

Lec7: Random Forest y XGB

Isidoro Garcia Urquieta

2021

Agenda

- ▶ Mini Repaso Trees
- ▶ Bootrapping
- ▶ Bagging
- ▶ Random Forests
- ▶ Boosting
- ▶ Gradient Boosting Machines

Resumen Trees

- ▶ Los árboles de decisión (Regresión y Clasificación) son algoritmos no paramétricos
- ▶ Son fáciles de interpretar!
- ▶ Detectan no linealidades de manera automática
- ▶ Se estiman de manera **greedy**, donde se buscan splits que maximicen la varianza entre nodos (inter-varianza) y se minimice la varianza dentro de cada nodo (intra-varianza)
- ▶ Hay dos maneras de parar la estimación de los arboles: observaciones mínimas en los nodos terminales o disminución en el deviance mínima.
- ▶ El greediness es miope. Esto es, un split malo puede seguirle un gran split.
- ▶ Es difícil evitar el overfitting en los árboles.

Cost complexity pruning

Cost complexity pruning (Weakest link pruning) es la manera más común de reducir el número de modelos candidatos y aplicar k-fold CV.

El proceso funciona así:

1. Estima árbol de manera greedy hasta llegar a `min obs`.
2. Aplica un parámetro de complejidad α que castigue la complejidad del árbol. Esto te dará una secuencia de subárboles como función de α .
3. Aplica k-fold CV para escoger α

Cost complexity pruning

Veamos la formula de cost complexity pruning:

$$\sum_{j=1}^J \sum_{R_j} (y_i - \hat{y}_i)^2 + \alpha |T|$$

Donde T es el número de nodos terminales. Se ve familiar?

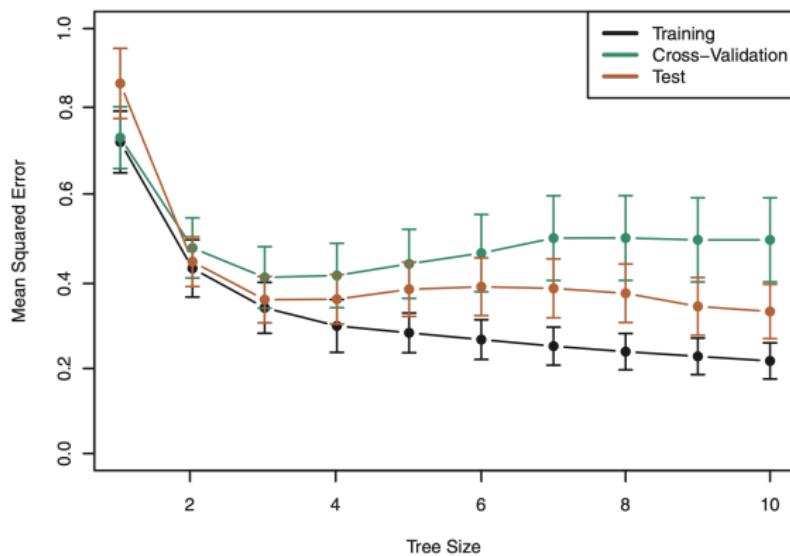
Resulta que si un α muy grande, tendremos un árbol con $T_0 \subset T$.

Mientras α tiende a cero $T_0 \rightarrow T$.

α nos ayuda a tener un set de candidatos de modelo estimable vía K-fold CV!!

Grafica de desempeño de CV tree

Noten como el CV es un buen estimador del desempeño fuera de la muestra (3 nodos terminales). Adicionalmente, noten como en el training set, el árbol siempre parece que va a estimar mejor mientras más grande haga los árboles.



Resumen Tree

- ▶ Para ayudar al overfitting, podemos hacer Tree pruning con Cross Validation
- ▶ Esto hará que estimemos greedy y luego tengamos un set de árboles candidatos dependiendo de cuánto castigamos la complejidad de los mismos.
- ▶ Los prune.trees son más poderosos (predicción) que los árboles normales. Por ende, preferibles.
- ▶ Aún así, los árboles son estimadores de alta varianza. Esto es, varían mucho dependiendo de la muestra en la que nos encontramos (no paramétrico). Por ende, necesitamos alguna manera de estabilizar las predicciones de los árboles para lograr mejor poder predictivo OOS.

Boostrapping

Llegamos al segundo método de **remuestreo** (El primero fue CV)!

El boostrapping es muy útil para conseguir errores estandares e intervalos de confianza cuando la distribución de algún estimador es desconocida, tenemos pocas observaciones o es muy difícil de estimar.

Por ejemplo, en el caso de la econometría, podemos inferir la distribución de nuestro estimador $\hat{\beta}$ al remuestrear la base **con reemplazo** y estimar β con cada muestra.

Lo bonito del Boostrapping es que tiene otras aplicaciones además de la generación de errores standár. Se puede aplicar a mejorar el poder predictivo de nuestros modelos estadísticos.

