



Machine Learning Advanced

- Aprendizaje Supervisado - Modelamiento -

Algoritmos de clasificación

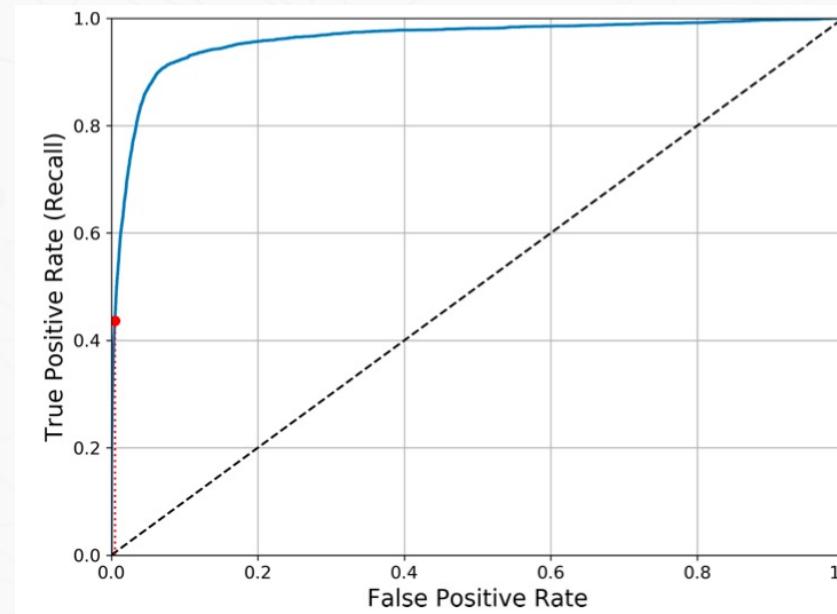
Docente: Manuel Montoya



Agenda

1. Evaluación de clasificación
2. Modelos ensamblados: Bagging
3. Optimización de hiperparámetros
4. Modelos ensamblados: Boosting
5. Validación Cruzada
6. Regularización

1. Evaluación de clasificación



Evaluación de modelos de clasificación: Matriz de confusión

- Es una tabla de tamaño $m \times m$, donde m es el número de valores que toma la variable objetivo
- Un clasificador perfecto tendría todos los elementos en la diagonal

Clase Original	<i>Clasificado como ...</i>	
	No Spam	Spam
No Spam	820	20
Spam	40	120

Evaluación de modelos de clasificación: Matriz de confusión

Clasificado como ...

<i>Clase Original</i>		
	No Spam	Spam
No Spam	Verdaderos Negativos	Falsos Positivos
Spam	Falsos Negativos	Verdaderos Positivos

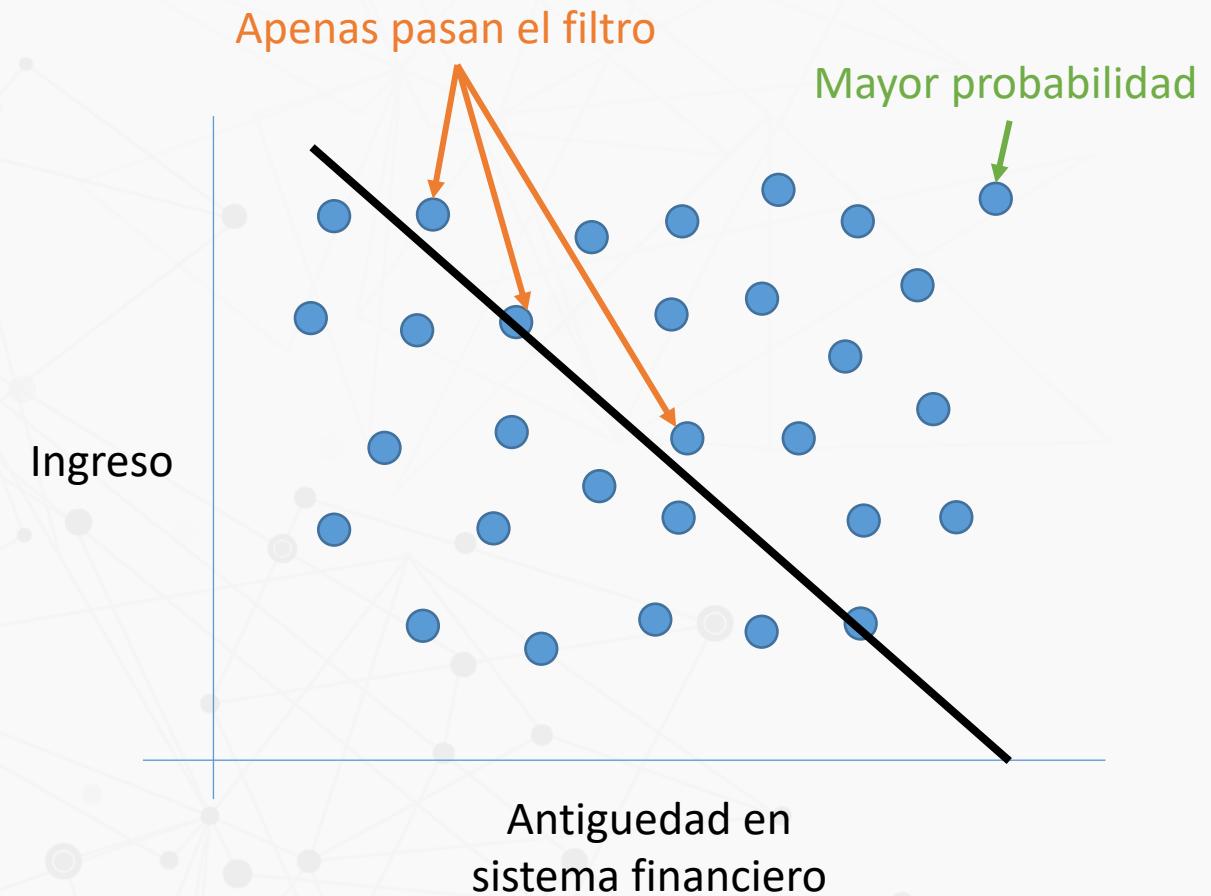
Evaluación de modelos de clasificación: Matriz de confusión

Clasificado como ...

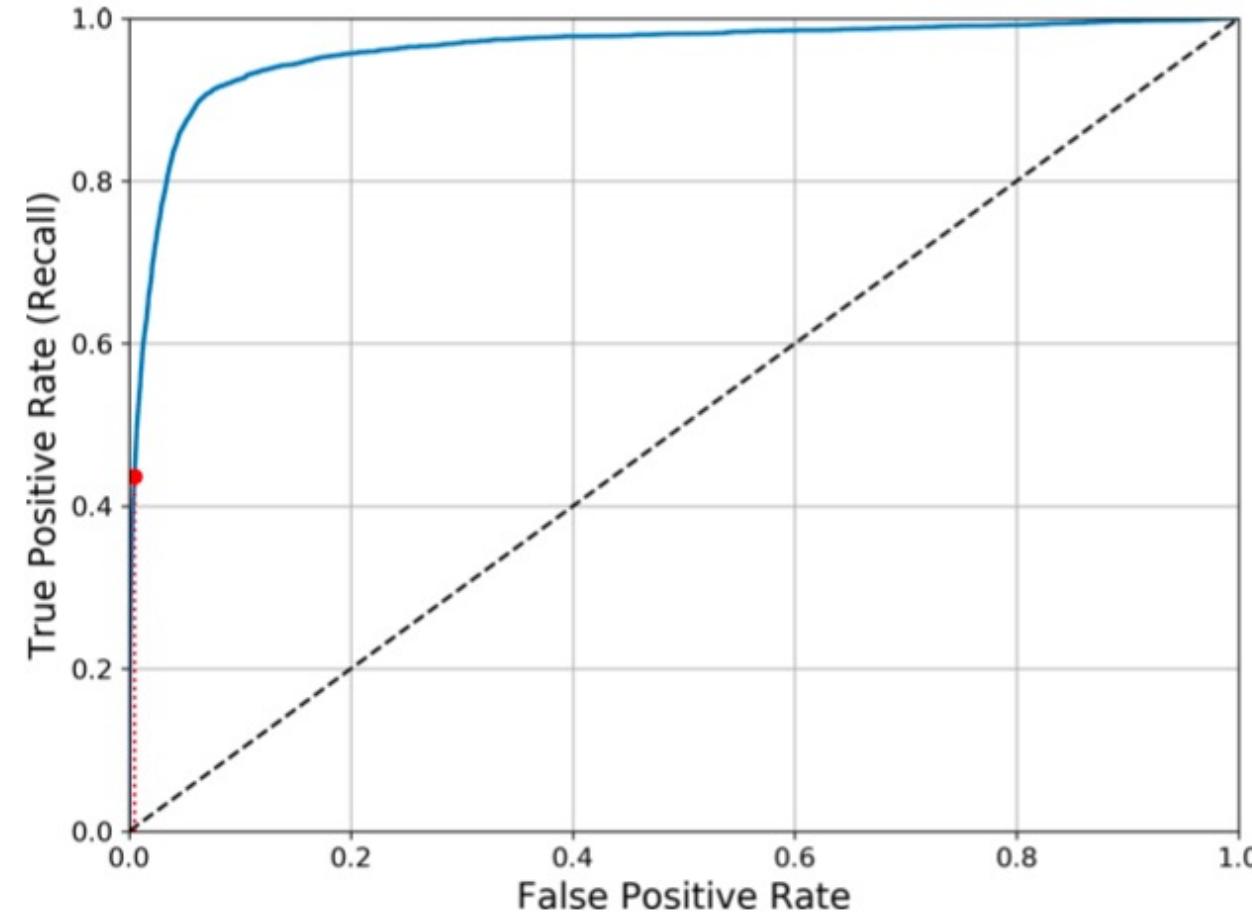
Clase Original		
	No Spam	Spam
No Spam	True Negative (TN)	False Positive (FP)
Spam	False Negative (FN)	True Positive (TP)

Predicción de probabilidades

- En los modelos de predicción reales, no se predice directamente una clase, sino se predice una probabilidad.
- ¿Qué tan probable es que un cliente me pague un crédito?
- Esto nos permite ordenar los registros en base a la probabilidad de mayor a menor y priorizar aquellos clientes que estamos seguros nos van a pagar.



