Lec13: Causal Machine Learning II

Isidoro Garcia Urquieta

2021

Agenda

- Causal Trees
- Double Debiased ML for HTE
- Causal forests
- Generalized Random Forests

Como combinar Inferencia Causal con Machine Learning?

De manera general, la inferencia causal trata de **inferir** sobre **derivadas** o cambios en la métrica Y_i en lugar de la Y_i misma (el nivel)

$$\tau_i = \frac{\partial Y_i}{\partial T_i}$$

Por otro lado, los modelos de Machine Learning que vimos buscan **predecir** Y_i fuera de la muestra:

$$\hat{Y}_i = f(\hat{X}_i) + \epsilon_i$$

Son problemas fundamentalmente distintos!

Un grupo de autores: (Christian Hansen, Susan Athey, Guido Imbens, Stephan Wager, Belloni, Victor Chernozshukov, Matt Taddy, Esther Duflo, Robert Tibshirani) se pusieron a pensar cómo aprovechar ML y aplicarlo a **predecir e inferir impactos causales (derivadas)**

Conclusiones clase anterior

En esta clase vimos los primeros algoritmos para hacer inferencia causal en Big Data apalancándonos de algoritmos de ML:

- El sesgo por variables omitidas es $\beta * \delta$ en $\hat{\tau} = \tau + \beta_1 \delta_1$
- Los propensity scores son **cruciales** para alcanzar balances en altas dimensiones.
- Estos se pueden 1) Estimar directamente via ML o 2) Resolver por los pesos que generan mayor balance
- El partialling out (conditional unconfoundness) está detrás de todos estos algoritmos
- Finalmente, el cross-fitting ó honesty (depende del autor) es un ejercicio importantísimo para asegurar estimadores insesgados.

La siguiente clase veremos como estos algoritmos (DDML) y otros nuevos (Causal Forests) extienden esta literatura para encontrar no sólo τ sino τ_i

Double Debiased Machine Learning

Chernoszhukov, Chetverikov, Duflo y Hansen (2018) propusieron un método parecido al de Belloni (2014), pero con dos mejoras sustanciales:

1. Se puede usar cualquier método de Machine Learning en las dos etapas
2. Sacan un método de estimación **Cross-Fitting** que permite insesgar los estimadores.

El problema se motiva así:

$$y_i = \tau T_i + m(X_i) + \epsilon_i$$

$$T_i = g(X_i) + u_i$$

Donde $m(X_i)$ y $g(X_i)$ son **nuisance functions** desconocidas de todas las variables. Estas pueden tener cualquier forma (lineal y no lineal) y ser estimadas con cualquier método de ML.