Boostrapping

Pasos:

Dada una base de datos $\{z_i\}_{i=1}^n$, para $b \in \{1, 2, 3, \dots, B\}$ muestras:

1. Tomas una muestra **con reemplazo** $\{z_i^b\}_{i=1}^n$
2. Estimas β_b usando la base de (1)

Luego, $\{\beta_b\}_{b=1}^B$ es una aproximación de la distribución de $\hat{\beta}$

El vector de tamaño B $\{\beta_b\}_{b=1}^B$ puede ser usado como la distribución teorica que aprendimos en la clase 2 (inferencia!) que usaba el Teorema del Límite Central.

$$\beta \in \hat{\beta} \pm 2sd(\hat{\beta}_b)$$

$$se(\hat{\beta}) \simeq sd(\beta_b) = \sqrt{\frac{1}{B} \sum_b (\hat{\beta}_b - \hat{\beta})^2}$$

Donde $\hat{\beta}$ es el estimador original usando toda la base de datos.

Bagging

Los árboles de decisión sufren alta varianza. Esto es, cuando dividimos la muestra en entrenamiento y validación y entrenamos el árbol (c/ CV pruning), los resultados pueden ser muy distintos en cada base.

En el caso del LASSO, esto no era así. Porqué?

- ▶ Por que teníamos **estimadores** paramétricos que se estabilizan con cierta facilidad ($n > p$).

El **Boosting Aggregation (Bagging!)** es un procedimiento muy utilizado para estabilizar la varianza en aprendizaje estadístico.

Bagging

El Bagging utiliza el poder de las medias para crear estimadores estables.

Recuerden que si tenemos un set de n observaciones independientes Z_1, Z_2, \dots, Z_n y *tenemos un procedimiento insesgado*:

Promediamos las Z_i : \bar{Z}

Sabemos que:

$$\bar{Z}_n \xrightarrow{D} N(\mu_Z, \sqrt{\sigma^2/n})$$

En otras palabras, la varianza original de los datos σ^2 se reduce cuando sacamos promedios a σ^2/n !

Bagging en Árboles de Decisión

Pasos para aplicar Bagging en árboles de decisión:

1. Construye B muestras con reemplazo
2. Construyes un árbol en cada muestra B : $\hat{f}^b(X)$ $b \in \{1, \dots, B\}$
3. Construye el Bag estimate:
 - ▶ Para árboles de regresión: $\hat{f}_{bag}(X) = \frac{1}{B} \sum_{b=1}^B \hat{f}^b(X)$
 - ▶ Para árboles de clasificación: Majority vote. La clase más vota es a predicción.

Queda pendiente algo: Cuántas submuestras B tomar? Con B alto caemos en overfitting?

Bagging

Cuantas B tomar?

- ▶ En la práctica, necesitamos B suficientemente grandes para estabilizar la varianza. Esto
- ▶ Otro punto importante es que un número grande B no induce overfitting, al contrario.
- ▶ Con: Estimar los árboles con demasiadas submuestras puede ser computacionalmente complejo. No obstante es alcanzable en una buena computadora (Parallel Computing!)

Out-of-Bag

Resulta que, como en Cross-Validation, hay manera de estimar el error OOS desde la muestra de entrenamiento.

Recuerden que en cada muestra B , dejamos algunas observaciones sin tomar. Estas observaciones pueden servir como estimador del error. Su nombre es out of bag.

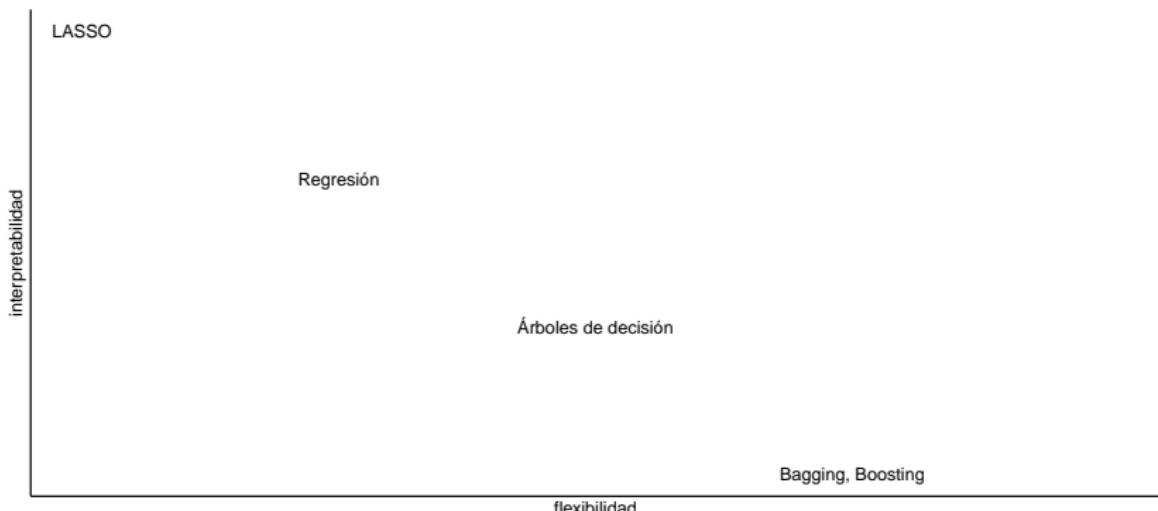
Los resultados muestran que en la práctica, los bagging trees superan en poder de predicción a los trees y pruned trees.