Curva ROC



Score ROC AUC

- Es una medida de qué tan bien ordena el modelo según las probabilidades
- Se calcula utilizando la curva ROC entre el output del modelo y las etiquetas reales
- Un modelo que ordena adecuadamente los registros tiene el valor del score ROC cercano a 1. Un modelo que no ordena bien tiene un score cercano a 0.

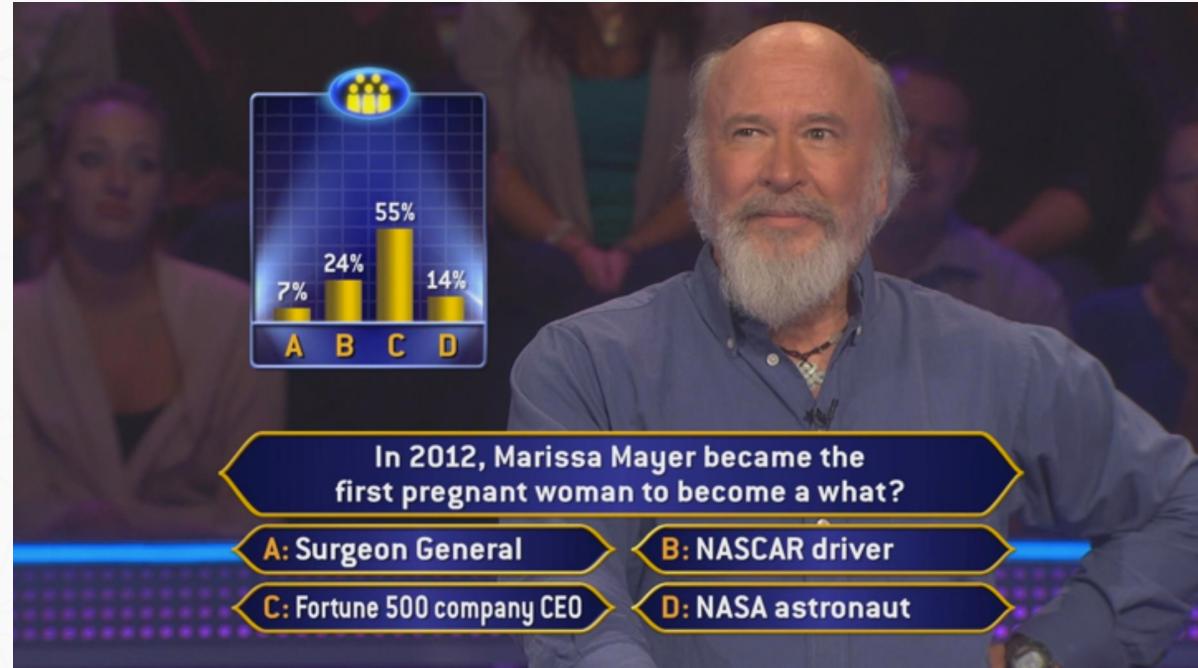
Predictión	Pagó
0.99	1
0.96	1
0.85	1
0.73	0
0.56	1
0.40	1
0.20	0
0.10	0
0.05	0

ROC AUC Score = 0.90

Predictión	Pagó
0.99	0
0.96	0
0.85	1
0.73	0
0.56	1
0.40	0
0.20	0
0.10	1
0.05	0

ROC AUC Score = 0.44

2. Modelos ensamblados



Intuición

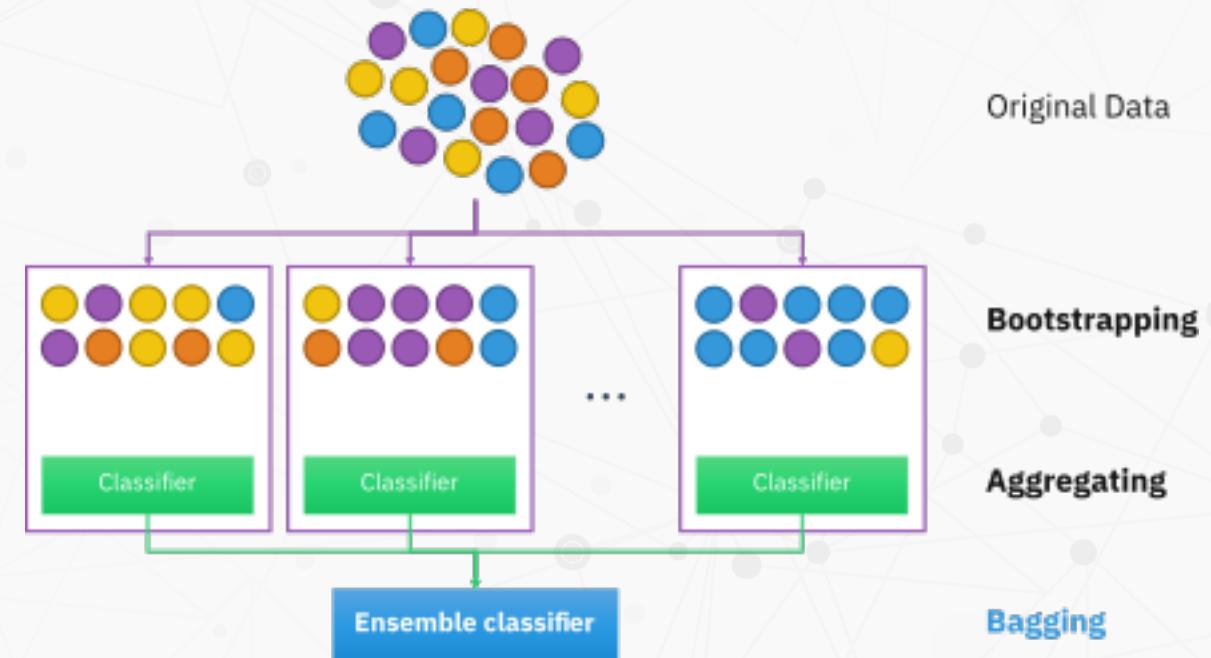
- Promediar modelos de alta varianza conduce a predicciones más estables y confiables.
- Se entrena múltiples algoritmos en un mismo conjunto de datos, o un mismo algoritmo en múltiples variaciones del conjunto de datos.
- Luego se combina las predicciones usando promedios ponderados o votación

Ensemble Learning

Son “meta-algoritmos” que combinan técnicas de aprendizaje de máquina en un único modelo predictivo con el objetivo de:

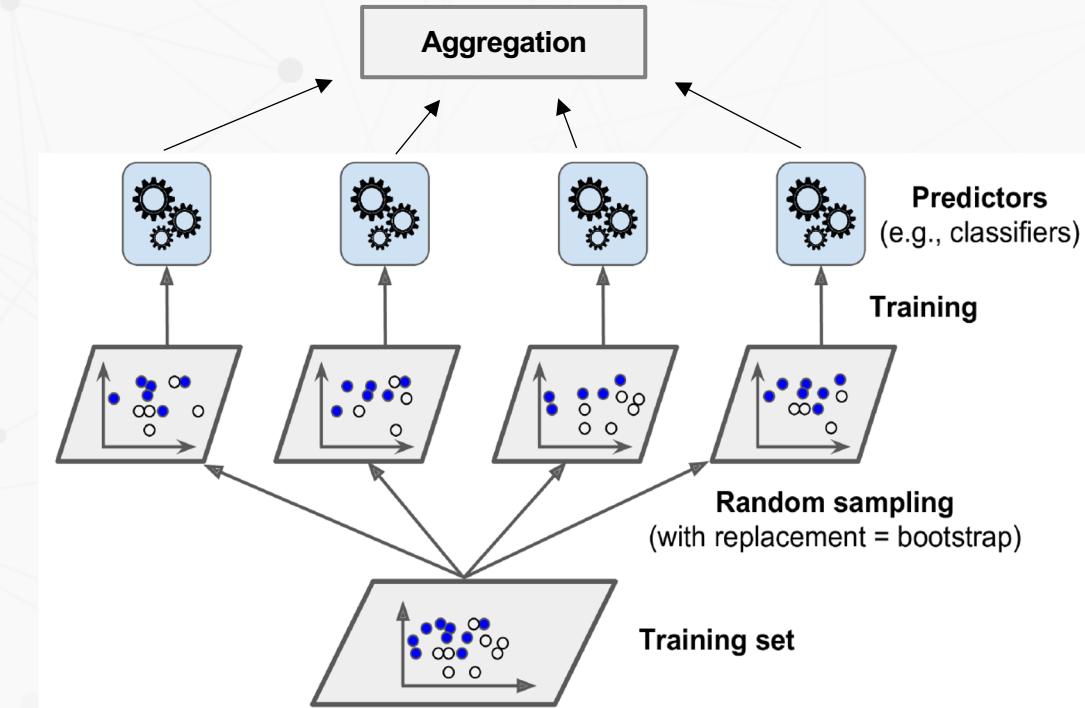
- Disminuir el error de **varianza** (*bagging*)
- Disminuir el error de **bias** (*boosting*)
- Mejorar el acierto de la predicción (*stacking*)

2.1 Bagging (Bootstrap Aggregating)



Bagging (Bootstrap Aggregating)

