Métodos de ensamble

Aprendizaje automático

Juan David Martínez
jdmartinev@eafit.edu.co

2023

UNIVERSIDAD EAFIT®

Agenda

- Motivación
- Voto mayoritario
- Bagging
- Boosting
- Random Forest
- Stacking

Motivación

Los métodos de ensamble son los modelos de ML diferentes a redes neuronales más utilizados

"More recently, gradient boosting machines (GBMs) hav
a Kaggler's toolbelt" [1]

[1] Sebastian Raschka, Joshua Patterson, and Corey Nolet (2020)
Machine Learning in Python: Main Developments and Technology Trends in Data Sc
Information 2020, 11, 4

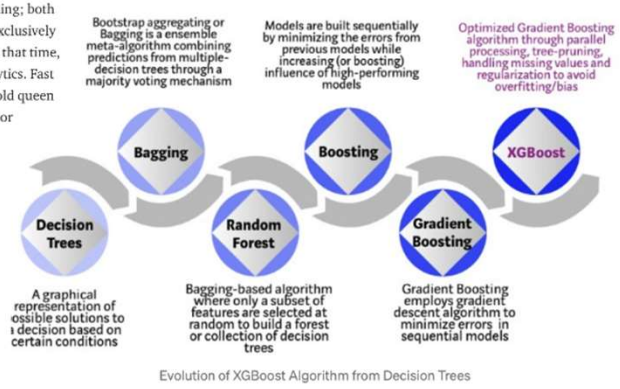
XGBoost Algorithm: Long May She Reign!

The new queen of Machine Learning algorithms taking over the world...

Vishal Morde Apr 7, 2019 · 7 min read

(This article was co-authored with Venkat Anurag Setty)

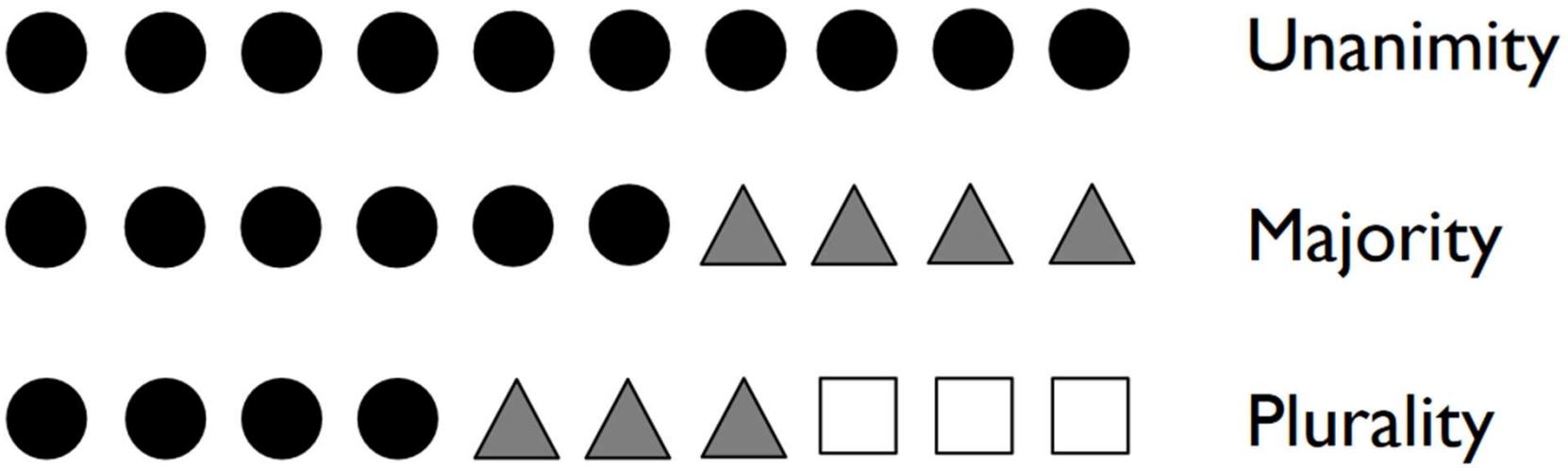
I remember thinking myself, "I got this!". I knew regression modeling; both linear and logistic regression. My boss was right. In my tenure, I exclusively built regression-based statistical models. I wasn't alone. In fact, at that time, regression modeling was the undisputed queen of predictive analytics. Fast forward fifteen years, the era of regression modeling is over. The old queen has passed. Long live the new queen with a funky name; XGBoost or Extreme Gradient Boosting!



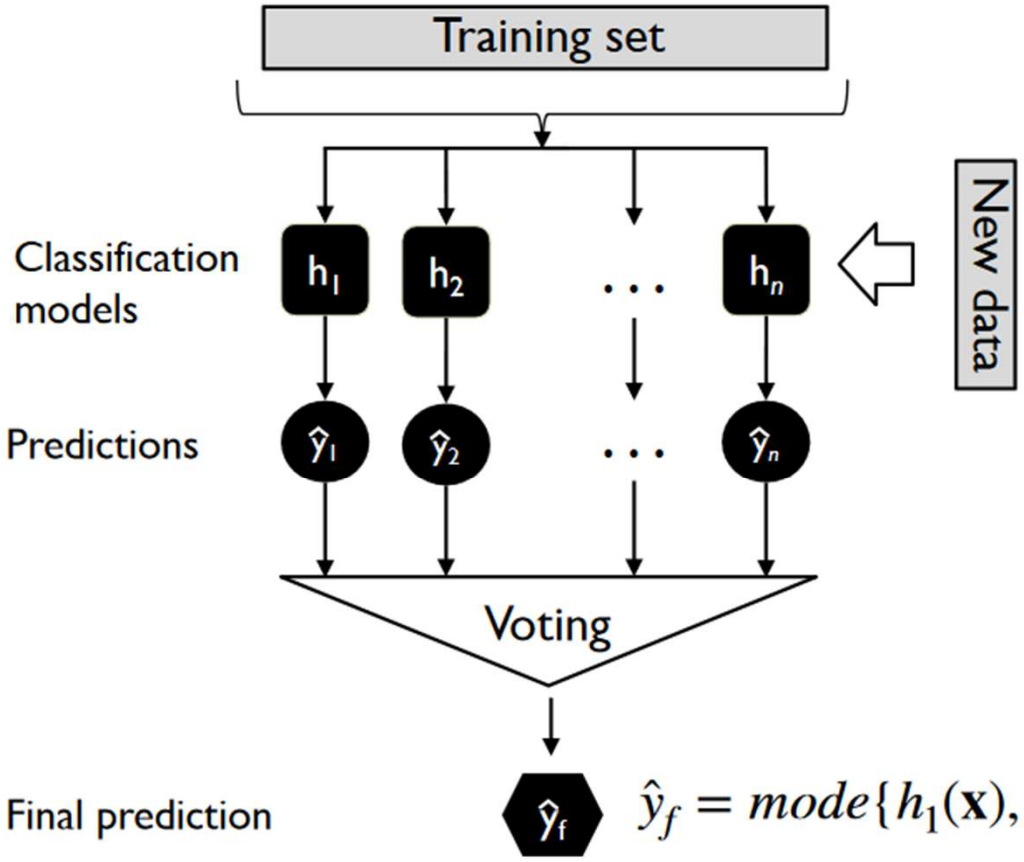
Evolution of XGBoost Algorithm from Decision Trees