Terminamos? que cosas se les ocurren que pueden seguir pasando con los Bagging Estimators?

Interpretabilidad

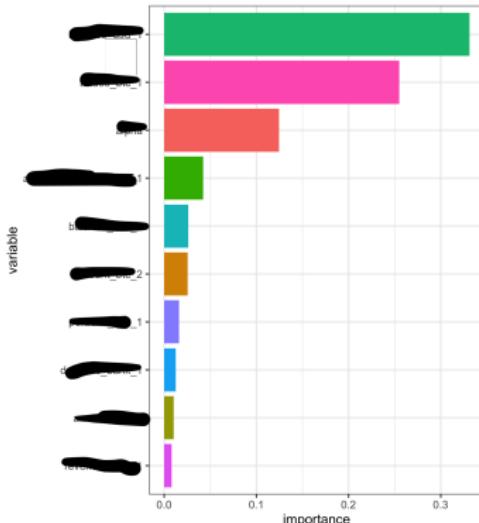
El bagging nos permite estabilizar las predicciones de los árboles y ganar poder predictivo.

El costo asociado es la **interpretabilidad**



Variable Importance

Una manera de interpretar los resultados es contar para cada variable cuántos splits de hicieron. Un valor grande es un indicativo de que esa variable es muy útil para predecir y .



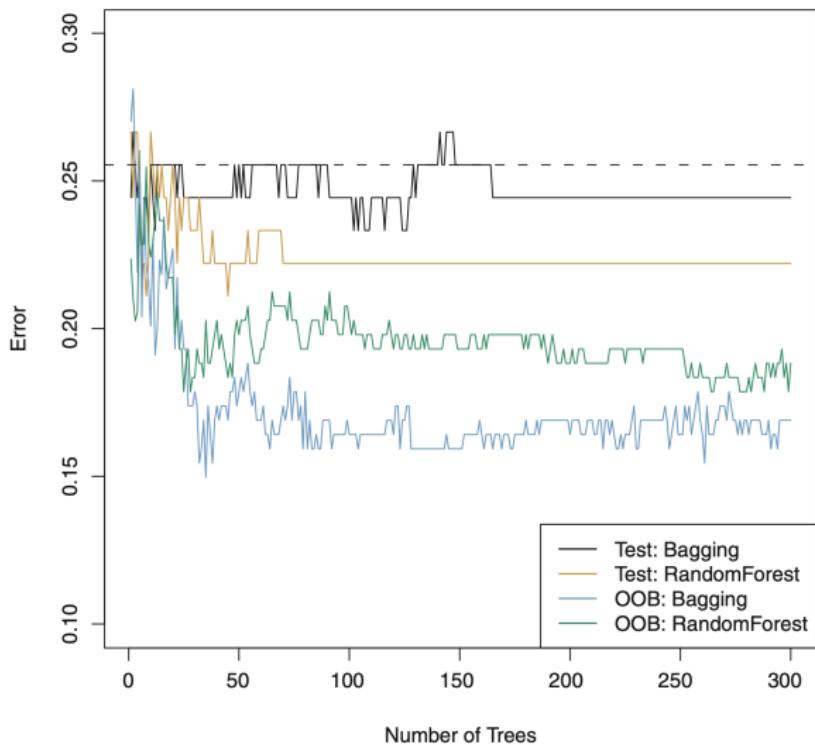
Random Forests

Un problema de los Bagging Trees es que incluso con el remuestreo, los árboles tienen mucha probabilidad de ser muy parecidos entre ellos. Muy **correlacionados**.

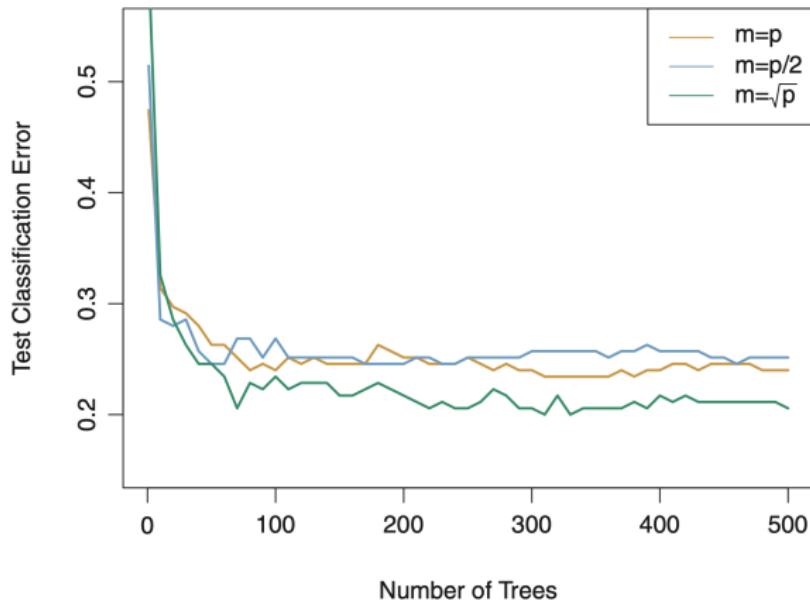
Un Random Forest resuelve esto así (para una base (n, p)):