Double Debiased Machine Learning II

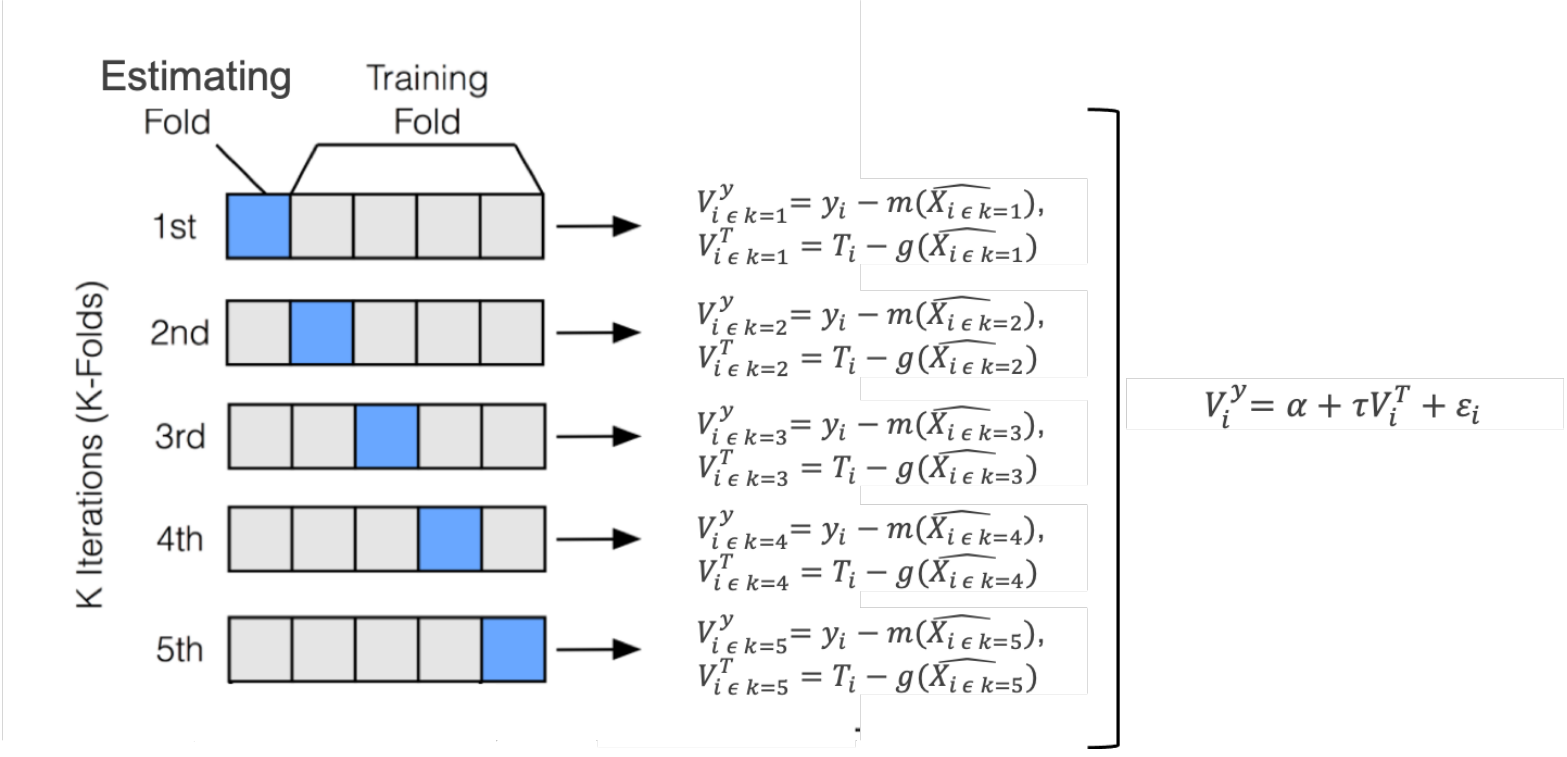
Pasos:

1. Divide tu muestra en K partes de manera aleatoria
2. Estima $g(\hat{X_{i \in K^c}})$ en K-1 partes usando LASSO, Random Forest, XGB u otro método.
3. Predice $g(\hat{X_{i \in K}})$ y $V^y_{i \in K} = y_{i \in K} - g(\hat{X_{i \in K}})$
4. Estima $m(\hat{X_{i \in K^c}})$ en K-1 partes usando LASSO, Random Forest, XGB u otro método.
5. Predice $m(\hat{X_{i \in K}})$ y $V^T_{i \in K} = T_{i \in K} - m(\hat{X_{i \in K}})$
6. Obtén τ al correr la regresión de $V^y_i = \alpha + V^T_i + \psi_i$

- Esto es DDML2. En DDML1 estimas $\tau = \frac{1}{k} \Sigma_k \tau_k$

Double Debiased Machine Learning III

Noten la parte del **cross-fitting**. Entrenas en todas vs una parte las nuisance functions y estimas (creas residuales) en la otra.



Cross-fitting

Los autores muestran que el cross-fitting es **crucial** para obtener estimadores insesgados/causales. Esto también se puede denominar **honesty**.