- ¿**Bootstrap?** tipo de muestreo (muestras aleatorias con reemplazo)
- Se construyen diferentes “*bootstrap samples*” a partir del conjunto de entrenamiento para cada clasificador.
- Ventaja: puede **reducir la varianza** en modelos complejos. Altamente paralelizable



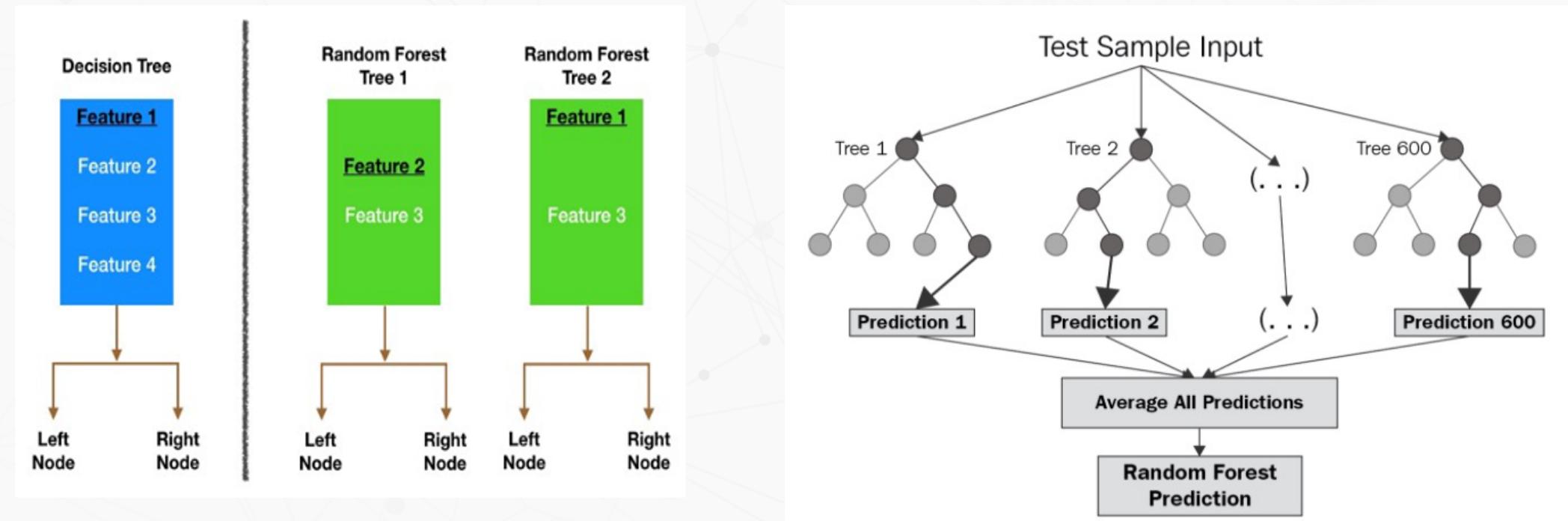
Random Forest

Modelo de ensamblaje compuesto por:

**Bagging + Decision trees + Feature Randomness
(subspace sampling)**

Reduce la varianza y ayuda a evitar el sobreajuste
Promueve la no colinealidad entre los subarboles





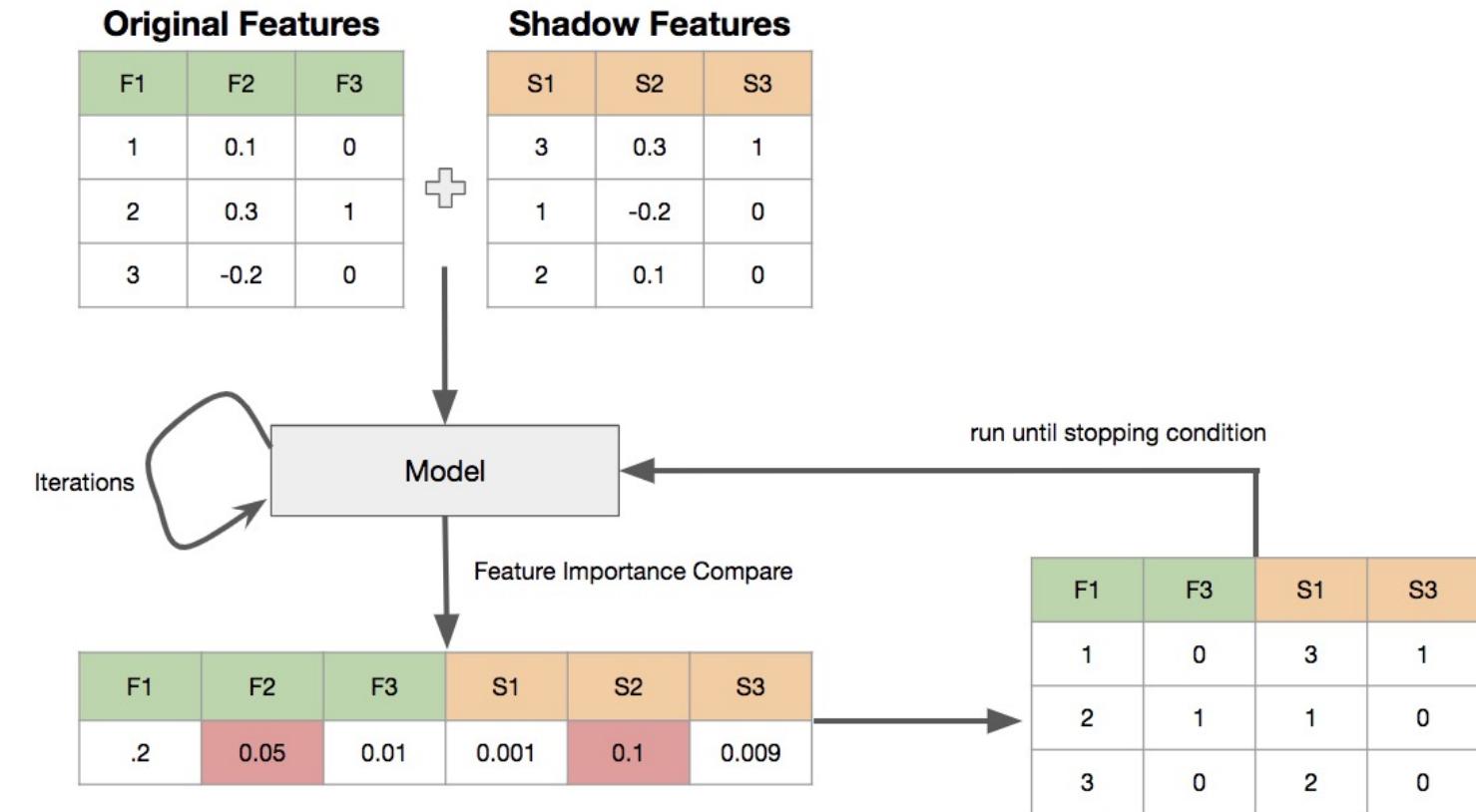
<https://towardsdatascience.com/understanding-random-forest-58381e0602d2>

Selección de variables con Random Forest y Boruta

Es una técnica de selección construida sobre el modelo de Random Forest.

No hace competir a las variables originales entre sí, sino con variables “shadow”.

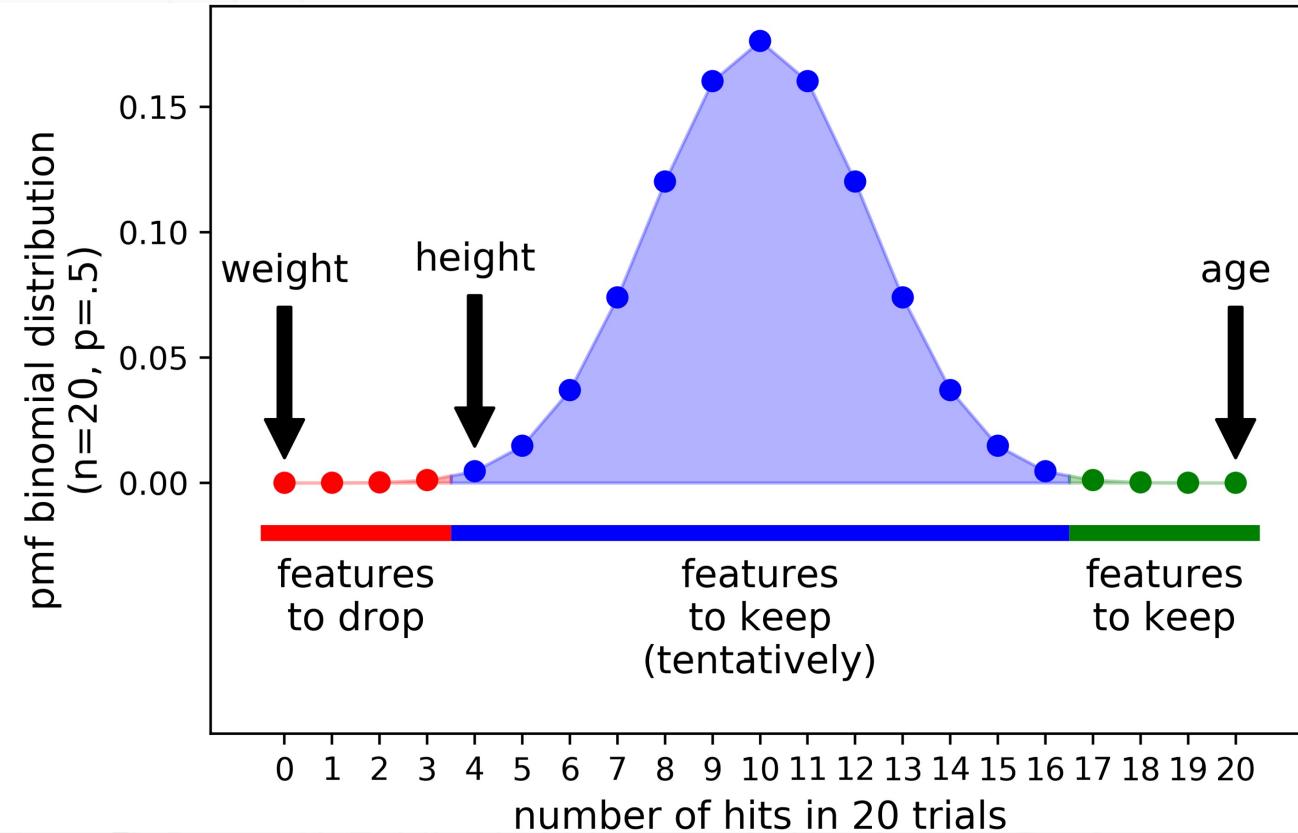
Estas variables ‘shadow’ se generan permutando los valores de cada variable.