1. Construye B muestras con reemplazo
2. **Toma una muestra de las columnas** $m = \sqrt{p}$
3. Construyes un árbol en cada muestra B : $\hat{f}^b(X) \ b \in \{1, \dots, B\}$
4. Construye el Bag estimate:
 - ▶ Para árboles de regresión: $\hat{f}_{bag}(X) = \frac{1}{B} \sum_{b=1}^B \hat{f}^b(X)$
 - ▶ Para árboles de clasificación: Majority vote. La clase más vota es a predicción.

Random Forest vs Bagging Trees



Cuántas columnas m



Random Forests en R

library(ranger) y library(randomForest) son las librerías más comunes en R. La primera es lo mismo que la segunda pero mucho más rápida en estimar.

```
detectCores()
cl<-makeCluster(12)
cl

# Estimation
a<-Sys.time()
rf<-ranger(y~., data = x, classification = T, num.trees = 750)

Sys.time() -a

save(rf, file = 'Modelos/rf_1000.Rdata')
stopCluster(cl)
```

ranger prediction

```
# Out of Bag  
oob<-rf$predictions  
  
# prediction vectors  
pred_val<-predict(rf, data = validation)$predictions
```

Ranger



ranger (<ranger>)

R Documentation

Ranger

Description

Ranger is a fast implementation of random forests (Breiman 2001) or recursive partitioning, particularly suited for high dimensional data. Classification, regression, and survival forests are supported. Classification and regression forests are implemented as in the original Random Forest (Breiman 2001), survival forests as in Random Survival Forests (Ishwaran et al. 2008). Includes implementations of extremely randomized trees (Geurts et al. 2006) and quantile regression forests (Meinshausen 2006).

Usage

```
ranger::ranger  
formula = NULL,  
data = NULL,  
num.trees = 500,  
mtry = 3,  
importance = "none",  
write.forest = TRUE,  
probability = FALSE,  
min.node.size = NULL,  
max.depth = NULL,  
replace = TRUE,  
sample.fraction = ifelse(replace, 1, 0.632),  
case.weights = NULL,  
class.weights = NULL,  
weights = NULL,  
num.random.splits = 1,  
alpha = 0.5,  
min.node.size = 0.1,  
split.select.weights = NULL,  
always.split.variables = NULL,  
respect.unordered.factors = NULL,  
scale.permutation.importance = FALSE,  
local.improvement = FALSE,  
quantile.permutation.factor = 1,  
regularization.used.edepth = FALSE,  
keep.inbag = FALSE,  
inbag = NULL,  
allow.miss = FALSE,  
quantreg = FALSE,  
oob.error = TRUE,  
num.threads = NULL,  
save.memory = FALSE,  
prob.above = TRUE,  
seed = NULL,  
dependent.variable.name = NULL,  
dependent.variable.name = NULL,  
classification = NULL,  
x = NULL,  
y = NULL  
)
```