UNIVERSIDAD EAFIT

Voto mayoritario



Voto mayoritario



$$\hat{y}_f = mode\{h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots h_n(\mathbf{x})\}$$

where $h_i(\mathbf{x}) = \hat{y}_i$

Voto mayoritario

- Asuma que tenemos n clasificadores independientes cada uno con un error ϵ
- Independientes significa que los errores no están correlacionados
- Asuma una tarea de clasificación binaria
- Asuma también que el clasificador es mejor que una moneda:

$$\forall \epsilon_i \in \{\epsilon_1, \ldots, \epsilon_n\}, \epsilon_i < 0.5$$

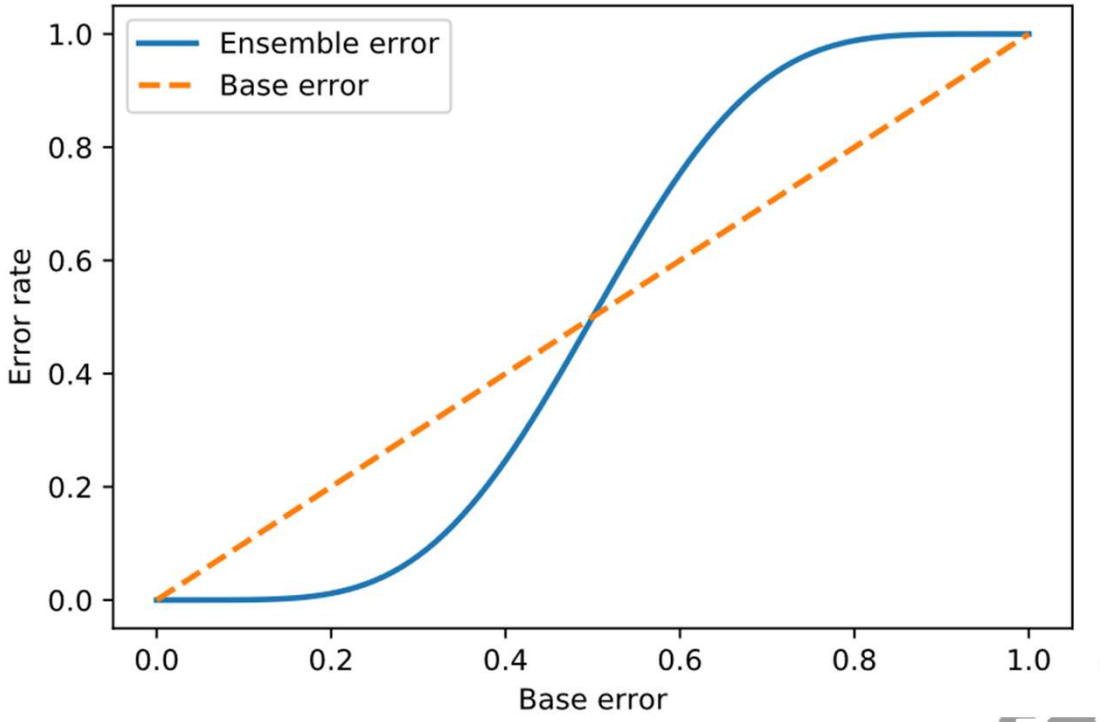
Voto mayoritario

- La probabilidad de tener una predicción errónea usando el ensamble si k clasificadores predicen la misma etiqueta es:

$$\binom{n}{k}\epsilon^k(1-\epsilon)^{n-k} \quad k > \frac{n}{2}$$

- El error del ensamble es:

$$\epsilon_{ens} = \sum_{k}^{n}\binom{n}{k}\epsilon^k(1-\epsilon)^{n-k}$$



Voto suave

$$\hat{y} = \arg\max_j \sum_{i=1}^{n} w_i p_{i,j} \qquad j \in \{0,1\} \qquad h_i(i \in \{1,2,3\})$$

$$h_1(\mathbf{x}) \rightarrow [0.9,0.1]$$
$$h_2(\mathbf{x}) \rightarrow [0.8,0.2]$$
$$h_3(\mathbf{x}) \rightarrow [0.4,0.6]$$

$$p(j=0\,|\,\mathbf{x}) = 0.2 \cdot 0.9 + 0.2 \cdot 0.8 + 0.6 \cdot 0.4 = 0.58$$

$$p(j=1\,|\,\mathbf{x}) = 0.2 \cdot 0.1 + 0.2 \cdot 0.2 + 0.6 \cdot 0.6 = 0.42$$

$$\hat{y} = \arg\max_j \left\{ p(j=0\,|\,\mathbf{x}),\, p(j=1\,|\,\mathbf{x}) \right\}$$

```
clf1 = DecisionTreeClassifier(random_state=1)
clf2 = DecisionTreeClassifier(random_state=1, max_depth=1)
clf3 = DecisionTreeClassifier(random_state=1, max_depth=3)
eclf = EnsembleVoteClassifier(clfs=[clf1, clf2, clf3], weights=[1, 1, 1])
```

Bagging (Bootstrapt Aggregating)

Algorithm 1 Bagging

1: Let n be the number of bootstrap samples
2:
3: **for** i=1 to n **do**
4: Draw bootstrap sample of size m, $\mathcal{D}_i$
5: Train base classifier h_i on $\mathcal{D}_i$
6: $\hat{y} = mode\{h_1(\mathbf{x}), ..., h_n(\mathbf{x})\}$