Boruta

Si una variable tiene mejores resultados que su variable shadow, se considera como un hit.

Se repite este proceso una cantidad determinada de veces y se seleccionan las variables que tienen un mayor porcentaje de hits.



3. Optimización de hiperparámetros (hyperparameter tuning)



En los modelos de machine learning existen dos tipos de parámetros

Parámetros

Se **aprenden** al momento del entrenamiento
Por ejemplo: los betas de una regresión

```
lr = LinearRegression()  
lr.fit(X_train, y_train)
```

$$\hat{y} = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 \dots + \beta_j * x_j$$

Hiperparámetros

Se **especifican** al momento de instanciar el modelo
Por ejemplo: la profundidad de un árbol de decisión

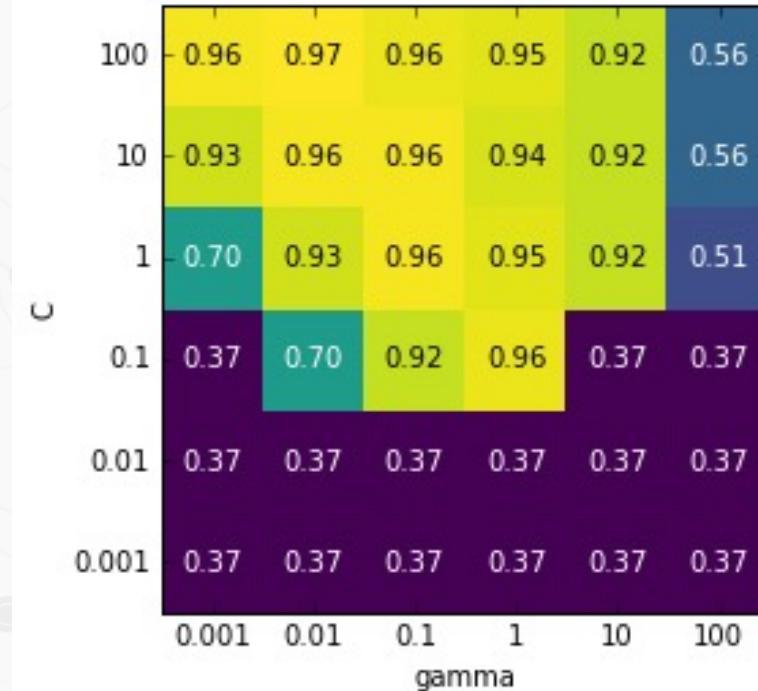
```
dt = DecisionTreeClassifier(max_depth = 5)  
dt.fit(X_train, y_train)
```

```
knn = KNeighborsClassifier(n_neighbors = 10)  
knn.fit(X_train, y_train)
```

Grid search

- Nos permite encontrar los mejores **hiperparámetros** para nuestro modelo mediante una **búsqueda exhaustiva**.
- Se entrega una lista de valores para distintos hiperparámetros del algoritmo.
- Se evalúa el modelo para cada combinación de hiperparámetros y se selecciona la que obtenga mejores valores en la métrica de evaluación

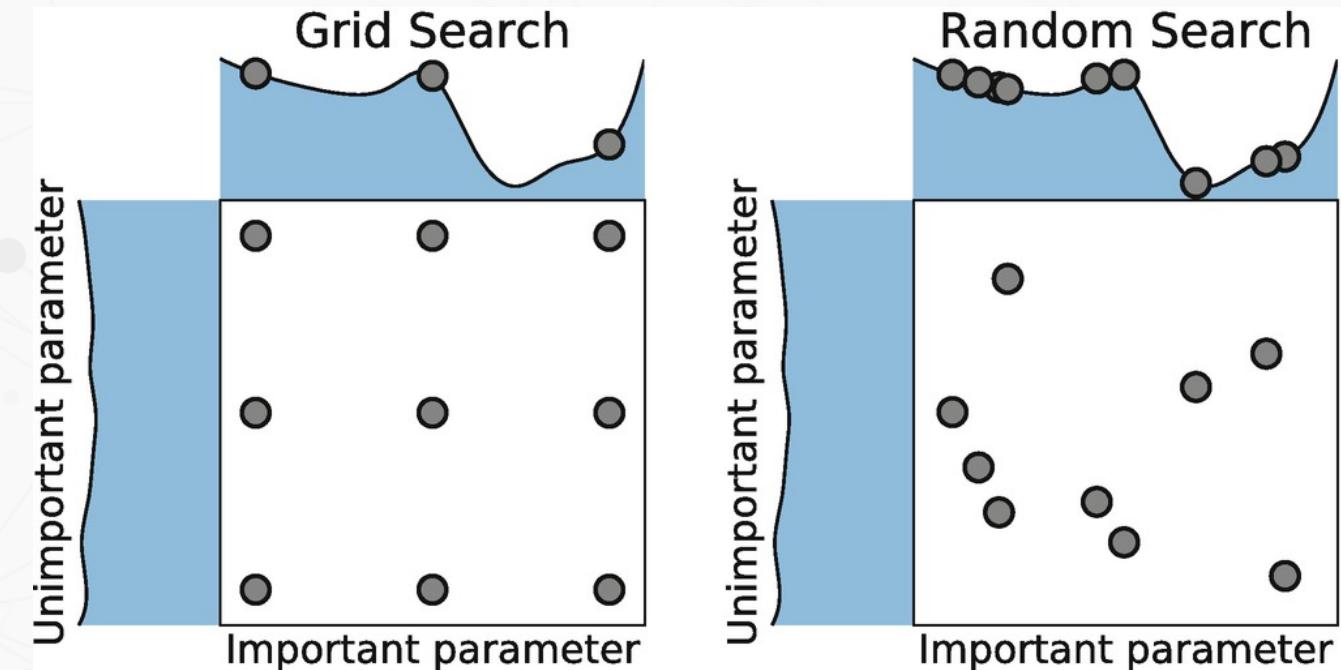
```
from sklearn.svm import SVC  
  
svc = SVC(kernel = 'rbf', gamma = 1, C = 20)
```



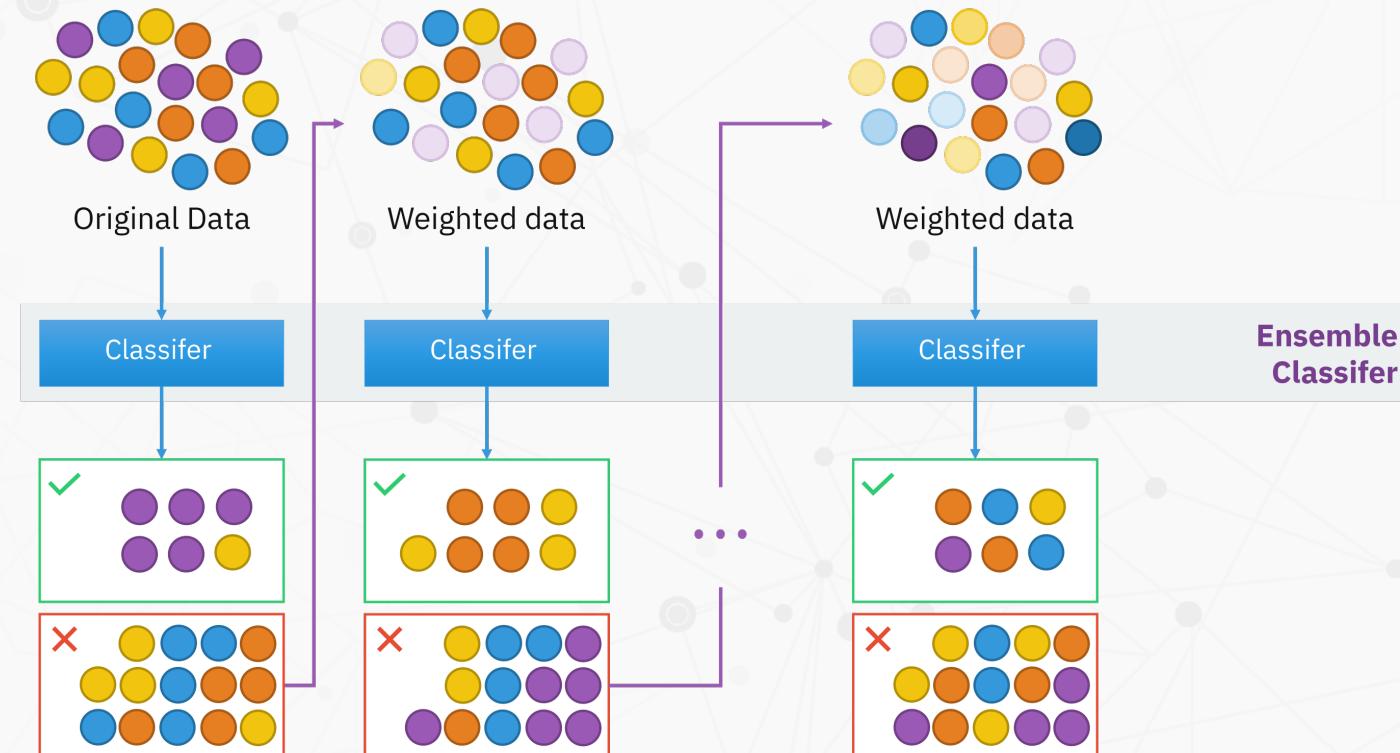
Otras alternativas

Random Search: intenta con combinaciones aleatorias dentro de un rango de valores para los hiperparámetros. No garantiza una solución óptima pero es más rápido

Optimización Bayesiana: Utiliza el teorema de Bayes para elegir secuencialmente la mejor combinación de parámetros.