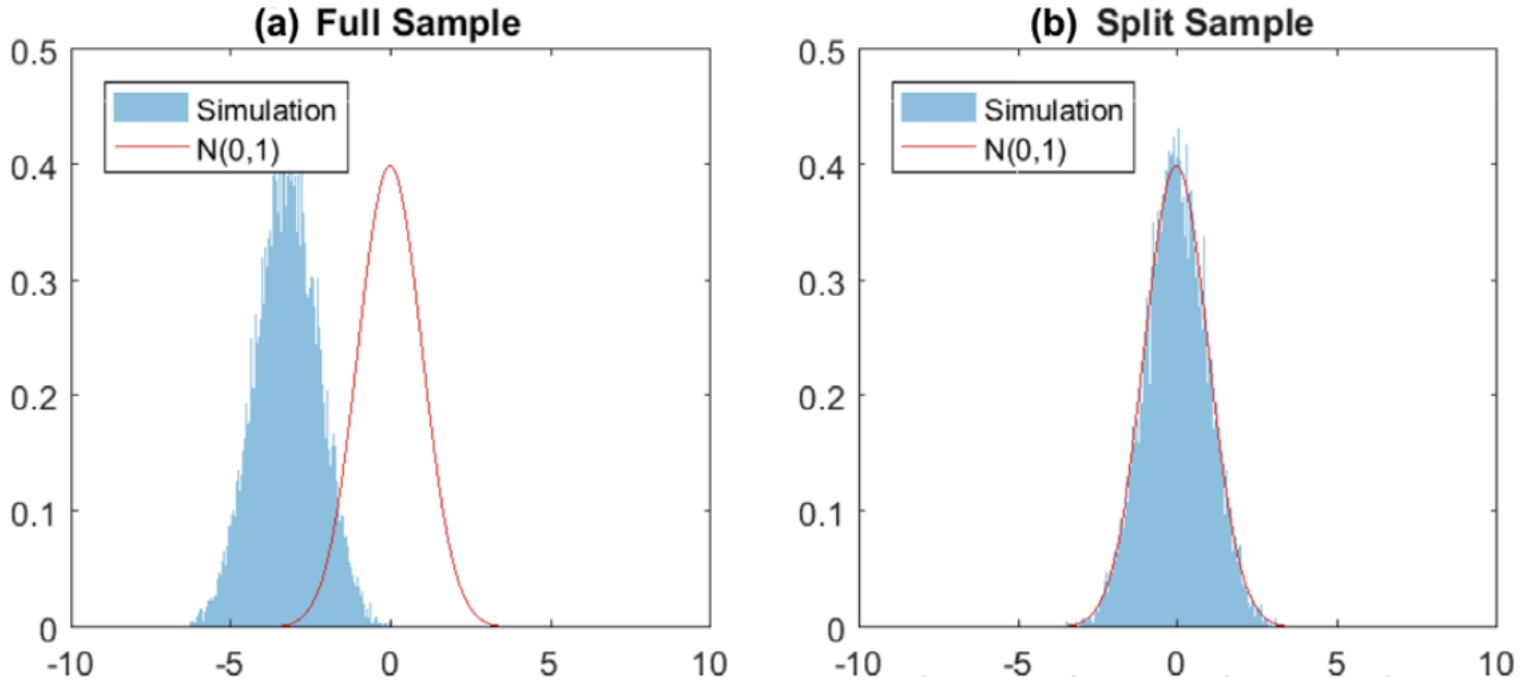


Figure 2. Comparison of full-sample and cross-fitting procedures. [Colour figure can be viewed at wileyonlinelibrary.com]

Double Debiased Machine Learning IV

Finalmente, noten como el algoritmo:

- Aprovecha ML para crear funciones que detectan todas las señales. Esto reduce los errores a un mínimo, pues nos da podemos estimar funciones muy complejas
- Usa el concepto de **partialling out**. $m(X_i)$ y $g(X_i)$ contienen la varianza de y_i y T_i explicada por X.
- Al crear los residuales V_i^y y V_i^T, estamos estimando la relacion entre toda la varianza no explicada por X_i de y_i vs T_i. Ergo, causal!
- Finalmente, la estimación es doble robusta porque, al estimar ambas nuisance functions, no se necesita perfección en ninguna.