Ranger



Arguments

<code>formula</code>	Object of class <code>formula</code> or <code>character</code> describing the model to fit. Interaction terms supported only for numerical variables.
<code>data</code>	Training data of class <code>data.frame</code> , <code>matrix</code> , <code>dgCMatrix</code> (<code>Matrix</code>) or <code>gwaa.data</code> (<code>GwABEL</code>).
<code>num.trees</code>	Number of trees.
<code>mtry</code>	Number of variables to possibly split at in each node. Default is the (rounded down) square root of the number variables. Alternatively, a single argument function returning an integer, given the number of independent variables.
<code>importance</code>	Variable importance mode, one of 'none', 'impurity', 'impurity_corrected', 'permutation'. The 'impurity' measure is the Gini index for classification, the variance of the responses for regression and the sum of test statistics (see <code>splitrule</code>) for survival.
<code>write.forest</code>	Save <code>ranger.forest</code> object, required for prediction. Set to <code>FALSE</code> to reduce memory usage if no prediction intended.
<code>probability</code>	Grow a probability forest as in Malfay et al. (2012).
<code>min.node.size</code>	Minimal node size. Default 1 for classification, 5 for regression, 3 for survival, and 10 for probability.
<code>max.depth</code>	Maximal tree depth. A value of <code>NULL</code> or 0 (the default) corresponds to unlimited depth, 1 to tree stumps (1 split per tree).
<code>replace</code>	Sample with replacement.
<code>sample.fraction</code>	Fraction of observations to sample. Default is 1 for sampling with replacement and 0.632 for sampling without replacement. For classification, this can be a vector of class-specific values.
<code>case.weights</code>	Weights for sampling of training observations. Observations with larger weights will be selected with higher probability in the bootstrap (or subsampled) samples for the trees.
<code>class.weights</code>	Weights for the outcome classes (in order of the factor levels) in the splitting rule (cost sensitive learning). Classification and probability prediction only. For classification the <code>weights</code> are also applied in the majority vote in terminal nodes.
<code>splitrule</code>	Splitting rule. For classification and probability estimation "gini", "extratrees" or "hollinger" with default "gini". For regression "variance", "extratrees", "maxstat" or "beta" with default "variance". For survival "logrank", "C" or "maxstat" with default "logrank".
<code>num.random.splits</code>	For "extratrees" splitrule: Number of random splits to consider for each candidate splitting variable.
<code>alpha</code>	For "maxstat" splitrule: Significance threshold to allow splitting.
<code>minprop</code>	For "maxstat" splitrule: Lower quantile of covariate distribution to be considered for splitting.
<code>split.select.weights</code>	Numeric vector with weights between 0 and 1, representing the probability to select variables for splitting. Alternatively, a list of size <code>num.trees</code> , containing split select weight vectors for each tree can be used.
<code>always.split.variables</code>	Character vector with variable names to be always selected in addition to the <code>mtry</code> variables tried for splitting.
<code>respect.unordered.factors</code>	Handling of unordered factor covariates. One of 'ignore', 'order' and 'partition'. For the "extratrees" splitrule the default is 'partition' for all other splitrules 'ignore'. Alternatively TRUE (=order) or FALSE (=ignore) can be used. See below for details.
<code>scale.permutation.importance</code>	Scale permutation importance by standard error as in (Breiman 2001). Only applicable if <code>importance</code> is set to 'permutation'.
<code>local.importance</code>	Calculate and return local importance values as in (Breiman 2001). Only applicable if <code>importance</code> is set to 'permutation'.
<code>regularization.factor</code>	Regularization factor (gain penalization), either a vector of length p or one value for all variables.
<code>regularization.useddepth</code>	Consider the depth in regularization.
<code>keep.inbag</code>	Show how often observations are in-bag in each tree.
<code>inbag</code>	Manually set observations per tree. List of size <code>num.trees</code> , containing inbag counts for each observation. Can be used for stratified sampling.
<code>holdout</code>	Hold-out mode. Hold-out all samples with case weight 0 and use these for variable importance and prediction error.
<code>quantreg</code>	Prepare quantile prediction as in quantile regression forests (Menshawen 2006). Regression only. Set keep.inbag = TRUE to prepare out-of-bag quantile prediction.
<code>oob.error</code>	Compute OOB prediction error. Set to <code>FALSE</code> to save computation time, e.g. for large survival forests.
<code>num.threads</code>	Number of threads. Default is number of CPUs available.
<code>save.memory</code>	Use memory saving (but slower) splitting mode. No effect for survival and GWAS data. Warning: This option slows down the tree growing, use only if you encounter memory problems.
<code>verbose</code>	Show computation status and estimated runtime.
<code>seed</code>	Random seed. Default is <code>NULL</code> , which generates the seed from R. Set to 0 to ignore the R seed.
<code>dependent.variable.name</code>	Name of dependent variable, needed if no formula given. For survival forests this is the time variable.
<code>status.variable.name</code>	Name of status variable, only applicable to survival data and needed if no formula given. Use 1 for event and 0 for censoring.
<code>classification</code>	Set to <code>TRUE</code> to grow a classification forest. Only needed if the data is a matrix or the response numeric.
<code>x</code>	Predictor data (independent variables), alternative interface to data with formula or <code>dependent.variable.name</code> .
<code>y</code>	Response vector (dependent variable), alternative interface to data with formula or <code>dependent.variable.name</code> . For survival use a <code>Surv()</code> object or a matrix with time and status.
<code>Details</code>	

