Final: AdaBoost

Dey, Patrick
Lombardi, Matías
Vázquez, Ignacio

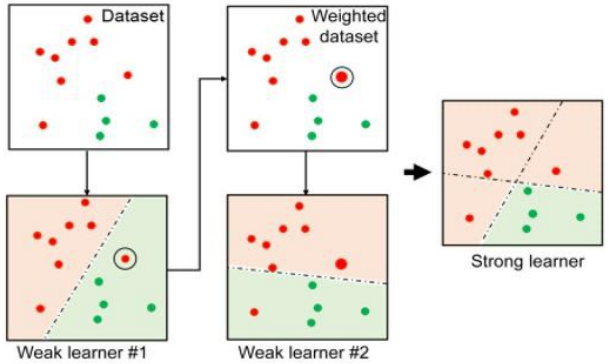
Introducción: Boosting

- Técnica que tiene como fin crear un modelo fuerte de predicción a partir de la combinación de un conjunto de modelos débiles (modelo cuya tasa de error es levemente superior a la de un modelo aleatorio)
- Vamos a llamar a los modelos débiles como estimadores
- Funciona de la siguiente manera:
    1. Construimos un estimador
    2. Generamos un nuevo estimador que ajusta mejor lo que el anterior no pudo
    3. Repetimos hasta tener la cantidad de estimadores deseada
    4. Tomando todos los estimadores generados podemos realizar una predicción (a la RandomForest)

Resumen: ideas detrás de AdaBoost

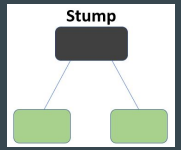
- Es un método basado en la técnica de boosting.
- Combinar estimadores débiles
  - En general se usa un Árbol de decisión con profundidad 1
- Algunos estimadores tienen más peso sobre la decisión final que otros
  - Al contrario que en RandomForest, donde todos los votos valen lo mismo
- Cada estimador se construye en base a los errores del anterior
  - Es decir, es secuencial, al contrario de RandomForest, en donde se construyen los árboles de forma independiente

AdaBoost


Dataset
Weighted dataset
Strong learner
Weak learner #1
Weak learner #2

¿Qué es Adaptive Boost?

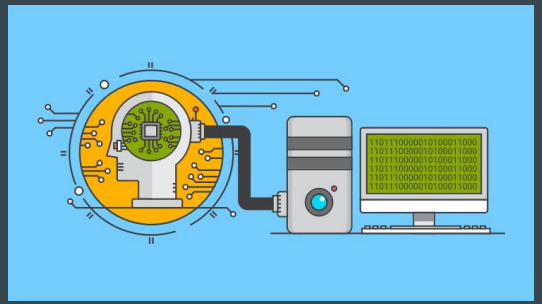
- El tipo de estimador más utilizado es el Decision Stump. Es un árbol con profundidad 1 (se clasifica en base a 1 solo atributo!! -> estimador débil)



- Cada ejemplo tiene un peso asociado en base a qué tan importante es clasificarlo bien (que inicialmente es el mismo)
- Estos pesos se van actualizando luego de la construcción de cada estimador *
- La importancia del estimador sobre la decisión final se basa en los errores cometidos al clasificar los ejemplos (sumado a sus pesos asociados).
- El siguiente estimador va a tener en cuenta estos errores!!

Aprendizaje

- Cada estimador se va a construir teniendo en cuenta los errores que cometió el anterior
- Combina estimadores débiles, en donde los errores de un estimador influencian la creación del segundo
- Para esto se usa la importancia que tiene clasificar bien un ejemplo de entrada
- Diferencia radical con RandomForest en donde los clasificadores eran independientes, en este algoritmo se construyen de forma secuencial!!