Double Debiased Machine Learning for HTE

Como podemos usar DDML para calcular efectos heterogeneos?

Pasos:

1. Divide tu muestra en K partes de manera aleatoria
2. Estima $g(\hat{X_{i \in K^c}})$ en K-1 partes usando LASSO, Random Forest, XGB u otro método.
3. Predice $g(\hat{X_{i \in K}})$ y $V^y_{i \in K} = y_{i \in K} - g(\hat{X_{i \in K}})$
4. Estima $m(\hat{X_{i \in K^c}})$ en K-1 partes usando LASSO, Random Forest, XGB u otro método.
5. Predice $m(\hat{X_{i \in K}})$ y $V^T_{i \in K} = T_{i \in K} - m(\hat{X_{i \in K}})$
6. Corre un LASSO de $V^y_i = \alpha + \tau V^T_i + X\beta + W\delta$. Esto es un LASSO con cada X interactuada con T, dejando que el LASSO escoja las interacciones relevantes.

Donde $W = \{X_1 * T, X_2 * T, ..., X_p * T\}$

Double Debiased Machine Learning for HTE II

Intuitivamente, el DDML crea 2 scores (uno para y y otro para T) que sean **as good as random** y con ellos:

- Estima el ATE
- Usa LASSO para estimar HTE

Algunas notas importantes:

Como uso DDML en bases que no sean en la que entrené?

- Tienes que construir V_j^T, V_j^y primero.
- Para ello, debes usar $g_k(\hat{X}_i)$ y $m_k(\hat{X}_i)$. Esto es genera k $\hat{y}_k$, $\hat{T}_k$.
- Promedia los k scores para obtener $\hat{y}$, $\hat{T}$
- Usa $\hat{y}$, $\hat{T}$ para construir V_j^T, V_j^y en la base de validación.
- Predice τ_i usando la información de la base

Causal Trees

Athey y Wager (2016) crearon un algoritmo llamado Causal Trees. Estos crecen muy parecido a los árboles de decisión, con dos cambios importantes:

1. **Honesty**:

- Tenemos una base de tamaño N. La dividimos en $N_{training}, N_{validation}$
- Despues, dividimos $N_{training}$ en en $N_{splitting}$ y en $N_{estimation}$
- La parte de **splitting** se usa para crecer el árbol causal. La parte **estimating** se usa para estimar (valga la redundancia) el impacto. Se ve familiar?
- El honesty ayuda a que la parte que busca los subgrupos con mayor impacto de tratamiento no sea usada para estimar. De alguna manera valida el estimador (muy parecido al Cross-fitting)

Causal Trees II

Cómo se entrena en la parte **splitting**:

1. Estimas un árbol de forma **greedy**. El **stop rule** se detiene cuando hay un mínimo de observaciones del control **y** del tratamiento.
2. Cuando el árbol termina, estimas el impacto de tratamiento en cada lift como:

$$\hat{\tau}(X_{i \in leaf}) = \bar{Y}_{T=1} - \bar{Y}_{T=0}$$

La intuición es muy simple:

- El Error Cuadrático Medio (ECM) de $ECM(\tau) = sesgo(\tau)^2 + Var(\tau)$
- El splitting set se enfoca únicamente en detectar **en que subpoblaciones** cambia más el impacto τ. Es decir se encarga de $Var(\tau)$
- El estimating set se enfoca en estimar τ de manera insesgada.

Causal Trees III

El segundo cambio **crucial** es el cambio de regla en el splitting set.

- En un tree normal el splitting busca disminuir el $ECM(y)$. Esto es, $ECM'_y = ECM_{y,split1} + ECM_{y,split2} < ECM^0_y$
- El Causal tree debe enfocar en diminuir el $ECM(\tau)$. El problema es que no es observable :(. Como le hacemos? Los autores consideran 3 posibilidades:

1. Transformed Outcome Tree ($Y_i^* = Y_i\left(\frac{T_i - e(X_i)}{e(X_i)*(1-e(X_i))}\right) = \tau_i$): Usar el transformed outcome para despues usar out of the box decision trees.
2. Squared T-Statistic Tree ($T^2 = \frac{(\bar{Y}_L - \bar{Y}_R)^2}{S^2/N_L + S^2/N_R}$)
3. Variance of Treatment Effects τ^2: Haces split donde la intravarianza se minimice y la intervarianza se maximice
4. Propensity Trees: Construyes un propensity tree; calculas τ en cada nodo terminal. (Balancing Score Rosenbaum!)