Ranger



Value

Object of class `ranger` with elements

<code>forest</code>	Saved forest (If <code>write.forest</code> set to TRUE). Note that the variable IDs in the <code>split.varIDs</code> object do not necessarily represent the column number in R.
<code>predictions</code>	Predicted classes/values, based on out of bag samples (classification and regression only).
<code>variable.importance</code>	Variable importance for each independent variable.
<code>variable.importance.local</code>	Variable importance for each independent variable and each sample, if <code>local.importance</code> is set to TRUE and <code>importance</code> is set to 'permutation'.
<code>prediction.error</code>	Overall out of bag prediction error. For classification this is the fraction of misclassified samples, for probability estimation the Brier score, for regression the mean squared error and for survival one minus Harrell's C-index.
<code>r.squared</code>	R squared. Also called explained variance or coefficient of determination (regression only). Computed on out of bag data.
<code>confusion.matrix</code>	Contingency table for classes and predictions based on out of bag samples (classification only).
<code>unique.death.times</code>	Unique death times (survival only).
<code>chf</code>	Estimated cumulative hazard function for each sample (survival only).
<code>survival</code>	Estimated survival function for each sample (survival only).
<code>call</code>	Function call.
<code>num.trees</code>	Number of trees.
<code>num.independent.variables</code>	Number of independent variables.
<code>ntry</code>	Value of mtry used.
<code>min.node.size</code>	Value of minimal node size used.
<code>tree.type</code>	Type of forest/tree, classification, regression or survival.
<code>importance.mode</code>	Importance mode used.
<code>num.samples</code>	Number of samples.
<code>inbag.counts</code>	Number of times the observations are in-bag in the trees.

Ejemplo: California Housing data

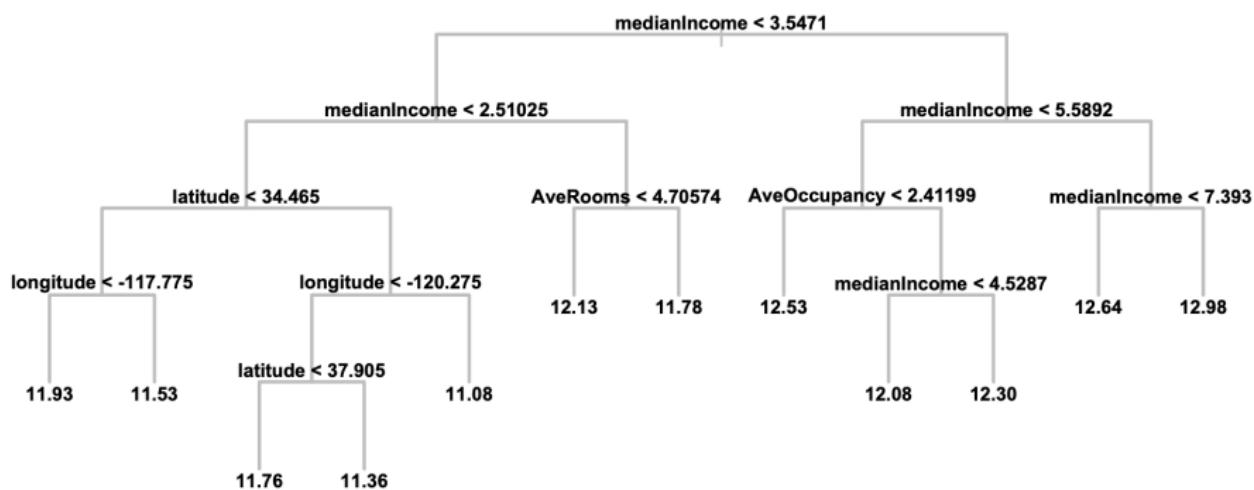
Tenemos una base de precios de casas en California que contiene, por código censal:

- ▶ Precios medianos de las casas
- ▶ Población e ingreso
- ▶ Características del hogar

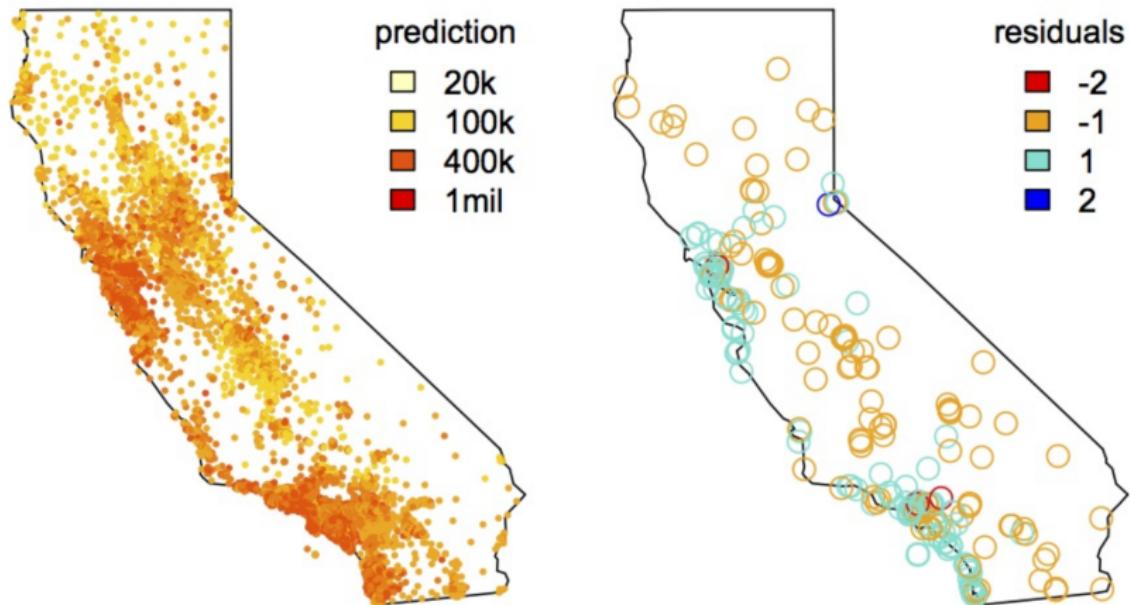
Queremos predecir el precio para cada código censal
1) LASOO 2) Prune Tree 3) Random Forest

```
carf<-ranger(logMedVal~., data= CAhousing,  
               write.forest = T, num.trees = 200,  
               min.node.size = 25,  
               importance = 'impurity')
```