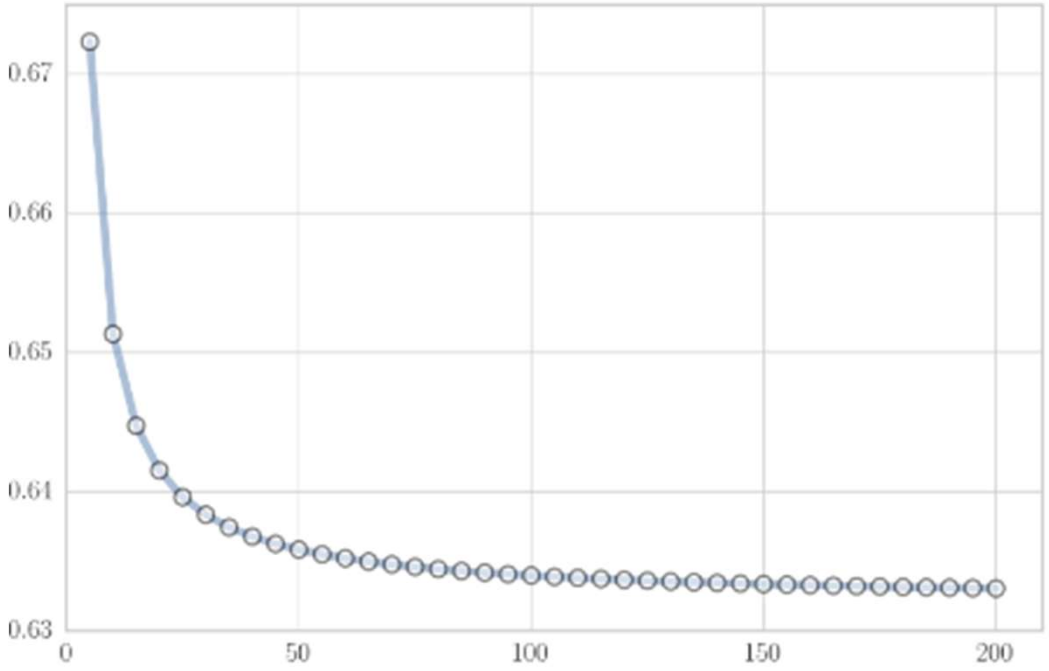
Bootstrap sampling

Bootstrap sampling

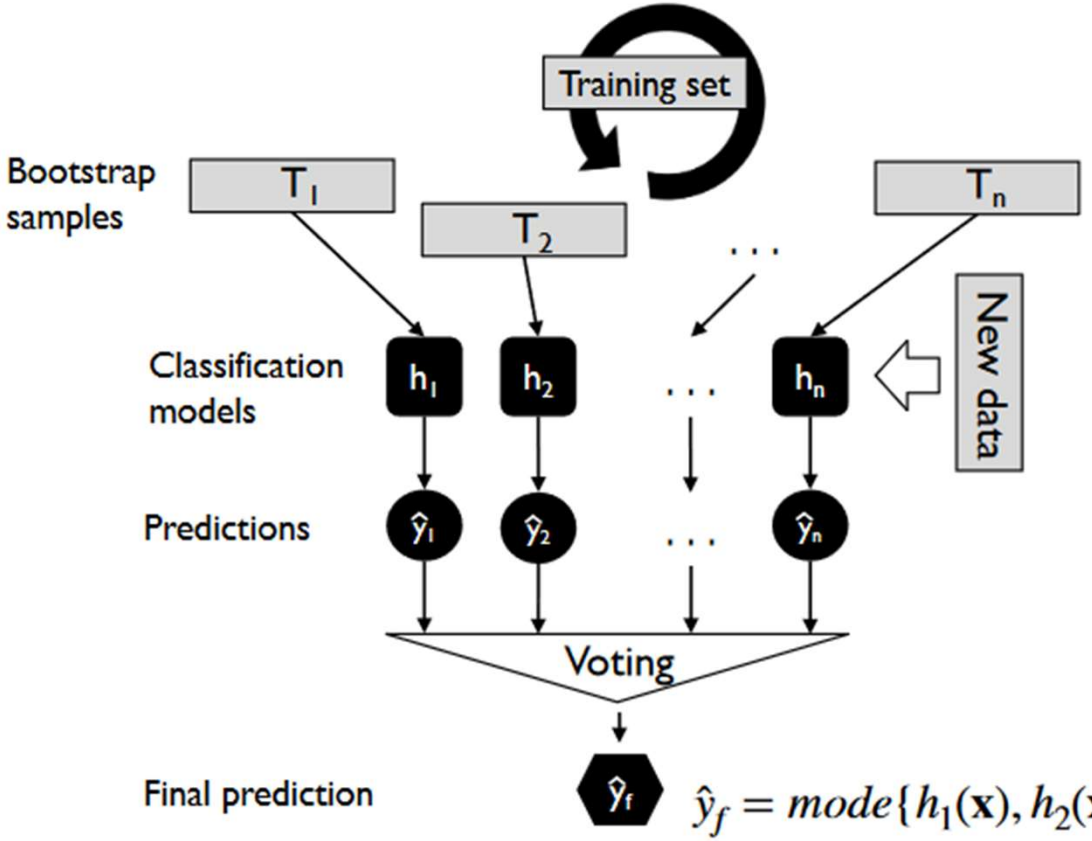
$$P(\text{not chosen}) = \left(1 - \frac{1}{m}\right)^m,$$

$$\frac{1}{e} \approx 0.368, \quad m \to \infty .$$

$$P(\text{chosen}) = 1 - \left(1 - \frac{1}{m}\right)^m \approx 0.632$$



Bagging Classifier

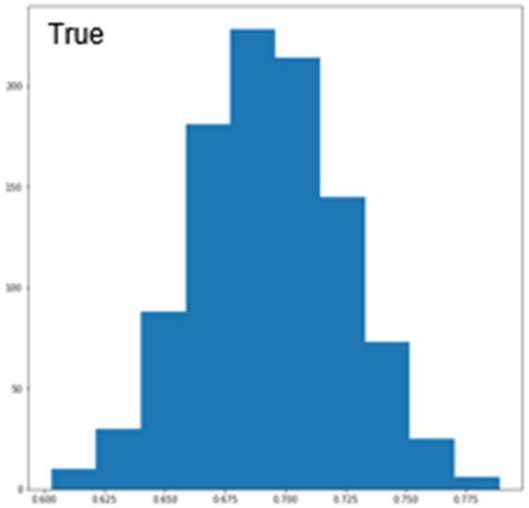


$$\hat{y}_f = mode\{h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots h_n(\mathbf{x})\}$$

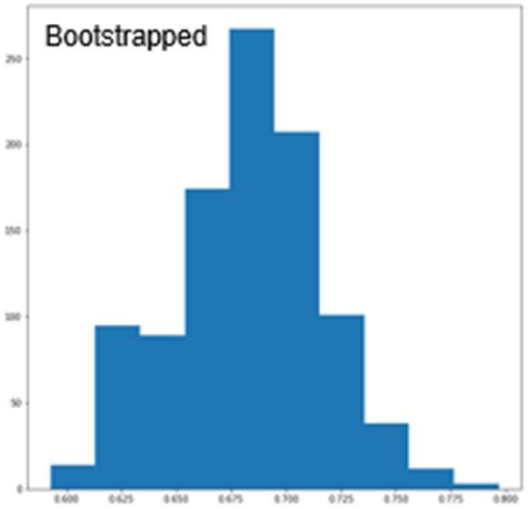
where $h_i(\mathbf{x}) = \hat{y}_i$

Bagging Classifier

```
import matplotlib.pyplot as plt

# histogram of 1000 independent medians of a long
#   tailed distribution (like house prices)
meds = [np.median(-np.log(np.random.rand(1000)))
    for x in range(1000)]

plt.figure(figsize = (10,10))
plt.hist(meds)
```

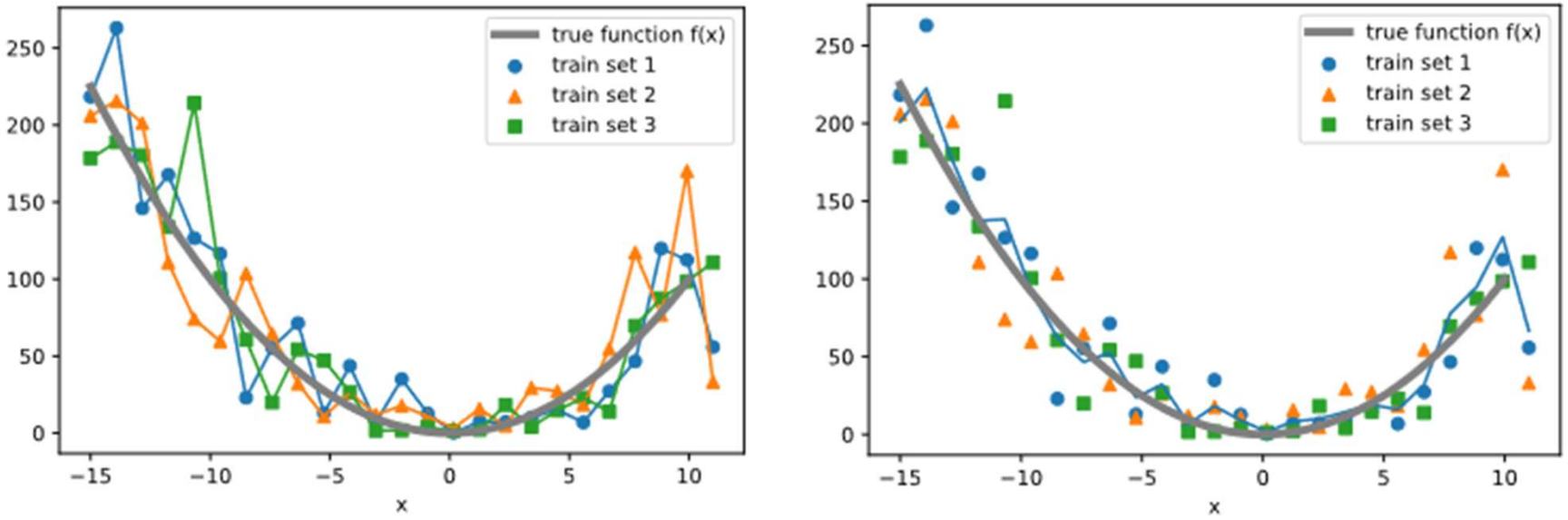



```
# A single distribution of 1000 datapoints ›
data = -np.log(np.random.rand(1000))

# 1000 samples with replacement and histogram
meds = [np.median(np.random.choice(data,1000))
    for x in range(1000)]

plt.figure(figsize = (10,10))
plt.hist(meds)
```