Algoritmo (estimador: Árboles de decisión + Índice de Gini)

Sea $X_i = (x_{i1}, x_{i2}, x_{i3}, \dots, x_{in}, y_i)$ el ejemplo i del dataset, donde x_{ij} es la variable j de la misma. y_i es el valor de la etiqueta

1. Asignamos un peso inicial igualitario a todas las instancias (la suma siempre debe dar 1)

$$w(x_i, y) = \frac{1}{N},\ i = 1, 2, \dots, n$$

2. Creamos un nodo comparando el índice de Gini/Entropía. Las hojas serán cada una de los valores posibles (recordar categorización!). En el caso de índice de Gini, nos habla de la "impureza" (clasificaciones mixtas), por lo que se toma el de menor valor

Algoritmo: performance del estimador

3. Evaluamos la performance del estimador. Supongamos el siguiente dataset en donde la entrada roja fue mal clasificada

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

Algoritmo: performance del estimador

En base a los errores cometidos por el estimador, calculamos el error total, que es la suma de los pesos asociados a las entradas mal clasificadas, en este caso sería ⅛.

Este valor siempre estará entre 0 y 1 (ya que la suma de los pesos debe ser 1)

$$TE = \sum_i w_i \text{ , donde i es el índice de la entrada mal clasificada}$$

En base a este valor vamos a calcular la importancia del estimador. Con este valor se pondera su voto a la hora de clasificar (y también influencia al siguiente clasificador). Llamemos α a la importancia.

$$\alpha = \eta * log(\frac{1-TE}{TE})$$

Como este valor no existe para TE = 0/1, en la práctica se le agrega un término al error. Notar que puede tomar valores negativos! (puede invertirse su decisión).

Algoritmo: reasignación a los pesos de ejemplos

4. Ahora debemos actualizar los pesos de las entradas. Buscamos incrementar el peso de las entradas mal clasificadas y disminuir el resto

   Aquellas entradas que fueron mal clasificadas aumentan su peso

$$w_i' = w_i * e^{\alpha}$$

   Aquellas entradas que fueron bien clasificadas reducen su peso

$$w_i' = w_i * e^{-\alpha}$$

   Una vez actualizados todos los pesos, debemos normalizarlos ya que la suma de los mismos no da 1

Algoritmo: creación del siguiente estimador

¿Cómo tenemos en cuenta los errores cometidos? Podemos ponderar los pesos a la hora de calcular Índice de Gini/Entropía. Sin embargo, existe otra forma en donde se crea un un nuevo dataset, *sampleando* en base a los pesos finales del estimador.

5. Creamos un dataset vacío del mismo tamaño que el original y lo vamos llenando en función de los pesos recién calculados. *Sampleamos* en base a esa distribución -> los ejemplos mal clasificados *deberían* aparecer más veces por lo que se penaliza más clasificarlos erróneamente.
    a. Tomamos un valor al azar entre 0 y 1
    b. Vemos a qué entrada pertenece
    c. Agregamos dicha entrada al nuevo dataset (pueden aparecer entradas repetidas)
    d. Repetimos hasta llenar el dataset

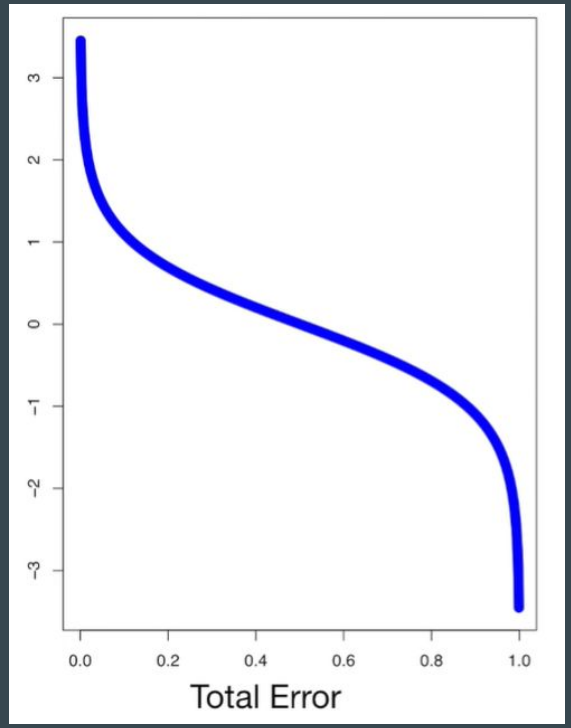
Weight	
0.083	**0 to 0.083**
0.083	**0.084 to 0.166**
0.25	**0.167 to 0.416**
0.083	**0.417 to 0.499**
0.083	**0.500 to 0.582**
0.25	**0.583 to 0.832**
0.083	**0.833 to 0.915**
0.083	**0.916 to 1**