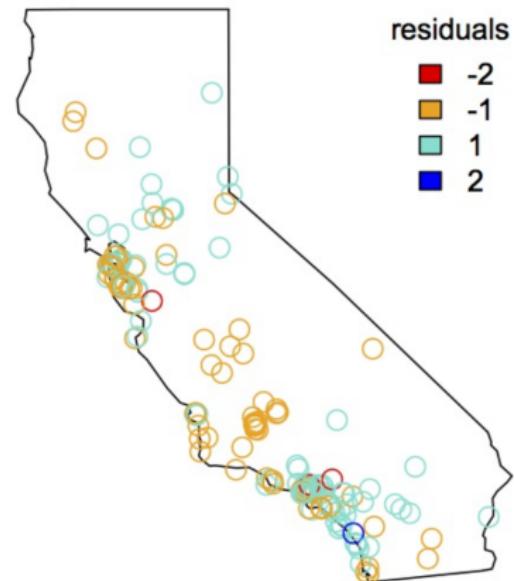
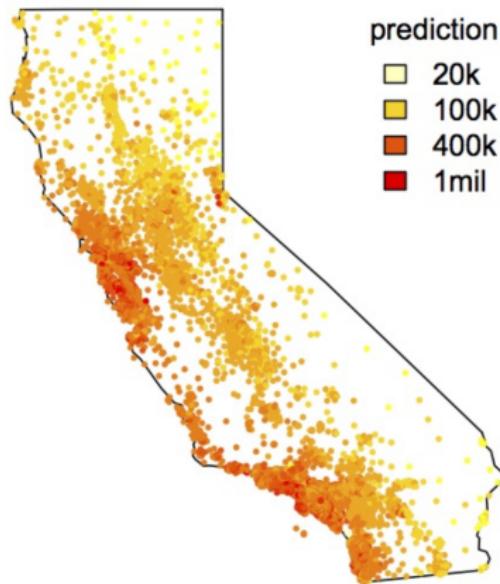
Ejemplo: California Housing data



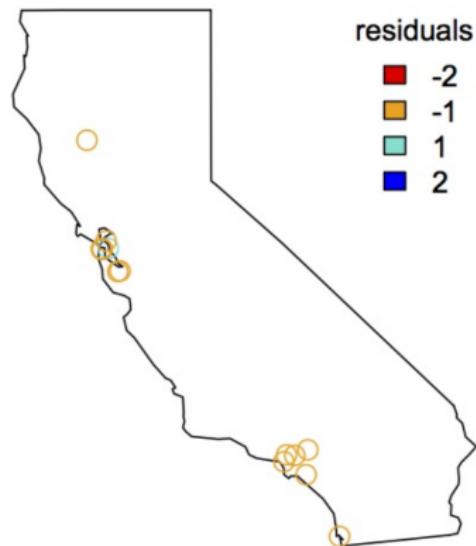
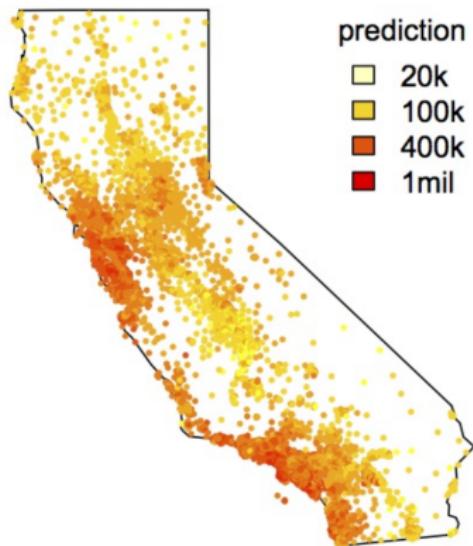
Fit Arbol