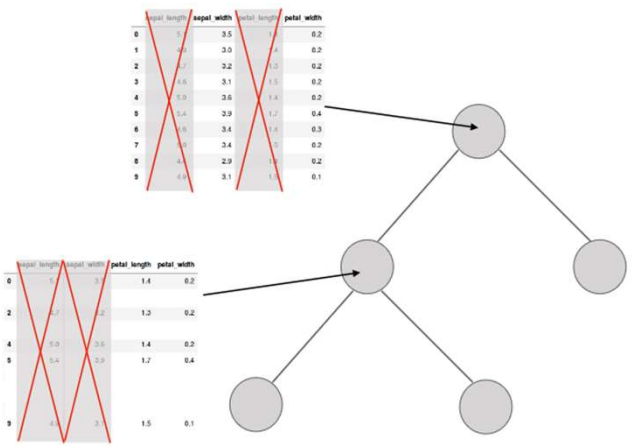
Bagging Classifier

$$\mathbb{E}_{\mathcal{D},\mathbf{y}}\left[(y-\hat{f}_{\mathcal{D}}(\mathbf{x}))^2\right] = \left(f(\mathbf{x})-\bar{f}(\mathbf{x})\right)^2 + \mathbb{E}_{\mathcal{D}}\left[\left(\bar{f}(\mathbf{x})-\hat{f}_{\mathcal{D}}(\mathbf{x})\right)^2\right] + \mathbb{E}_{\epsilon}\left[\epsilon(\mathbf{x})^2\right]$$



UNIVERSIDAD EAFIT

Random Forest



Tin Kam Ho used the "**random subspace method,**" where each tree got a random subset of features.

"Our method relies on an autonomous, pseudo-random procedure to select a small number of dimensions from a given feature space …"

- Ho, Tin Kam. "The random subspace method for constructing decision forests." IEEE transactions on pattern analysis and machine intelligence 20.8 (1998): 832-844.

$$\text{num features} = \log_2 m + 1$$

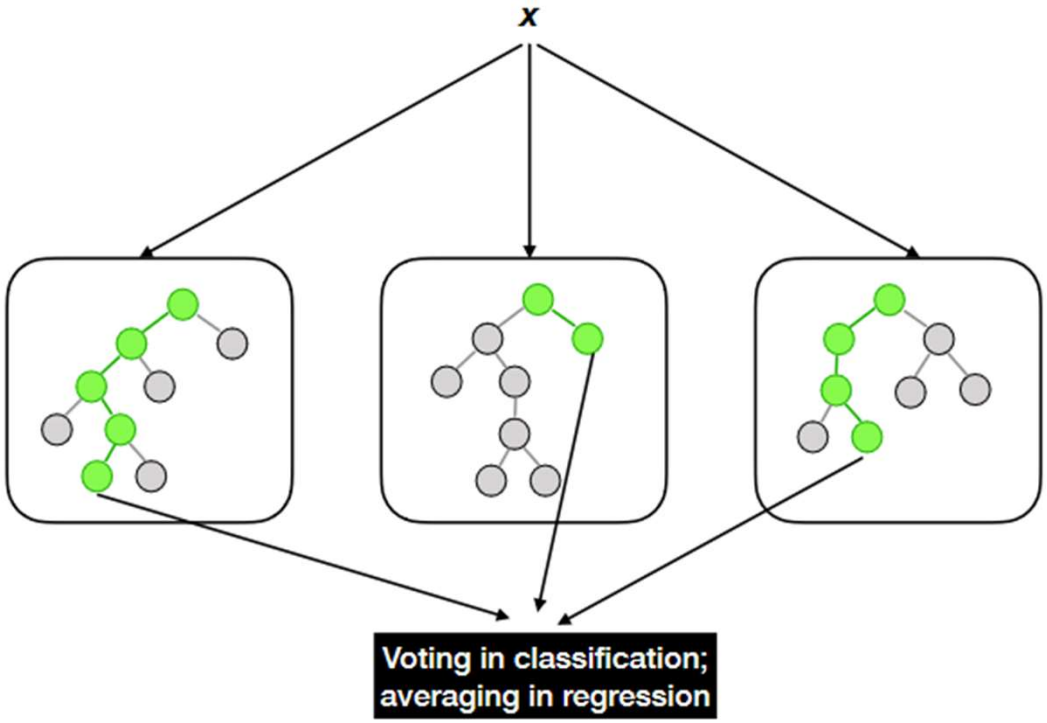
where m is the number of input features

"Trademark" random forest:

"… random forest with random features is formed by selecting at random, at each node, a small group of input variables to split on."

- Breiman, Leo. "Random Forests" Machine learning 45.1 (2001): 5-32.

Random Forest



A diferencia de la publicación original:

Breiman, L., & Cutler, R. A. (2001). Random forests machine learning [J]. *journal of clinical microbiology*, *2*, 199-228.

La implementación en sklearn de Random Forest utiliza Soft Voting.

Random Forest

- Si dos árboles tienen diferentes características, sus predicciones serán independientes
- **Intuitivamente:** Si árboles diferentes entrenados en características diferentes y datos diferentes concuerdan en la misma conclusión, podemos estar seguros de esa conclusión
- **Problema:** hiperparámetros adicionales (tamaño del subconjunto de datos, número de características, número de árboles). Cross-validación
- Para el número de características por árbol: $m_{tree} = \sqrt{m}$

```
RandomForestClassifier(n_estimators=100, criterion='gini',
    max_depth=None, min_samples_split=2, min_samples_leaf=1,
    min_weight_fraction_leaf=0.0, max_features='auto',
    max_leaf_nodes=None, min_impurity_decrease=0.0,
    min_impurity_split=None, bootstrap=True, oob_score=False,
    n_jobs=1)
```

Random Forest

- People have done meta-analysis of ML algorithms, and random forests often come out at or tied for the top.
- The results from a study * that put 179 different ML techniques against 121 different datasets is **on the right**.
- Note: If random forests are not on top, our topic for next lecture is

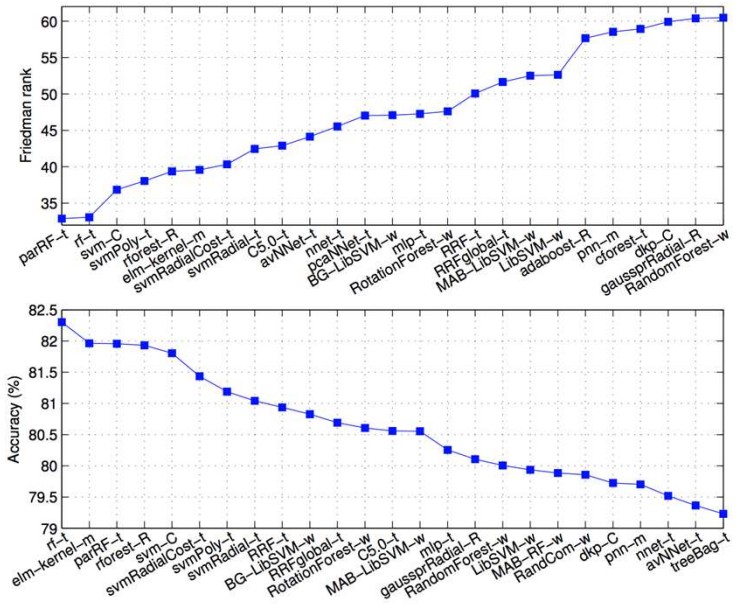


Figure 3: Friedman rank (upper panel, increasing order) and average accuracies (lower panel, decreasing order) for the 25 best classifiers.