6. Repetimos todos los pasos del 1 al 3 hasta tener la cantidad de estimadores deseada -> notar que los pesos vuelven a ser uniformes

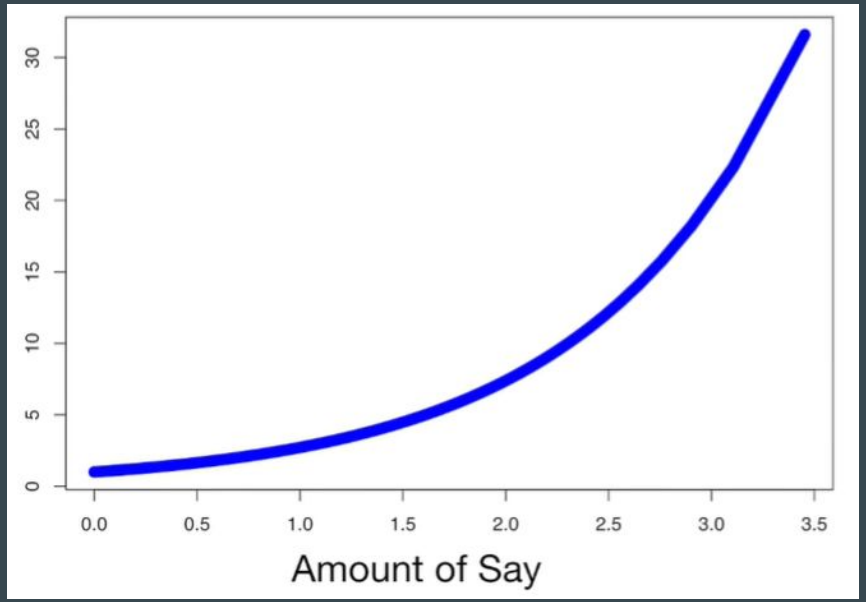
Notas

- En este caso mostramos una forma de decidir cuál será el nodo del Stump usando el índice de Gini, también se puede hacer con entropía como en el TP2
- En algunas implementaciones no se altera el peso de los ejemplos bien clasificados
- Hay otras formas de construir el siguiente árbol, no hace falta crear un nuevo dataset
  - Se podría ponderar los pesos asociados a cada ejemplo para decidir la variable correspondiente al nodo
- El algoritmo que presentamos es SAMME
  - Existe otra implementación llamada SAMME.R
    - Devuelve la probabilidad de que pertenezca a determinada clase, en vez de ser binario
    - En teoría converge más rápido

¿Cómo influye la importancia de un estimador?



Importancia vs TE



Nuevo peso vs Importancia
(ejemplo mal clasificado)

Predicción

Finalmente, una vez que tenemos todos los estimadores, vamos a realizar predicciones

1. Presentamos la instancia X_i a todos los estimadores generados
2. Dividimos los estimadores en 2 grupos según cómo clasificaron a X_i
3. Ponderamos la importancia de cada estimador y los sumamos por clase

$$\alpha_k = \sum_i \alpha_{i,k}$$

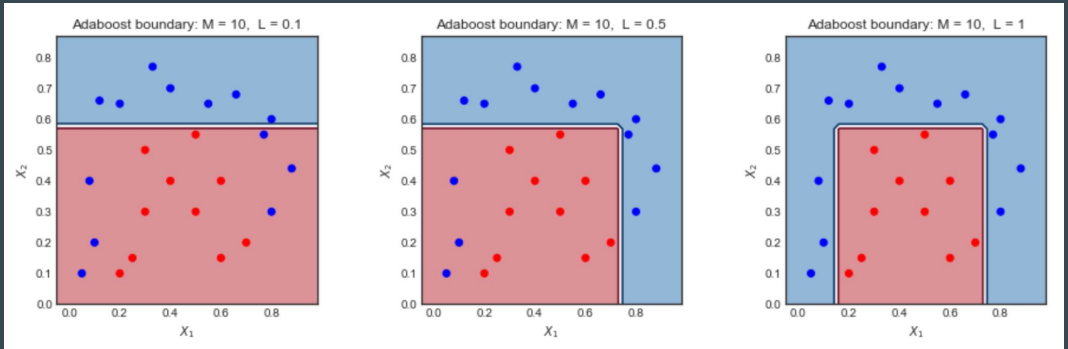
4. El grupo con mayor importancia determina la clase