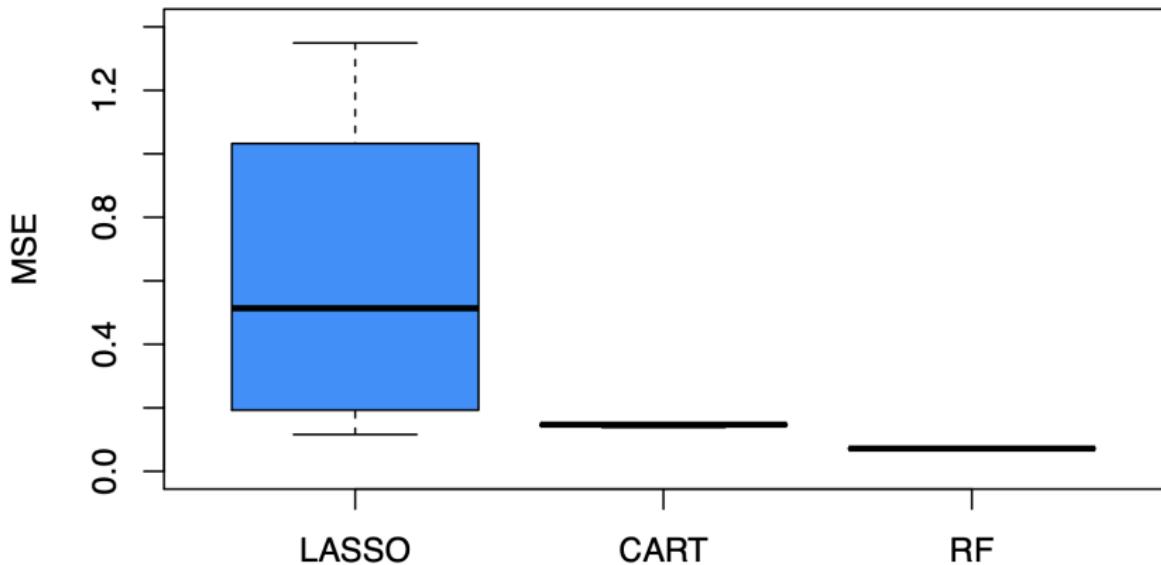
Fit LASSO



Fit Random Forest



Comparativa



Boosting

Ahora discutamos Boosting. Como el bagging y el CV, se puede utilizar en muchos contextos.

En Bagging creamos varias bases de datos, estimamos árboles en cada uno y promediamos. En Boosting, los árboles crecen **secuencialmente**.

- ▶ Esto es, cada árbol usa información del árbol anterior.

Pasos. Para $b \in \{1, \dots, B\}$:

1. Estimas el primer árbol $\hat{f}^b(X)$ sobre los residuales r con d splits ($d + 1$ nodos terminales)
2. Obtienes una nueva función de predicción: $\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(X)$
3. Actualiza los residuales: $r_i \leftarrow r_i + \lambda \hat{f}^b(x)$

El modelo final: $\hat{f}(x) = \sum_{b=1}^B \lambda \hat{f}^b(x)$

Gradient Boosting Machines

Cuál es la intuición del Boosting?

- ▶ En lugar de tener una combinación de arboles que aprenden mucho tengamos árboles sencillos
- ▶ Cada árbol va a estar aprendiendo lo siguiente más importante de $f(x)$. Se toma el árbol que disminuye más el RSS (gradient)
- ▶ Esto hace que las GBM aprendan **lento**.

Parámetros a tunear:

- ▶ Número de árboles B : A diferencia de bagging y RF, si B es muy grande, caeremos en overfitting. Usamos cross-validation para obtener B
- ▶ λ shrinkage parameter: Este controla (regulariza) al ritmo al que el algoritmo lee. Baja el peso de cada nuevo árbol. Se interpreta como “escepticismo”. Típicamente tiene valores de 0.01, 0.001.
- ▶ d Número de splits en cada árbol. Típicamente $d = 1$

Extreme Gradient Boosting Machines (XGBoosting)

Una generalización de los Gradient Boosting Machines. Igual busca disminuir el gradiente de RSS mediante Boosting. La diferencia es que los XGBoosting tienen muchos más parámetros a tunear (aquí algunos):

- ▶ B [nrounds]: Número de árboles
- ▶ `early_stopping_rounds`: Resuelve la miopía y la gobierna.
Cuántas rondas pueden pasar sin que disminuya el RSS para parar.
Igual los hace más ligeros que un Random Forest.
- ▶ d `max_depth`: El tamaño máximo permitido por árbol.
- ▶ `min_child_weight`: Muy similar a mínimas observaciones en los nodos terminales del árbol.
- ▶ `gamma`: La ganancia mínima requerida dentro de cada árbol. Controla la profundidad y cantidad de árboles.
- ▶ λ `eta`: El tasa de aprendizaje $[0, 1]$.
- ▶ `subsample`: La muestra que toma para estimar el árbol (m en RF)

Extreme Gradient Boosting Machines (XGBoosting)

- ▶ Como ven, los XGB combinan muchos conceptos que vimos en trees, LASSO e incluso Random Forest.
- ▶ Esto los convierte en los algoritmos favoritos (por poderosos) de muchos científicos de datos.
- ▶ Tienden a ser más ligeros que los Random Forests.
- ▶ la librería para estimarlos en R es `library(xgboost)`
- ▶ Típicamente se tunea estos parámetros haciendo un grid de todos ellos (que cubra mucho del espacio) y despues se observa cuál combinación fue la ganadora.