M. Fernandez-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we Need Hundredsof Classifiers to Solve Real World Classification Problems?The Journal of MachineLearning Research, 15(1), pp. 3133–3181, 2014

Boosting

Adaptive Boosting - AdaBoost

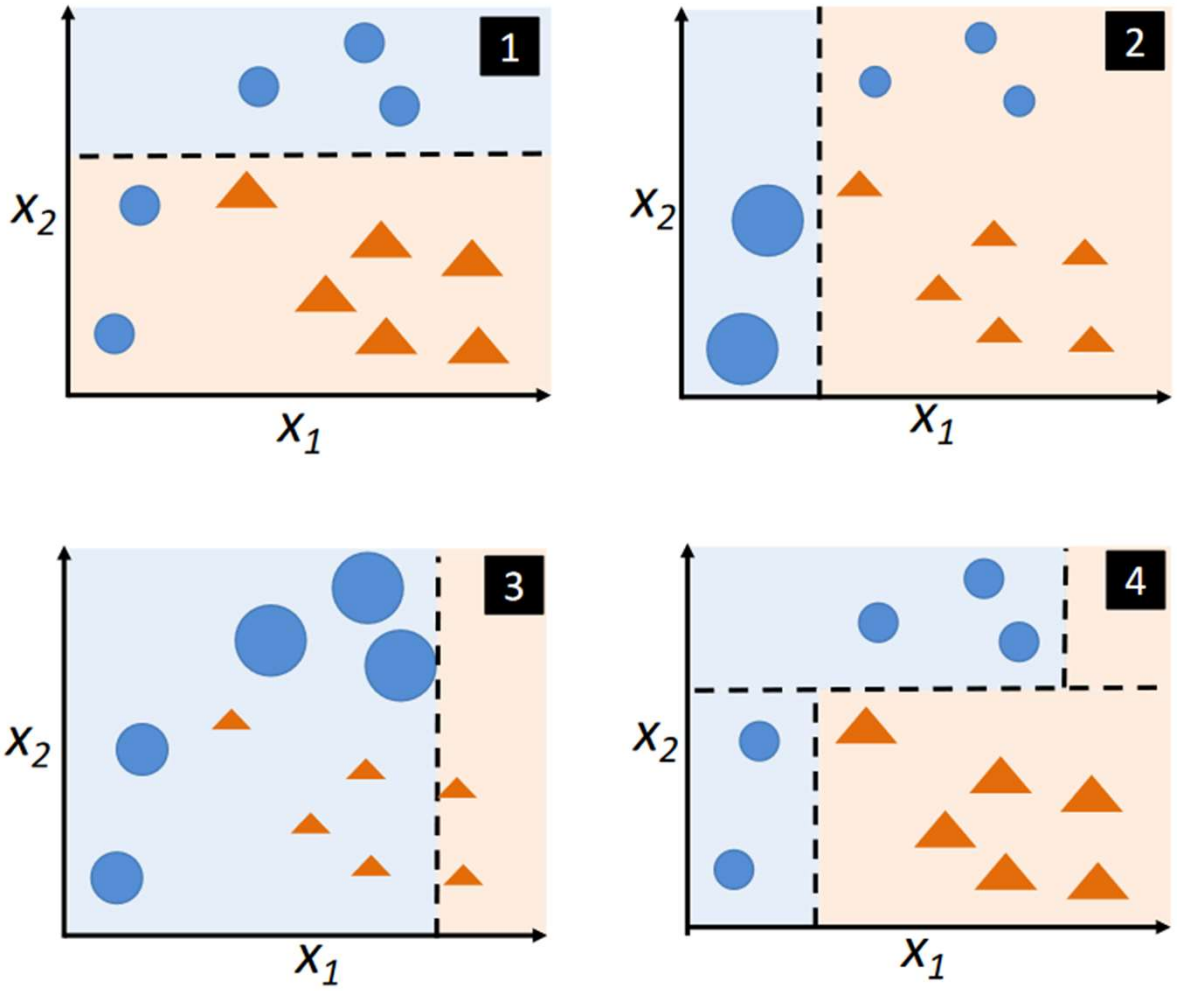
Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, *55*(1), 119-139.

Gradient Boosting – LightGBM, XGBoost, GradientBoostingClassifier

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 1189-1232.

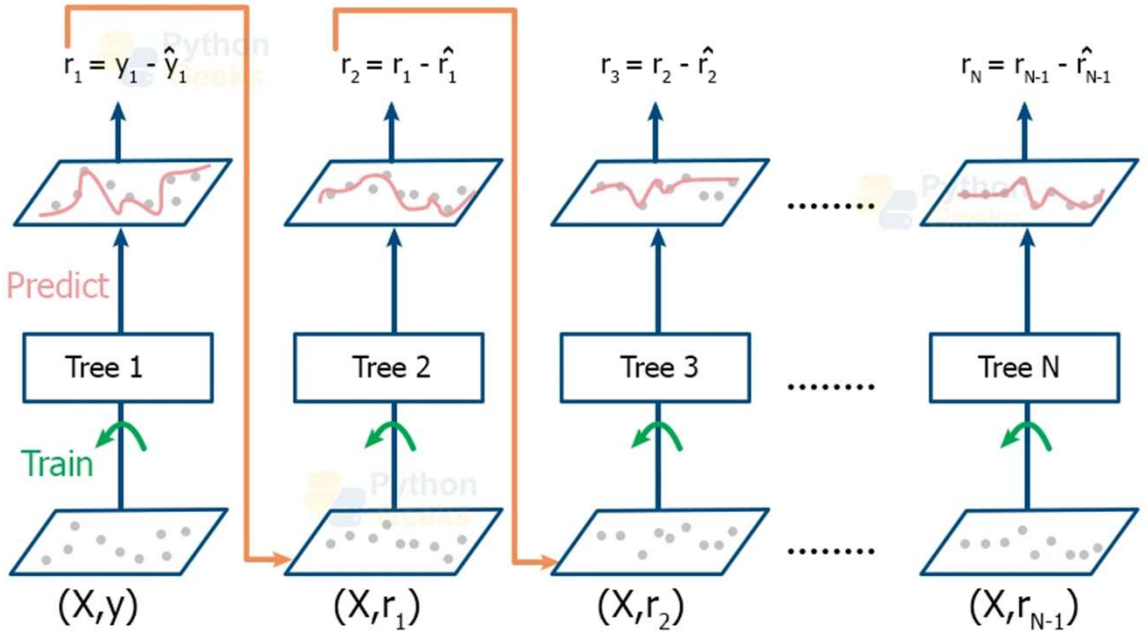
Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785-794). ACM.

AdaBoost in a nutshell



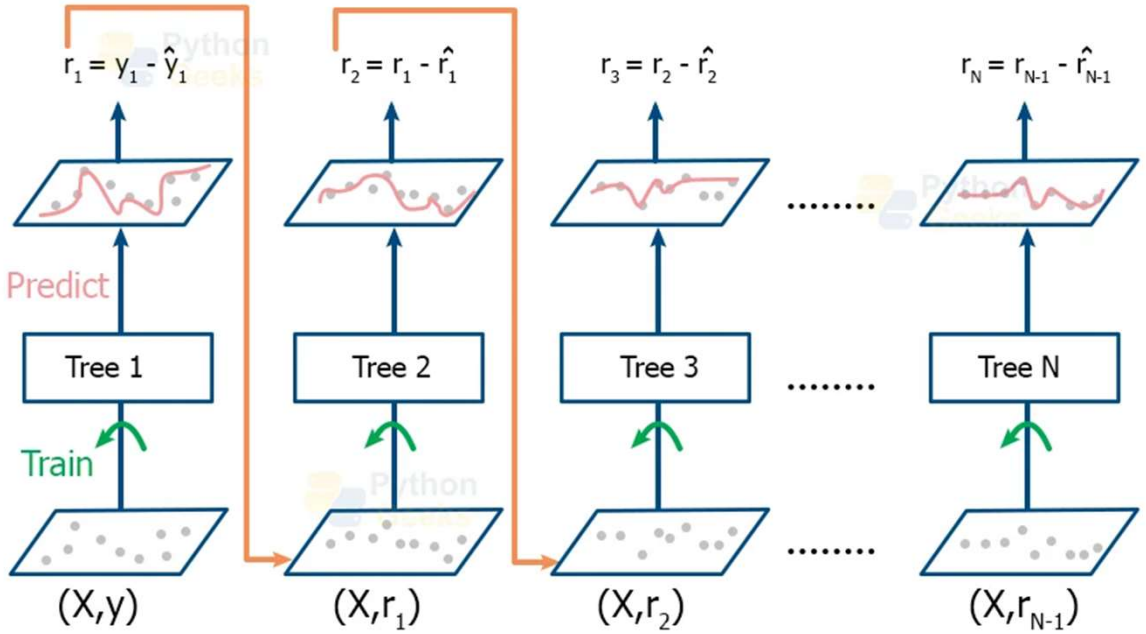
Gradient Boosting

- **Gradient Boosting** es similar a **AdaBoost:** se ajustan árboles secuencialmente para mejorar los errores de los árboles anteriores. Esto puede convertir un “aprendiz débil” en un “aprendiz fuerte”.
- La forma en la que los árboles se ajustan de forma secuencial es diferente:



Gradient Boosting

- **Paso 1:** Construir la base del árbol (únicamente la raíz)
- **Paso 2:** Construir el siguiente árbol basado en los errores del árbol anterior
- **Paso 3:** Combinar el modelo del paso 1 con el modelo del paso 2. Volver al paso 2



Gradient Boosting

In million US Dollars

x1# Rooms	x2=City	x3=Age	y=Price
5	Boston	30	1.5
10	Madison	20	0.5
6	Lansing	20	0.25
5	Waunakee	10	0.1

- **Paso 1:** Construir la base del árbol (únicamente la raíz)

$$\hat{y}_1 = \frac{1}{n}\sum_{i=1}^{n} y^{(i)} = 0.5875$$

Gradient Boosting

- **Paso 2:** Construir el siguiente árbol basado en los errores del árbol anterior

Para esto primero vamos a calcular los residuales $r_1 = y_1 - \hat{y}_1$

In million US Dollars

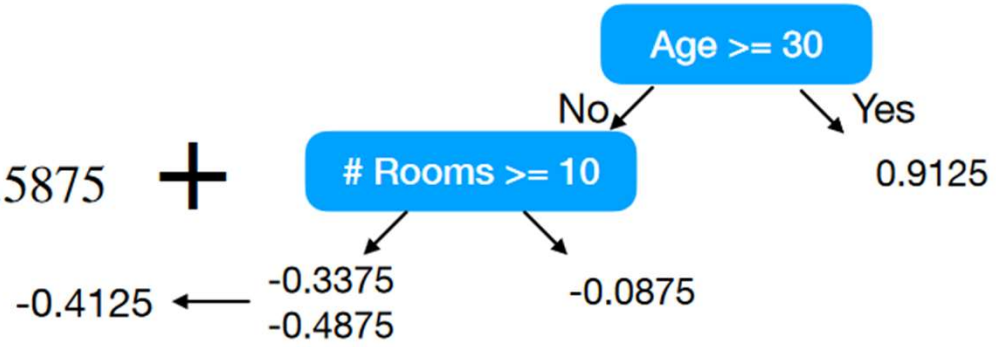
x1#	x2=City	x3=Age	y=Price	r1=Res
5	Boston	30	1.5	1.5 - 0.5875 = 0.9125
10	Madison	20	0.5	0.5 - 0.5875 = -0.0875
6	Lansing	20	0.25	0.25 - 0.5875 = -0.3375
5	Waunake	10	0.1	0.1 - 0.5875 = -0.4875

Gradient Boosting

- **Paso 3:** Combinar el modelo del paso 1 con el modelo del paso 2. Volver al paso 2

x1#	x2=City	x3=Age	y=Price	r=Res
5	Boston	30	1.5	1.5 - 0.5875 = 0.9125
10	Madison	20	0.5	0.5 - 0.5875 = -0.0875
6	Lansing	20	0.25	0.25 - 0.5875 = -0.3375
5	Waunake	10	0.1	0.1 - 0.5875 = -0.4875

$$\hat{y}_1 = \frac{1}{n}\sum_{i=1}^{n} y^{(i)} = 0.5875$$



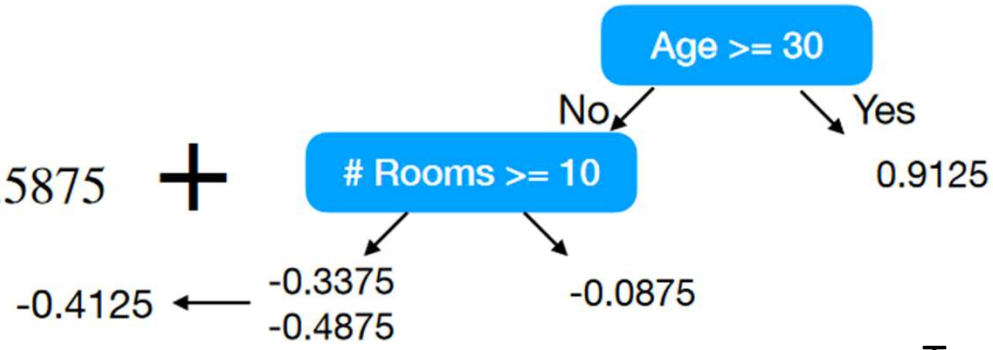
UNIVERSIDAD EAFIT

Gradient Boosting

- **Paso 3:** Combinar el modelo del paso 1 con el modelo del paso 2. Volver al paso 2

x1#	x2=City	x3=Age	y=Price	r=Res
5	Boston	30	1.5	1.5 - 0.5875 = 0.9125
10	Madison	20	0.5	0.5 - 0.5875 = -0.0875
6	Lansing	20	0.25	0.25 - 0.5875 = -0.3375
5	Waunake	10	0.1	0.1 - 0.5875 = -0.4875

$$\hat{y}_1 = \frac{1}{n}\sum_{i=1}^{n} y^{(i)} = 0.5875$$



$$\hat{y} = 0.5875 + \alpha(-0.4125)$$

- **α:** Tasa de aprendizaje entre 0 y 1.
- Si **α=1** bajo sesgo pero alta varianza

Gradient Boosting

- Asumamos que tenemos un problema de regresión donde tenemos los datos de entrada $\left\{\left(\mathbf{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{N}$
- El objetivo es producir un modelo $h(\mathbf{x}^{(i)})$ que minimice la siguiente función de costo:

$$\mathcal{L}(f(\mathbf{x})) = \frac{1}{2N}\sum_{i=1}^{N}(y^{(i)} - f(\mathbf{x}^{(i)}))^2$$

- El término $2N$ se utiliza como un truco de derivación.
- Suponga que tenemos un modelo f_0 que intenta resolver el problema, ¿qué deberíamos hacer?

