NULL, select=TRUE, df=TRUE, ...)
```

S3 method for class "gamr"

```
coef(object, select=NULL, k=0, corrected=TRUE, ...)
```

S3 method for class "gamr"

```
predict(object, newdata,
         type = c("link", "response"), ...)
```

S3 method for class "gamr"

```
loglik(object, ...)
```

Arguments

x	A dense matrix or sparse matrix of covariates, with <code>ncol(x)</code> variables and <code>nrow(x)==length(y)</code> observations. This should not include the intercept.
y	A vector of response values. There is almost no argument checking, so be careful to match y with the appropriate family.
family	Response model type: either "gaussian", "poisson", or "binomial". Note that for "binomial", y is in [0,1].
gamma	Penalty concavity tuning parameter: see details. Zero (default) yields the lasso, and higher values correspond to a more concave penalty.
nlambda	Number of regularization path segments.
lambda.start	Initial penalty value. Default of <code>Inf</code> implies the infimum lambda that returns all zero coefficients. This is the largest absolute coefficient gradient at the null model.
lambda.min.ratio	The smallest penalty weight (expected L1 <code>coef</code>) as a ratio of the path start value. Our default is always 0.01; note that this differs from <code>glmnet</code> , whose default depends upon the dimension of x.
fine	Free variables: indices of the columns of x which will be unpunished.
standardize	Whether to standardize the coefficients to have standard deviation of one. This is equivalent to multiplying the L1 penalty by each coefficient standard deviation.
downweight	For family="gaussian" only, weights on each observation in the weighted least squares objective. For other response families, <code>downweight</code> are overwritten by FRS weights. Defaults to <code>rep(1,n)</code> .
varweight	Multipiers on the penalty associated with each covariate coefficient. Must be non-negative. These are further multiplied by <code>adj_ij</code> if <code>standardize=TRUE</code> . Defaults to <code>rep(1,p)</code> .
prexx	Only possible for family="gaussian": whether to use pre-calculated weighted variable covariances in gradient calculations. This leads to massive speed-ups for big-n datasets, but can be slow for <code>p<n</code> datasets. See note.
tol	Optimization convergence tolerance relative to the null model deviance for each inner coordinate-descent loop. This is measured against the maximum coordinate change times deviance curvature after full parameter-set update.
maxit	Max iterations for a single segment coordinate descent routine.
verb	Whether to print some output for each path segment.
object	A <code>gamr</code> object.
against	Whether to plot paths against log penalty or deviance.
select	In <code>coef</code> (and <code>predict</code> , which calls <code>coef</code>), the index of path segments for which you want coefficients or prediction (e.g., do <code>select=which.abs(BIC(object))</code> for BIC selection). If null, the segments are selected via our <code>AICc</code> function with k as specified (see also <code>corrected</code>). If <code>select=0</code> all segments are returned. In <code>plot</code> , <code>select</code> is just a flag for whether to add lines marking AICc and BIC selected models.
k	If <code>select=NULL</code> , in <code>coef</code> or <code>predict</code> , the AICc complexity penalty, k defaults to the usual 2.
corrected	A flag that swaps <code>correct</code> (for high dimensional bias) AICc in for the standard AIC. You almost always want <code>corrected=TRUE</code> , unless you want to apply the BIC in which case use <code>b=log(n)</code> , <code>corrected=FALSE</code> .
newdata	New x data for prediction.

 Below "k" is the BIC selection, or "penalized" for model selection based on the BIC statistic (see ...)

Ejemplo LASSO en R



cv.gamlr (gamlr)

R Documentation

Cross Validation for gamlr

Description

Cross validation for gamma lasso penalty selection.

Usage

```
cv.gamlr(x, y, nfold=5, foldid=NULL, verb=FALSE, cl=NULL, ...)
## S3 method for class 'cv.gamlr'
plot(x, select=TRUE, df=TRUE, ...)
## S3 method for class 'cv.gamlr'
coef(object, select=c("lse", "min"), ...)
## S3 method for class 'cv.gamlr'
predict(object, newdata, select=c("lse", "min"), ...)
```

Arguments

x Covariates; see `gamlr`.
y Response; see `gamlr`.
nfold The number of cross validation folds.
foldid An optional length-*n* vector of fold memberships for each observation. If specified, this dictates *nfold*.
verb Whether to print progress through folds.
cl possible parallel library cluster. If this is not-NULL, the CV folds are executed in parallel. This copies the data *nfold* times, so make sure you have the memory space.
... Arguments to `gamlr`.
object A `gamlr` object.
newdata New *x* data for prediction.
select In prediction and coefficient extraction, select which "best" model to return: `select="min"` is that with minimum average OOS deviance, and `select="lse"` is that whose average OOS deviance is no more than 1 standard error away from the minimum. In `plot`, whether to draw these selections.
df Whether to add to the plot degrees of freedom along the top axis.

Details

`Fits` a `gamlr` regression to the full dataset, and then performs *nfold* cross validation to evaluate out-of-sample (OOS) performance for different penalty weights.

`plot.cv.gamlr` can be used to plot the results: it shows mean OOS deviance with 1se error bars.

Value

gamlr The full-data fitted `gamlr` object.
nfold The number of CV folds.
foldid The length-*n* vector of fold memberships.