4. Boosting

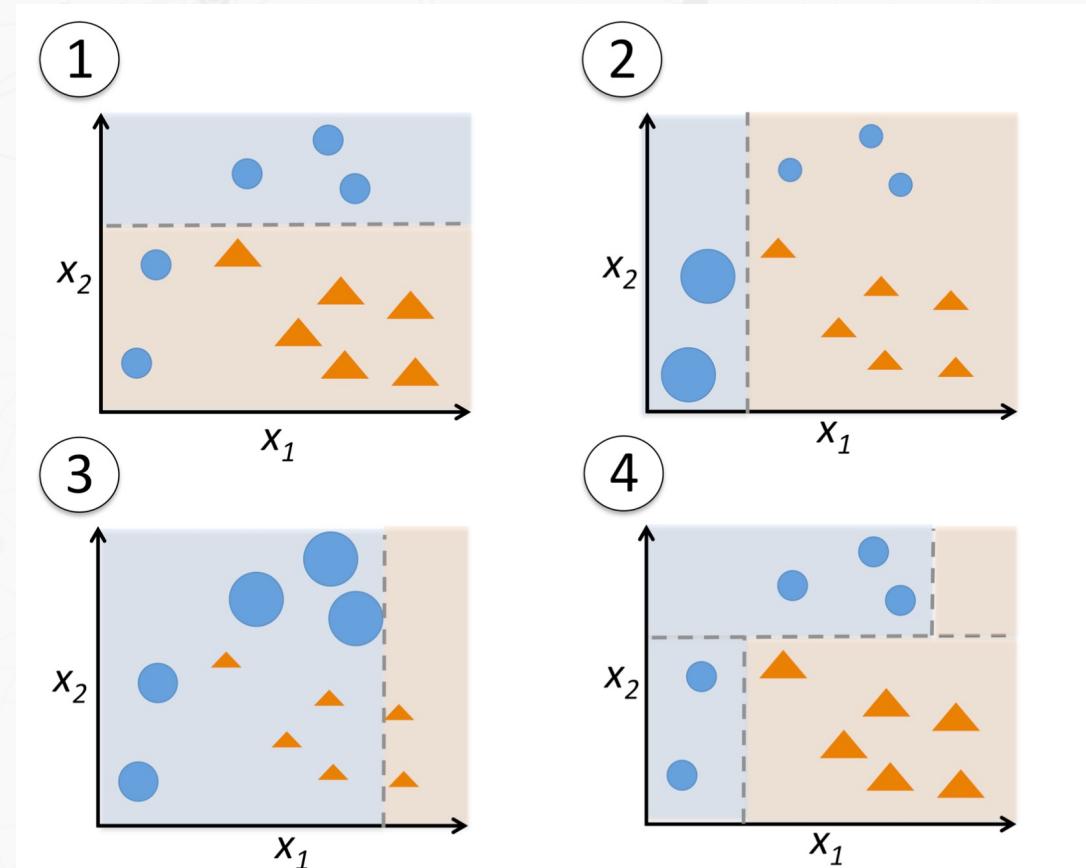


Boosting: Un clasificador que no aprende de sus errores está condenado a repetirlos

- ¿Se puede combinar modelos de aprendizaje débiles (weak learners) para crear un modelo de aprendizaje fuerte (strong learners)?
- Combina **clasificadores débiles** (*weak learners*)
- Se enfoca en los **datos difíciles de clasificar** correctamente
- Se usa **muestreo aleatorio pero sin reemplazo** (a diferencia de *bagging*)

Adaboost (Adaptive Boosting)

- Se denomina Adaboost porque en cada iteración **se adapta a los errores de los modelos anteriores.**



Gradient Boosting

- Gradient Boosting es una generalización del algoritmo AdaBoost que permite emplear **cualquier función de coste**, siempre que esta sea diferenciable.
- Para cada uno de ellos, el algoritmo de Gradient Boosting es ligeramente distinto, pero, para todos, la idea es la misma:
 - Dada una función de coste (por ejemplo, residuos cuadrados para regresión)
 - Utilizando un weak learner (por ejemplo, árboles)
 - El algoritmo trata de encontrar el modelo que minimiza la función de coste.

Gradient Boosting

- Dado que el objetivo de Gradient Boosting es ir minimizando los residuos iteración a iteración, es susceptible de overfitting.
- Una forma de evitar este problema es empleando un valor de regularización, también conocido como learning rate (λ), que limite la influencia de cada modelo en el conjunto del ensemble.
- Como consecuencia de esta regularización, se necesitan más modelos para formar el ensemble pero se consiguen mejores resultados.



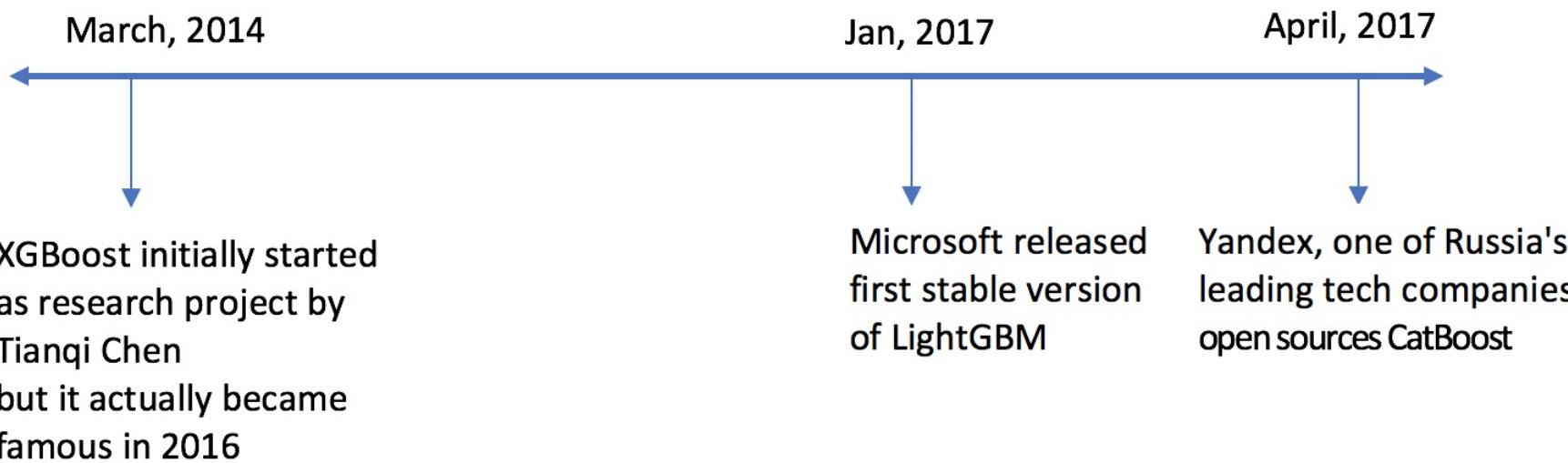
Gradiente descendiente



Es un algoritmo iterativo que se utiliza para minimizar una función según sus parámetros

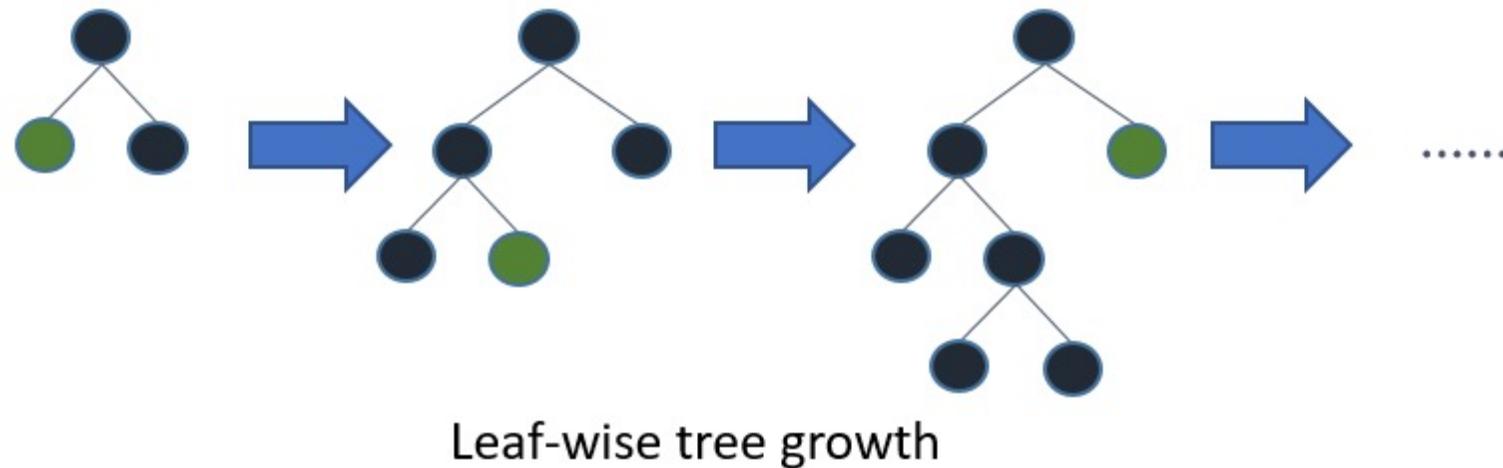
Se inician los parámetros con valores aleatorios y en cada iteración se actualizan para reducir el costo