Gradient Boosting

- Lo que podemos hacer es mirar los errores o residuales:

$$\epsilon^{(i)} = y^{(i)} - f_0(\mathbf{x}^{(i)})$$

- Con estos errores, podemos intentar aprender un nuevo modelo $g_0(\mathbf{x}^{(i)})$ para predecir estos errores:

$$\mathcal{L}(g_0) = \frac{1}{2N}\sum_{i=1}^{N}(\epsilon^{(i)} - g_0(\mathbf{x}^{(i)}))^2$$

- O más intuitivamente $g_0(\mathbf{x}^{(i)}) = \epsilon_i$
- Podemos producir entonces un nuevo modelo

$$f_1(\mathbf{x}^{(i)}) = f_0(\mathbf{x}^{(i)}) + g_0(\mathbf{x}^{(i)})$$

Gradient Boosting

- Tomar un modelo f_k
- Calcular los residuos $\epsilon^{(i)} = y^{(i)} - f_k(\mathbf{x}^{(i)})$
- Ajusta un modelo $g_k(\mathbf{x}^{(i)}) \approx \epsilon^{(i)}$
- Definir un nuevo modelo $f_{k+1}(\mathbf{x}^{(i)}) = f_k(\mathbf{x}^{(i)}) + g_k(\mathbf{x}^{(i)})$
- De forma general podemos empezar con cualquier modelo:

$$f_0(\mathbf{x}^{(i)}) = \frac{1}{N} \sum_i y^{(i)}$$

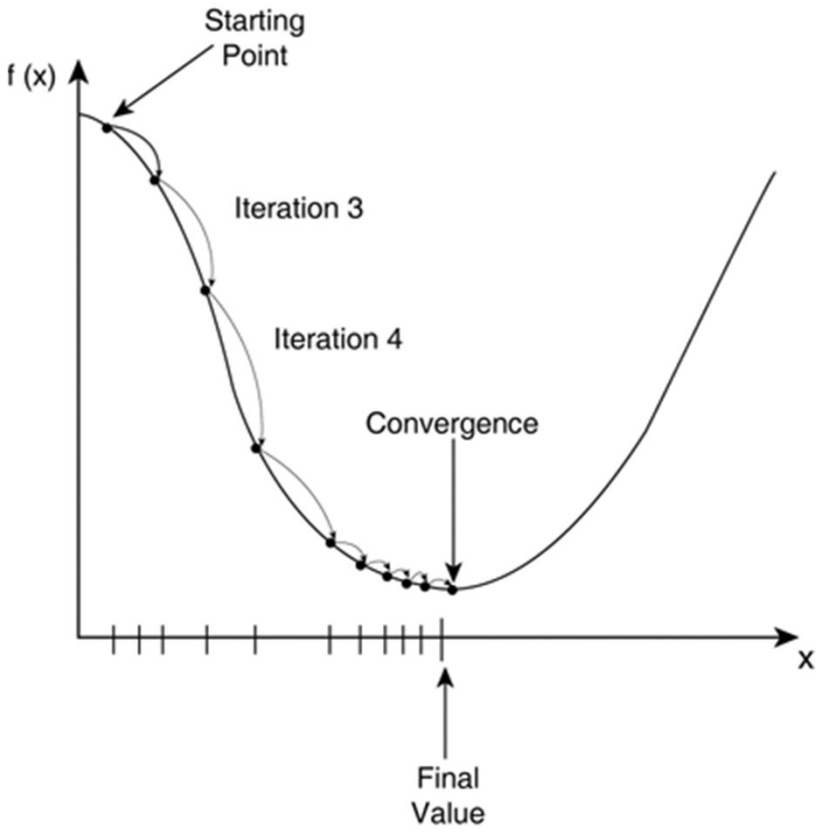
- Desde que el modelo g_k prediga mejor que "adivinar", se mejorará la pérdida en cada iteración

Gradient Boosting

- ¿Qué significa **mejorar**?
  - **Para regresión** que el modelo reduzca la función de costo (más que adivinar)
  - **Para clasificación** que el modelo clasifique mejor que adivinar la clase de cada muestra
- Un buen ejemplo son árboles de decisión cortos o de un solo nodo (aprendices débiles)
- Por lo general, estos modelos tienen un alto sesgo

Gradient Boosting

- **¿Por qué gradient boosting?**
- **Optimización basada en gradiete:** Tipo de optimización utilizada en redes neuronales.
- De forma general:



Gradient Boosting

- Nuestra pérdida es una suma de pérdidas para cada muestra:

$$\mathcal{L}^{(i)} = \frac{1}{2}(y^{(i)} - f(\mathbf{x}^{(i)}))^2$$

- Si calculamos la derivada de la pérdida y reorganizamos unos términos, tenemos:

$$\epsilon^{(i)} = y^{(i)} - f(\mathbf{x}^{(i)}) = -\frac{\partial \mathcal{L}^{(i)}}{\partial f(\mathbf{x}^{(i)})}$$

- ¡Los residuos son el gradiente negativo de la función de costo con respecto a nuestra predicción!

Gradient Boosting

- Tomar un modelo f_k
- Calcular los residuos $\epsilon^{(i)} = y^{(i)} - f_k(\mathbf{x}^{(i)})$ $\epsilon^{(i)} = -\frac{\partial \mathcal{L}^{(i)}}{\partial f_k(\mathbf{x}^{(i)})}$
- Ajusta un modelo $g_k(\mathbf{x}^{(i)}) \approx \epsilon^{(i)}$
- Definir un nuevo modelo $f_{k+1}(\mathbf{x}^{(i)}) = f_k(\mathbf{x}^{(i)}) + g_k(\mathbf{x}^{(i)})$
- De forma general podemos empezar con cualquier modelo:

$$f_0(\mathbf{x}^{(i)}) = \frac{1}{N} \sum_i y^{(i)}$$

- Lo que estamos haciendo es **gradiente descendente** en la función de costo con respecto al modelo, donde aproximamos el gradiente usando un aprendiz débil.

Gradient Boosting

- El aprendiz débil es crucial dado que genera de forma natural una forma de regularización, donde las actualizaciones del algoritmo de optimización son simples por defecto. Si fueran aprendices fuertes, en una sola iteración el modelo se sobre-entrenaría.

- El tamaño de paso no tiene que ser necesariamente 1. Se puede utilizar alguna forma de optimización de línea (line search):

$$f_{k+1}(\mathbf{x}^{(i)}) = f_k(\mathbf{x}^{(i)}) + \gamma_k g_k(\mathbf{x}^{(i)}).$$

- Podemos agregar un tamaño de paso adicional v, conocido como término de "recogimiento" (shrinkage), de forma que la actualización es:

$$f_{k+1}(\mathbf{x}^{(i)}) = f_k(\mathbf{x}^{(i)}) + v\gamma_k g_k(\mathbf{x}^{(i)}).$$

Gradient Boosting

- Algunas selecciones comunes de la función de costo para problemas de regresión son:
- Error cuadrático medio:

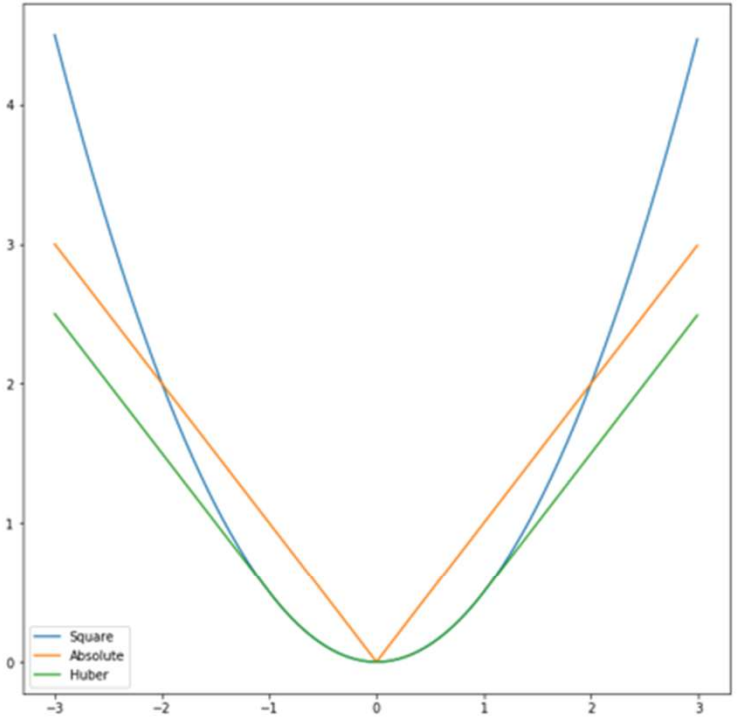
$$\mathcal{L}(f) = \frac{1}{2N} \sum_i (y^{(i)} - f(\mathbf{x}^{(i)}))^2$$