Parámetros importantes

- Tipo de estimador: El tipo de estimador que utiliza el modelo internamente
  - Decision Stumps
  - SVM
- Cantidad de estimadores
  - A mayor cantidad de estimadores, mayor cantidad de iteraciones
  - Mayor cantidad de estimadores permite representar límites de decisión más complejos
- Learning rate: disminuye la contribución de cada estimador al siguiente
  - Menor learning rate presenta menor variación en los pesos asociados a los ejemplos -> menor diferencia entre los límites de decisión de los estimadores ya que se penaliza menos (además que cada estimador contribuye menos!)

N° estimadores vs Learning Rate

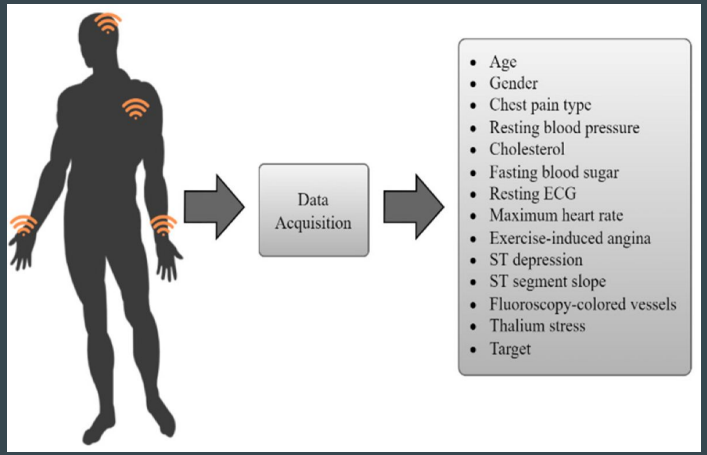
- Existe un *trade off* entre ambos parámetros
  - Si tomo un learning rate muy chico, debería incrementar la cantidad de estimadores, ya que se decrementa la contribución de un estimador al otro
- Poca cantidad de estimadores o un learning rate chico lleva a límites de decisión simples
- Gran cantidad de estimadores y un learning rate alto lleva a límites de decisión más complejos (pero puede causar *overfitting*!!)



Enfermedades coronarias

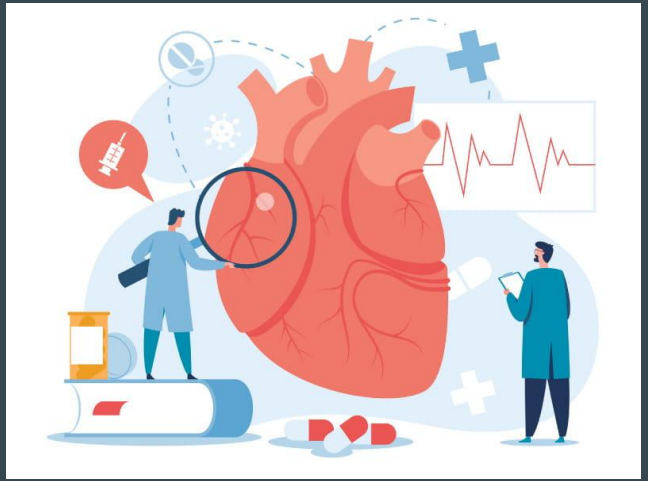
Objetivo

- Intentar predecir si un paciente sufre una enfermedad coronaria
- Análisis de un dataset que contenga estudios hechos a pacientes
- Análisis de los parámetros óptimos
- Obtener métricas sobre el modelo construido
- Comparar con otro clasificador: Random Forest