XGBoost – LightGBM - CatBoost





XGBoost – LightGBM - CatBoost



Parámetros en algoritmos de boosting

Function	XGBoost	CatBoost	Light GBM
Important parameters which control overfitting	<ul style="list-style-type: none"> 1. learning_rate or eta – optimal values lie between 0.01-0.2 2. max_depth 3. min_child_weight: similar to min_child_leaf; default is 1 	<ul style="list-style-type: none"> 1. Learning_rate 2. Depth - value can be any integer up to 16. Recommended - [1 to 10] 3. No such feature like min_child_weight 4. L2-leaf-reg: L2 regularization coefficient. Used for leaf value calculation (any positive integer allowed) 	<ul style="list-style-type: none"> 1. learning_rate 2. max_depth: default is 20. Important to note that tree still grows leaf-wise. Hence it is important to tune num_leaves (number of leaves in a tree) which should be smaller than 2^{max_depth}. It is a very important parameter for LGBM 3. min_data_in_leaf: default=20, alias= min_data, min_child_samples
Parameters for categorical values	Not Available	<ul style="list-style-type: none"> 1. cat_features: It denotes the index of categorical features 2. one_hot_max_size: Use one-hot encoding for all features with number of different values less than or equal to the given parameter value (max – 255) 	<ul style="list-style-type: none"> 1. categorical_feature: specify the categorical features we want to use for training our model
Parameters for controlling speed	<ul style="list-style-type: none"> 1. colsample_bytree: subsample ratio of columns 2. subsample: subsample ratio of the training instance 3. n_estimators: maximum number of decision trees; high value can lead to overfitting 	<ul style="list-style-type: none"> 1. rsm: Random subspace method. The percentage of features to use at each split selection 2. No such parameter to subset data 3. iterations: maximum number of trees that can be built; high value can lead to overfitting 	<ul style="list-style-type: none"> 1. feature_fraction: fraction of features to be taken for each iteration 2. bagging_fraction: data to be used for each iteration and is generally used to speed up the training and avoid overfitting 3. num_iterations: number of boosting iterations to be performed; default=100

5. Validación cruzada (cross validation)

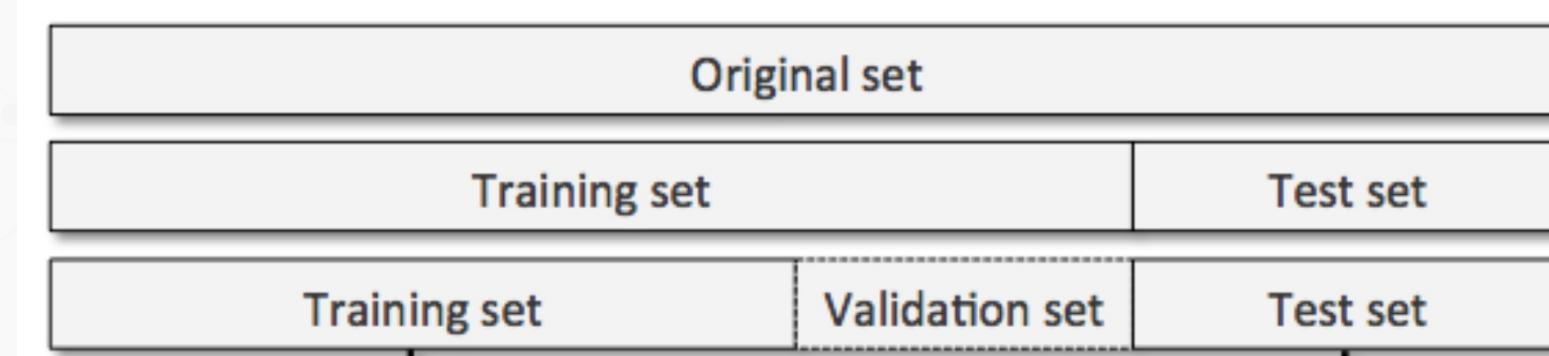
1. Holdout cross validation (conjunto de validación)

Original set

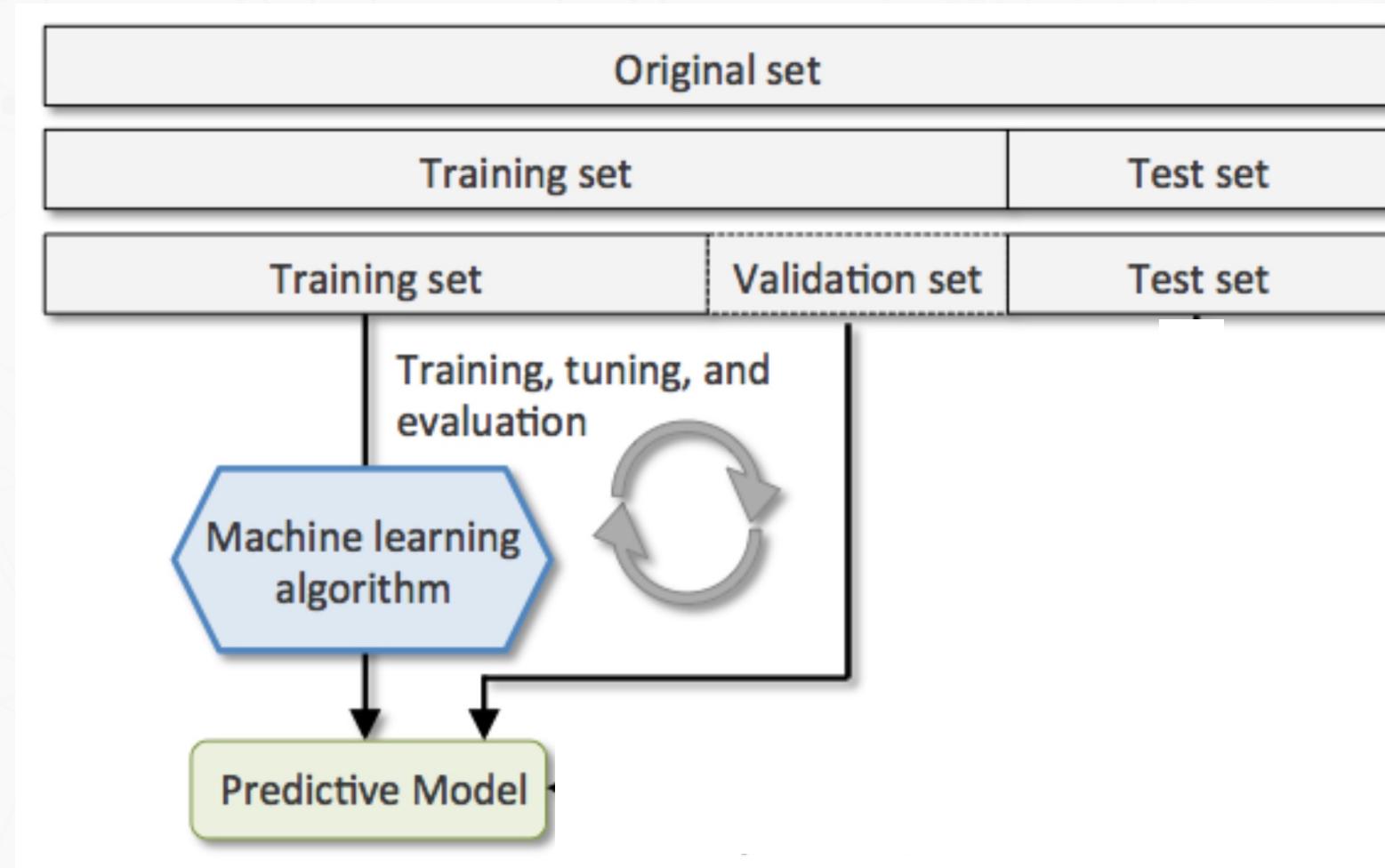
1. Holdout cross validation (conjunto de validación)