- Error absoluto medio:

$$\mathcal{L}(f) = \frac{1}{2N} \sum_i |y^{(i)} - f(\mathbf{x}^{(i)})|$$

- Hubber

$$\mathcal{L}(f) = \frac{1}{N} \sum_i \begin{cases} \frac{1}{2}(y^{(i)} - f(\mathbf{x}^{(i)}))^2 & |y^{(i)} - f(\mathbf{x}^{(i)})| < 1 \\ \frac{1}{2}|y^{(i)} - f(\mathbf{x}^{(i)})| - \frac{1}{2} & |y^{(i)} - f(\mathbf{x}^{(i)})| \geq 1 \end{cases}$$



Gradient Boosting

- Algunas selecciones comunes de la función de costo para problemas de clasificación son:

- Pérdida de regresión logística:

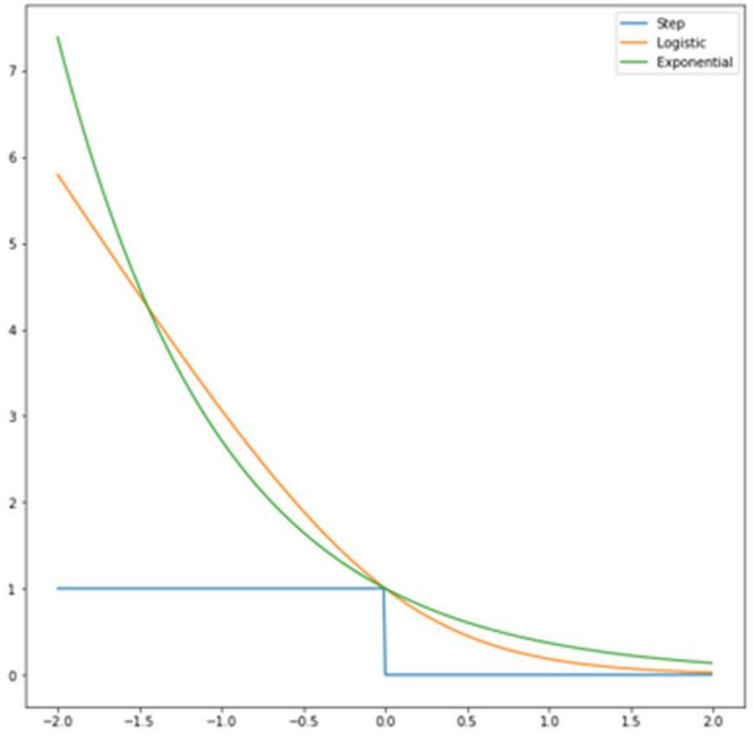
$$\mathcal{L}(f) = \frac{1}{N} \sum_i \log \left[1 + \exp(-2y^{(i)} f(\mathbf{x}^{(i)}) \right]$$

- Pérdida exponencial (AdaBoost):

$$\mathcal{L}(f) = \frac{1}{N} \sum_i \exp(-2y^{(i)} f(\mathbf{x}^{(i)}))$$

- Pérdida bisagra (hinge-loss SVMs):

$$\mathcal{L}(f) = \frac{1}{N} \sum_i \max(0, 1 - y^{(i)} f(\mathbf{x}^{(i)}))$$



Gradient Boosting

XGBoost

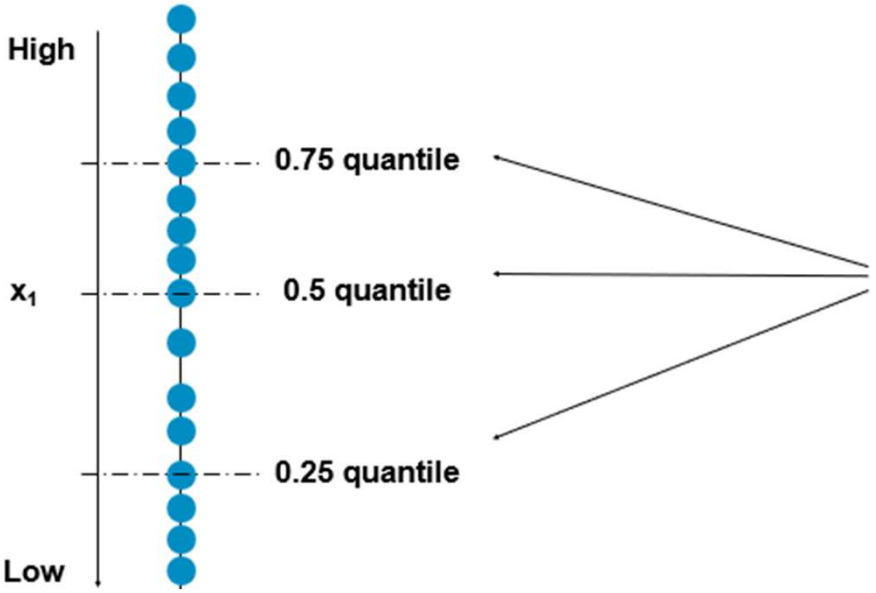
- XGBoost stands for **"Extreme Gradient Boosting"**
- Algorithm details are described in the 2016 paper* here.
- Some contributions of this model (there are many more):
  - Approximate Greedy Algorithm
  - Reformulate split conditions and gains considering weights on leaves.
  - Performance improvement techniques: Parallel blocks, Cache-aware Access and Out-of-core computation

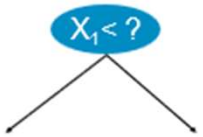
* Chen, T. & Guestrin, C. Xgboost: A scalable tree boosting system. In Proc. 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining 785–794 (ACM, 2016).

Gradient Boosting

Approximate Greedy Algorithm

- Trying all possible split thresholds can be time consuming for large datasets.
- XGBoost uses **quantiles** as threshold candidates. Assume a feature ordered like this:





- We can use these three quantiles as the splits to try.
- There are usually more than three quantiles and they apply some extra optimizations to do this in parallel.

Gradient Boosting

XGBoost Interface

Here is the XGBoostClassifier interface with some default parameters. The full interface is longer.

```
XGBClassifier(max_depth=3,
              learning_rate=0.1,
              n_estimators=100,
              objective='binary:logistic',
              booster='gbtree',
              n_jobs=1,
              ...)
```

Gradient Boosting

LightGBM: A Highly Efficient Gradient Boosting Decision Tree

Guolin Ke[1], **Qi Meng**[2], **Thomas Finley**[3], **Taifeng Wang**[1],
Wei Chen[1], **Weidong Ma**[1], **Qiwei Ye**[1], **Tie-Yan Liu**[1]
[1]Microsoft Research [2]Peking University [3] Microsoft Redmond
[1]{guolin.ke, taifengw, wche, weima, qiwye, tie-yan.liu}@microsoft.com;
[2]qimeng13@pku.edu.cn; [3]tfinely@microsoft.com;

LightGBM

CatBoost: unbiased boosting with categorical features

Liudmila Prokhorenkova[1,2], **Gleb Gusev**[1,2], **Aleksandr Vorobev**[1],
Anna Veronika Dorogush[1], **Andrey Gulin**[1]
[1]Yandex, Moscow, Russia
[2]Moscow Institute of Physics and Technology, Dolgoprudny, Russia
{ostroumova-la, gleb57, alvor88, annaveronika, gulin}@yandex-team.ru

CatBoost

Variables categóricas

UNIVERSIDAD EAFIT

Gradient Boosting

	XGBoost	LightGBM	CatBoost
Categorical variables	Needs to be **one-hot-encoded** or **target-encoded** beforehand	• Set column type as "category" in data frame, or • Use **categorical_feature** parameter	• Set column type as "category" in data frame, or • Use **cat_features** parameter
Missing values	Branch directions are learned for missing values	Skips the data point during split, allocates later to leaves	Numerical missing values are set to min value for that feature (default)