Causal Trees IV

En su primer paper (Athey, Wager (2016)), los autores muestran que 1) El honesty es crucial para eliminar el sesgo, y 2) Que los trees que crecen usand τ^2 como regla funcionan mejor

Esto convierte a los Causal Trees en el primer algoritmo que **por diseño** busca efectos heterogeneos de tratamiento. Esto lo hace muy poderoso, ya que (como todo árbol) puede encontrar funciones $\tau(X_i)$ que sean altamente no-lineales.

Pasos finales:

1. Divide la muestra en Training y Validation
2. Divide Training en Splitting y Estimating
3. Estima un árbol en splitting usando τ^2 se minimice. Detente donde hay k observaciones de control y tratamiento.
4. Pruning del árbol con CV
5. Usa el árbol de splitting para **estimar** τ_l en el estimating set.

Causal Forest

Los causal forests son una generalización de los Causal Trees:

- Usan los mismos principios de Bagging de un RF.
- En cada muestra B, ahora el default de `mtry` es $\sqrt{p} + 20$ en lugar de $\sqrt{p}$. Esto es para incluir más variables y controlar por el sesgo de la estimación.

Causal Forest I vs DDML HTE

Veamos los pros y cons de CF vs DDML

- Ambos derivan intervalos de confianza del estimador

Pros:

- Construido explícitamente para Efectos Heterogeneos
- En promedio, insesgado.
- Puede mejorar al DDML si la función $\tau(X)$ es altamente no lineal

Cons:

- Asume explícitamente que $T_i \in \{0, 1\}$
- Es un algoritmo algo lento.
- Sólo se utilizar para RCTs

Generalized Random Forests

Los Generalized Random Forests son una generalización más de los causal forests. Las mejoras son 2:

1. Usa una nueva regla de splitting que apalanca el concepto de **partialling out** de DDML:

$$\hat{\tau} = \frac{\Sigma_i^N \alpha_i(x)(Y_i - \hat{m}^{-i}(X_i))(T_i - \hat{e}^{(-i)}(X_i))}{\Sigma_i^N \alpha_i(x)(T_i - -\hat{e}^{(-i)}(X_i))^2}$$

- Esto permite usar el algoritmo en estudios observacionales.
- Permite incluir T que sean continuos, no sólo dicotómicas.
- Aceleera el tiempo de estimación

Generalized Random Forests II

Analicemos los cambios. Tienes:

$$y_i = \tau T_i + m(X_i) + \epsilon_i$$
$$T_i = g(X_i) + u_i$$

1. Estimas regression forests sobre $\hat{g}(X_i), \hat{m}(X_i)$
2. Genera predicciones **out-of-bag** de estos (Cross-fitting!!!!) $\hat{g}^{(-i)}(X_i), \hat{m}^{(-i)}(X_i)$
3. Usa $\hat{g}^{(-i)}(X_i), \hat{m}^{(-i)}(X_i)$ para crecer el causal forest. Donde en cada árbol vas de forma greedy y te detienes dodne haya k observaciones por grupo de tratamiento.
4. Estimas τ en el estimating set con $\hat{\tau} = \frac{\Sigma_i^N \alpha_i(x)(Y_i - \hat{m}^{-i}(X_i))(T_i - \hat{e}^{(-i)}(X_i))}{\Sigma_i^N \alpha_i(x)(T_i - -\hat{e}^{(-i)}(X_i))^2}$.