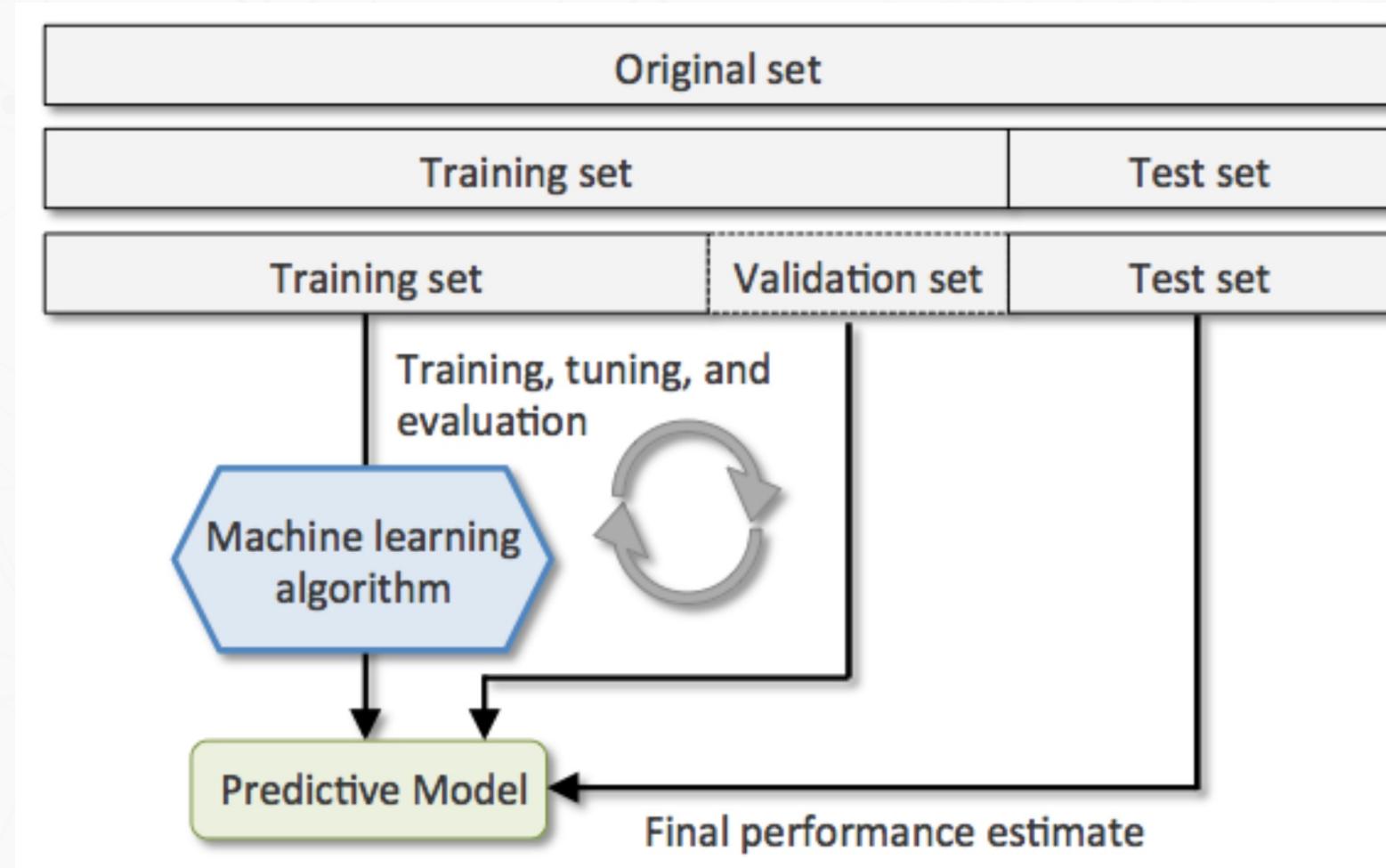
1. Holdout (conjunto de validación)



1. Holdout cross validation (conjunto de validación)



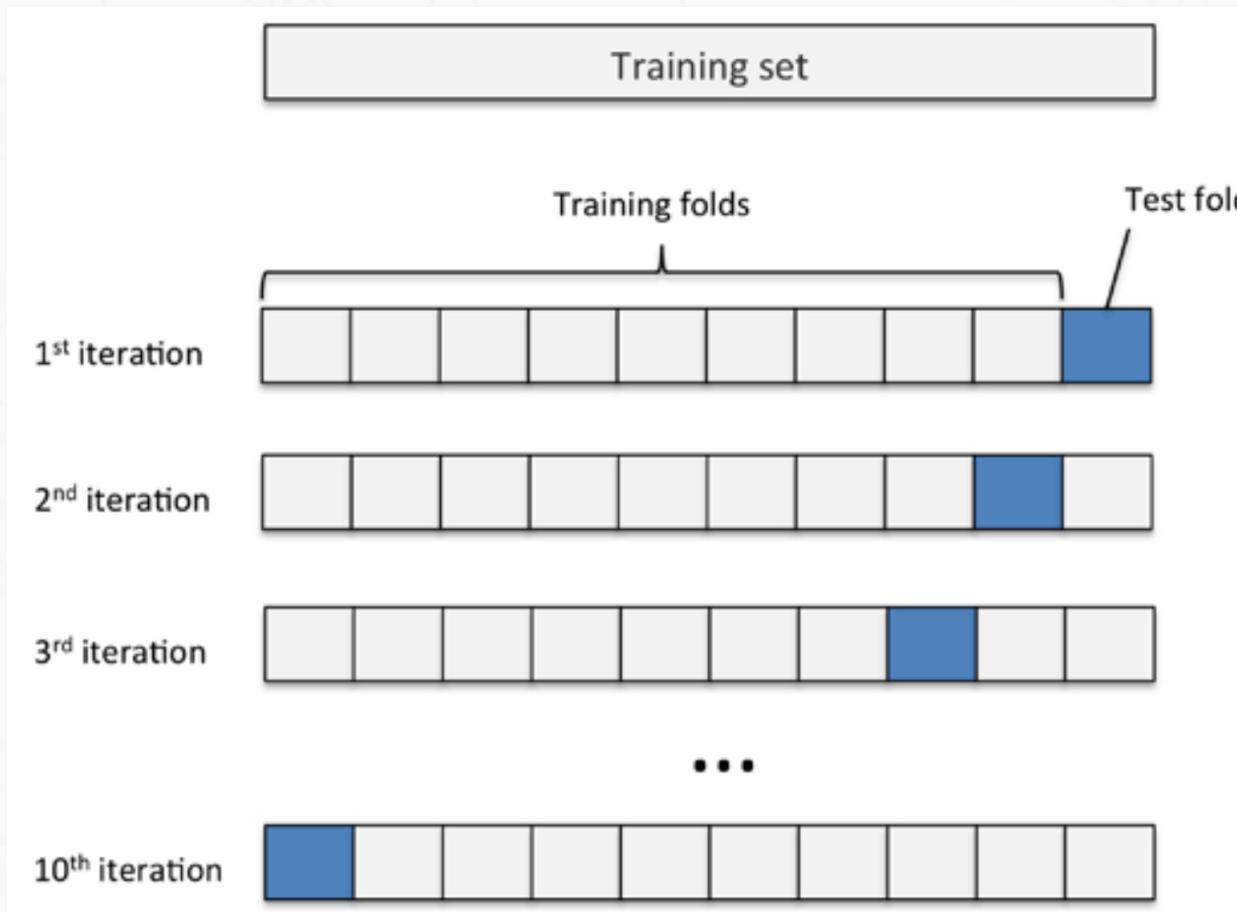
1. Holdout cross validation (conjunto de validación)