- Noten como esto es equivalente a estimar $\tau_i = \frac{Cov(V^y, V^T)}{Var(V^T)}$

Generalized Random Forest III

Noten como los GRF son muy parecidos a un DDML. La diferencia es que:

- GRF usa out of bag estimates para estimar las nuisance functions + Honesty. DDML usa cross-fitting (Muy parecido).
- EL GRF sigue creciendo con base en $Var(\tau) = (T_{i \in I} - \hat{e}^{(-i)}(X_i))^2$. Esto hace que se enfoque directamente en encontrar subgrupos donde los impactos de tratamiento difieren.
- Ambos estiman las nuisance functions.

GRF en R

causal_forest {grf} R Documentation

Causal forest

Description

Trains a causal forest that can be used to estimate conditional average treatment effects tau(X). When the treatment assignment W is binary and unconfounded, we have tau(X) = E[Y(1) - Y(0) | X = x], where Y(0) and Y(1) are potential outcomes corresponding to the two possible treatment states. When W is continuous, we effectively estimate an average partial effect Cov[Y, W | X = x] / Var[W | X = x], and interpret it as a treatment effect given unconfoundedness.

Usage

```
causal_forest(
  X,
  Y,
  W,
  Y.hat = NULL,
  W.hat = NULL,
  num.trees = 2000,
  sample.weights = NULL,
  clusters = NULL,
  equalize.cluster.weights = FALSE,
  sample.fraction = 0.5,
  mtry = min(ceiling(sqrt(ncol(X)) + 20), ncol(X)),
  min.node.size = 5,
  honesty = TRUE,
  honesty.fraction = 0.5,
  honesty.prune.leaves = TRUE,
  alpha = 0.05,
  imbalance.penalty = 0,
  stabilize.splits = TRUE,
  ci.group.size = 2,
  tune.parameters = "none",
  tune.num.trees = 200,
  tune.num.reps = 50,
  tune.num.draws = 1000,
  compute.oob.predictions = TRUE,
  orthog.boosting = FALSE,
  num.threads = NULL,
  seed = runif(1, 0, .Machine$integer.max)
)
```

Arguments

`X`	The covariates used in the causal regression.
`Y`	The outcome (must be a numeric vector with no NAs).
`W`	The treatment assignment (must be a binary or real numeric vector with no NAs).
`Y.hat`	Estimates of the expected responses E[Y | Xi], marginalizing over treatment. If Y.hat = NULL, these are estimated using a separate regression forest. See section 6.1.1 of the GRF paper for further discussion of this quantity. Default is NULL.
`W.hat`	Estimates of the treatment propensities E[W | Xi]. If W.hat = NULL, these are estimated using a separate regression forest. Default is NULL.
`num.trees`	Number of trees grown in the forest. Note: Getting accurate confidence intervals generally requires more trees than getting accurate predictions. Default is 2000.
`sample.weights`	(experimental) Weights given to each sample in estimation. If NULL, each observation receives the same weight. Note: To avoid introducing confounding, weights should be independent of the potential outcomes given X. Default is NULL.
`clusters`	Vector of integers or factors specifying which cluster each observation corresponds to. Default is NULL (ignored).
`equalize.cluster.weights`	If FALSE, each unit is given the same weight (so that bigger clusters get more weight). If TRUE, each cluster is given equal weight in the forest. In this case, during training, each tree uses the same number of observations from each drawn cluster: If the smallest cluster has K units, then when we sample a cluster during training, we only give a random K elements of the cluster to the tree-growing procedure. When estimating average treatment effects, each observation is given weight 1/cluster size, so that the total weight of each cluster is the same. Note that, if this argument is FALSE, sample weights may also be directly adjusted via the sample.weights argument. If this argument is TRUE, sample.weights must be set to NULL. Default is FALSE.
`sample.fraction`	Fraction of the data used to build each tree. Note: If honesty = TRUE, these subsamples will further be cut by a factor of honesty.fraction. Default is 0.5.
`mtry`	Number of variables tried for each split. Default is $\sqrt{p} + 20$ where p is the number of variables.
`min.node.size`	A target for the minimum number of observations in each tree leaf. Note that nodes with size smaller than min.node.size can occur, as in the original randomForest package. Default is 5.
`honesty`	Whether to use honest splitting (i.e., sub-sample splitting). Default is TRUE. For a detailed description of honesty, honesty.fraction, honesty.prune.leaves, and recommendations for parameter tuning, see the grf algorithm reference.
`honesty.fraction`	The fraction of data that will be used for determining splits if honesty = TRUE. Corresponds to set J1 in the notation of the paper. Default is 0.5 (i.e. half of the data is used for determining splits).
`honesty.prune.leaves`	If TRUE, prunes the estimation sample tree such that no leaves are empty. If FALSE, keep the same tree as determined in the splits sample (if an empty leave is encountered, that tree is skipped and does not contribute to the estimate). Setting this to FALSE may