2. K-fold cross validation

Training set

2. K-fold cross validation

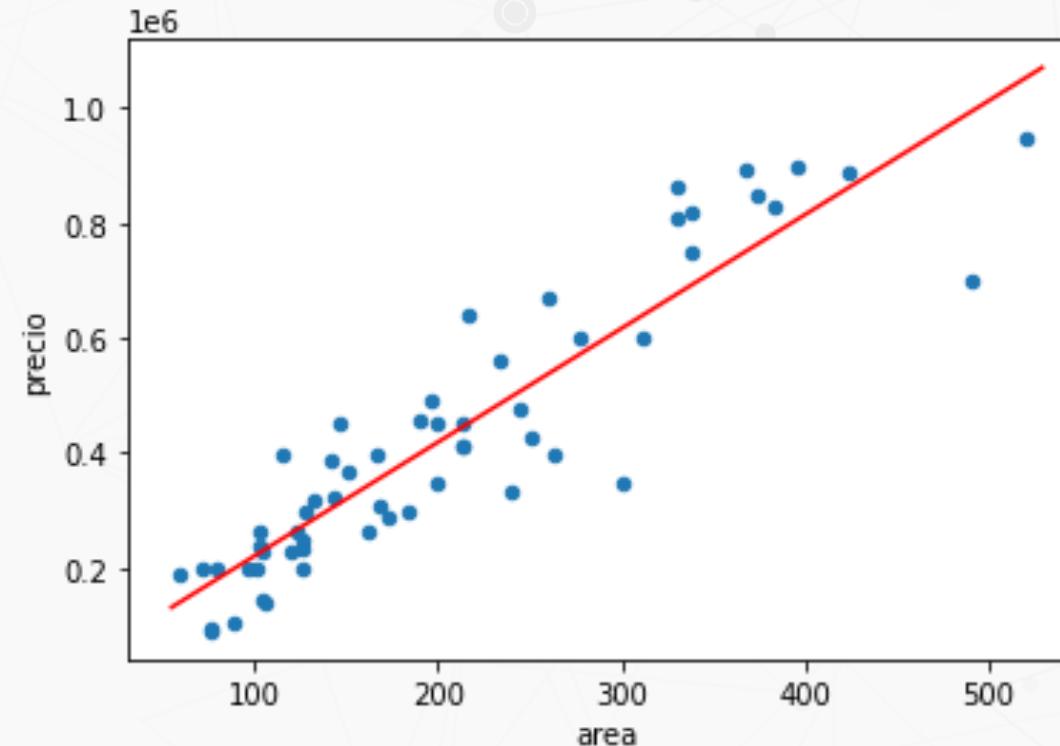


2. K-fold cross validation



6. Regularización

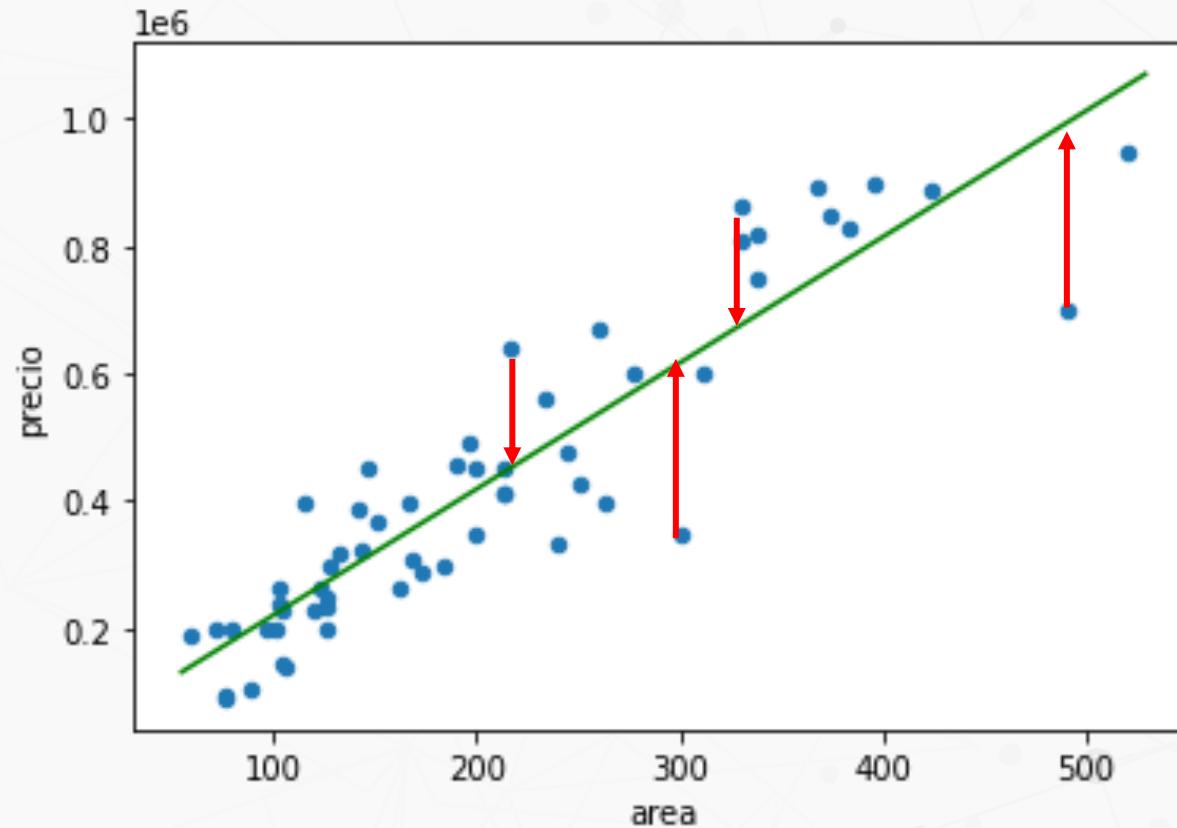
Regresión Lasso: Utilizar regularización sobre regresión logística



Regresión Lineal

- Es un método de aprendizaje supervisado
- Es un método de regresión porque estimamos un valor numérico real

Función de costo



$$\text{costo} = \sum_{1}^{n} (\hat{y} - y)^2$$

$\hat{y}: mx + b$ $y: \text{valor real}$

Utilizando un algoritmo de optimización, se eligen los mejores m y b para minimizar el costo

¿Y si deseo usar más de una variable predictoria?

Regresión Lineal Simple

$$\hat{y} = b_0 + b_1 * x$$

$$costo = \sum_{1}^{n} (\hat{y} - y)^2$$

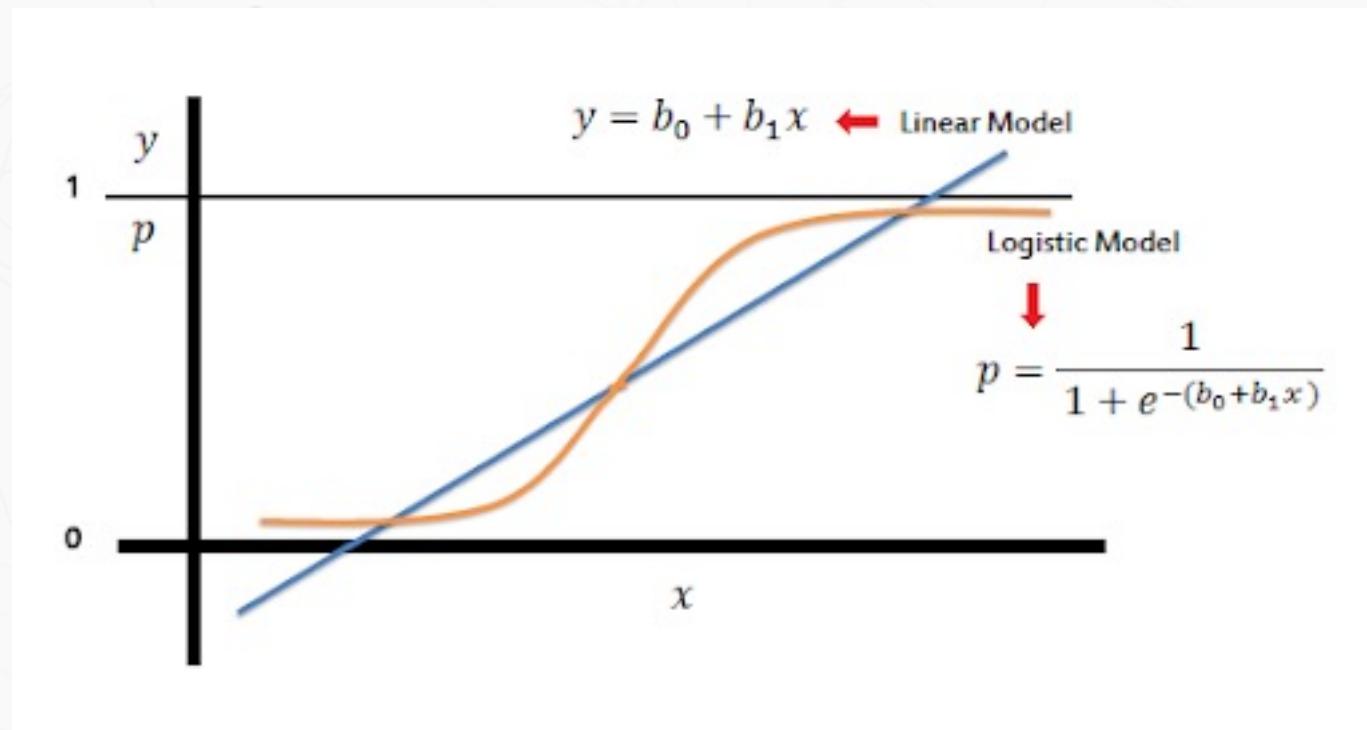
Regresión Lineal Múltiple

$$\hat{y} = b_0 + b_1 * x_1 + b_2 * x_2 + b_3 * x_3$$

\hat{y} : predicción
 y : valor real

Regresión Logística

- En este modelo la regresión se utiliza para predecir un target binario
- Para obtener la probabilidad de que ocurra un evento (target) se utilizan reglas de exponentes y logaritmos



Regresión Logística

- En este modelo la regresión se utiliza para predecir un target binario
- Para obtener la probabilidad de que ocurra un evento (target) se utilizan reglas de exponentes y logaritmos

$$\hat{y} = \beta_1 * x_1 + \beta_2 * x_2 + \dots \beta_j = \log_b \left(\frac{p}{1-p} \right)$$

p = probabilidad de evento

$$p = \frac{1}{1 + e^{-(\beta_1 * x_1 + \beta_2 * x_2 + \dots \beta_j)}}$$

Regularización: Regresión Lasso

- Regularización L1
- Permite reducir a valores cercanos a cero los coeficientes de las variables menos relevantes
- Se utiliza como técnica para la reducción de dimensionalidad
- El factor lambda controla el nivel de regularización

$$\text{costo} = \sum_1^n (\hat{y} - y)^2 + \lambda * \sum_1^j |\beta|$$
$$\hat{y} = \beta_1 * x_1 + \beta_2 * x_2 + \dots \beta_j$$

Regularización: Regresión Ridge

- Regularización L2
- Es útil cuando existe colinearidad entre las variables utilizadas para el entrenamiento
- El factor λ (lambda) sirve para controlar la intensidad de la regularización

$$J(\beta) = \frac{1}{2} \sum_{1}^n (\hat{y} - y)^2 + \lambda * \sum_{1}^j \beta^2$$

$$\hat{y} = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 \dots + \beta_j * x_j$$

Lecturas recomendadas

- Python Machine Learning – Capítulo 4: Building good training datasets
 - Sección: L1 and L2 regularization as penalties against model complexity
 - Sección: A geometric interpretation of L2 regularization
 - Sección: Sparse solutions with L1 regularization
- Hands on Machine Learning – Capítulo 7: Ensemble learning and Random Forest
- Python Machine Learning - Capítulo 6: Learning best practices for model evaluation and hyperparameter tuning
- Python Machine Learning – Capítulo 7: Combining different models for Ensemble Learning
- Making Predictive Models Robust: Holdout vs Cross-Validation <https://www.kdnuggets.com/2017/08/dataiku-predictive-model-holdout-cross-validation.html>
- Random search for hyperparameter optimization
<https://jmlr.csail.mit.edu/papers/volume13/bergstra12a/bergstra12a.pdf>

Enlaces sobre boosting

- Parámetros de XGBoost: <https://xgboost.readthedocs.io/en/latest/parameter.html>
- Parámetros de LightGBM: <https://lightgbm.readthedocs.io/en/latest/Parameters-Tuning.html>
- Catboost vs LightGBM vs XGBoost: <https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db>
- XGBoost, LightGBM or CatBoost — which boosting algorithm should I use?: <https://medium.com/riskified-technology/xgboost-lightgbm-or-catboost-which-boosting-algorithm-should-i-use-e7fda7bb36bc>
- What makes LightGBM lightning fast?: <https://towardsdatascience.com/what-makes-lightgbm-lightning-fast-a27cf0d9785e>

¡Gracias!