GRF en R II

`alpha`	A tuning parameter that controls the maximum imbalance of a split. Default is 0.05.
`imbalance.penalty`	A tuning parameter that controls how harshly imbalanced splits are penalized. Default is 0.
`stabilize.splits`	Whether or not the treatment should be taken into account when determining the imbalance of a split. Default is TRUE.
`ci.group.size`	The forest will grow ci.group.size trees on each subsample. In order to provide confidence intervals, ci.group.size must be at least 2. Default is 2.
`tune.parameters`	A vector of parameter names to tune. If "all": all tunable parameters are tuned by cross-validation. The following parameters are tunable: ("sample.fraction", "mtry", "min.node.size", "honesty.fraction", "honesty.prune.leaves", "alpha", "imbalance.penalty"). If honesty is FALSE the honesty.* parameters are not tuned. Default is "none" (no parameters are tuned).
`tune.num.trees`	The number of trees in each 'mini forest' used to fit the tuning model. Default is 200.
`tune.num.reps`	The number of forests used to fit the tuning model. Default is 50.
`tune.num.draws`	The number of random parameter values considered when using the model to select the optimal parameters. Default is 1000.
`compute.oob.predictions`	Whether OOB predictions on training set should be precomputed. Default is TRUE.
`orthog.boosting`	(experimental) If TRUE, then when Y.hat = NULL or W.hat is NULL, the missing quantities are estimated using boosted regression forests. The number of boosting steps is selected automatically. Default is FALSE.
`num.threads`	Number of threads used in training. By default, the number of threads is set to the maximum hardware concurrency.
`seed`	The seed of the C++ random number generator.

Value

A trained causal forest object. If tune.parameters is enabled, then tuning information will be included through the 'tuning.output' attribute.

Examples

```
# Train a causal forest.
n <- 500
p <- 10
X <- matrix(rnorm(n * p), n, p)
W <- rbinom(n, 1, 0.5)
Y <- pmax(X[, 1], 0) * W + X[, 2] + pmin(X[, 3], 0) + rnorm(n)
c.forest <- causal_forest(X, Y, W)

# Predict using the forest.
X.test <- matrix(0, 101, p)
X.test[, 1] <- seq(-2, 2, length.out = 101)
c.pred <- predict(c.forest, X.test)

# Predict on out-of-bag training samples.
c.pred <- predict(c.forest)

# Predict with confidence intervals; growing more trees is now recommended.
c.forest <- causal_forest(X, Y, W, num.trees = 4000)
c.pred <- predict(c.forest, X.test, estimate.variance = TRUE)
```

GRF en R III

average_treatment_effect {grf} R Documentation

Estimate average treatment effects using a causal forest

Description

Gets estimates of one of the following.

- The (conditional) average treatment effect (target.sample = all): sum_i = 1^n E[Y(1) - Y(0) | X = Xi] / n
- The (conditional) average treatment effect on the treated (target.sample = treated): sum_Wi = 1 E[Y(1) - Y(0) | X = Xi] / |i : Wi = 1|
- The (conditional) average treatment effect on the controls (target.sample = control): sum_Wi = 0 E[Y(1) - Y(0) | X = Xi] / |i : Wi = 0|
- The overlap-weighted (conditional) average treatment effect sum_i = 1^n e(Xi) (1 - e(Xi)) E[Y(1) - Y(0) | X = Xi] / sum_i = 1^n e(Xi) (1 - e(Xi)), where e(x) = P[Wi = 1 | Xi = x].

This last estimand is recommended by Li, Morgan, and Zaslavsky (JASA, 2017) in case of poor overlap (i.e., when the propensities e(x) may be very close to 0 or 1), as it doesn't involve dividing by estimated propensities.

Usage

```
average_treatment_effect(
  forest,
  target.sample = c("all", "treated", "control", "overlap"),
  method = c("AIPW", "TMLE"),
  subset = NULL
)
```

Arguments

`forest`	The trained forest.
`target.sample`	Which sample to aggregate treatment effects over.
`method`	Method used for doubly robust inference. Can be either augmented inverse-propensity weighting (AIPW), or targeted maximum likelihood estimation (TMLE).
`subset`	Specifies subset of the training examples over which we estimate the ATE. WARNING: For valid statistical performance, the subset should be defined only using features Xi, not using the treatment Wi or the outcome Yi.

Details

If clusters are specified, then each unit gets equal weight by default. For example, if there are 10 clusters with 1 unit each and per-cluster ATE = 1, and there are 10 clusters with 19 units each and per-cluster ATE = 0, then the overall ATE is 0.05 (additional sample.weights allow for custom weighting). If equalize.cluster.weights = TRUE each cluster gets equal weight and the overall ATE is 0.5.

Value

An estimate of the average treatment effect, along with standard error.

Validación

En la práctica, como validas estos modelos? No tienes una y que te indique si el modelo es acertado en predecir τ y τ_i. Por ende se hace lo siguiente:

- Valida el ATE en la base de validación vs el pronosticado por el modelo (promedio de τ_i)
- Construye segmentos (quintiles, deciles) con base en el score τ_i y genera estimaciones de ATE para cada segmento.
- Valida si el impacto observado y el score tienen una relación monotónica y si el valor es acertado.

Referencias

Athey, Susan, Guido Imbens y Stefan Wager (2018). "Approximate Residual Balancing: De-biased Inference of Average Treatment Effects in High Dimensions", Working Paper, arXiv:1604.07125v5

Stefan Wager & Susan Athey (2018) Estimation and Inference of Heterogeneous Treatment Effects using Random Forests, Journal of the American Statistical Association, 113:523, 1228-1242, DOI: 10.1080/01621459.2017.1319839

Chernoszhukov et al. (2018) "Double/debiased machine learning for treatment and structural parameters", Journal of Econometrics 21:C8-C29. doi: 10.1111/ectj.12097

Chernoszhukov, Victor, Vira Semenova, Matt Goldman and Matt Taddy (2021), "Estimation and Inference on Heterogeneous Treatment Effects in High-Dimensional Dynamic Panels", Working paper. arXiv:1712.09988v4 [stat.ML] 16 Mar 2021

Athey, Susan, Julie Tibshirani and Stefan Wager (2018), "Generalized Random Forest", The Annals of Statistics

Referencias II

Belloni, Alexandre, Victor Chernozhukov, and Christian Hansen. "High-Dimensional Methods and Inference on Structural and Treatment Effects". Journal of Economics Perspectives 28-2, p29-50.

Athey, Susan and Guido Imbens (2016), "Recursive partitioning for heterogeneous causal effects". PNAS.

Athey, Susan and Guido W. Imbens (2017), "The State of Applied Econometrics: Causality and Policy Evaluation", Journal of Economic Perspectives—Volume 31, Number 2—Spring 2017—Pages 3–32

Taddy, Matt "Business Data Science: Combining Machine Learning and Economics to Optimize, Automate and Accelerate Business Decisions". Chapter 5,6 and 9. McGraw-Hill, Ed 2019.

Imbens, Guido and Donald Rubin "Causal Inference for Statistics, Social and Biomedical Sciences: An Introduction." Chapter 12. Cambridge University Press. Ed 2019