Laboratorio Cleveland ->

- Age: variable discreta
- Sex: M/F
- ChestPain: tipo de dolor en el pecho (4 categorías)
- RestBP: presión arterial en reposo (variable continua)
- Cholesterol: variable discreta
- Fbs: azúcar en sangre en ayunas >120 mg/dl (2 categorías)
- RestECG: electrocardiograma en reposo (3 categorías)
- MaxHR: variable discreta
- ExAng: angina por ejercicio (dos categorías)
- Oldpeak: prueba de esfuerzo
- Slope: pendiente de la prueba de esfuerzo
- Ca: cantidad de vasos principales (4 categorías)
- Thal: prueba de Thalium (3 categorías)
- HDisease: tiene o no la enfermedad

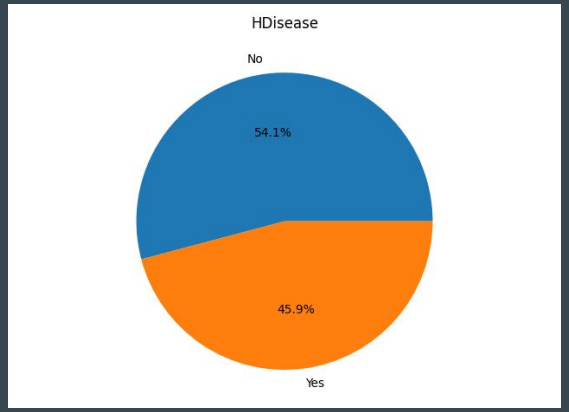


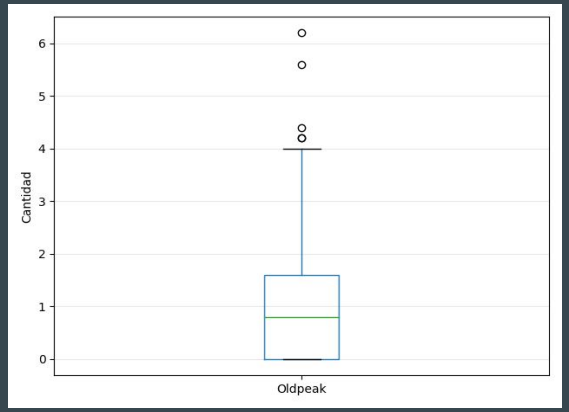
Análisis de las variables

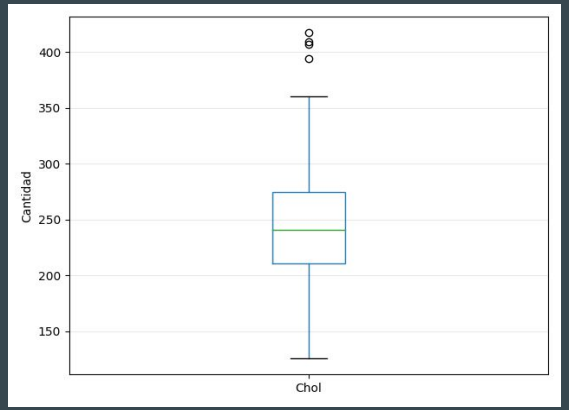
(Antes del preprocesamiento)

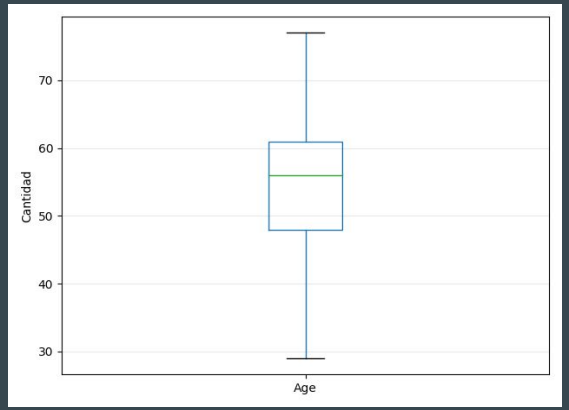
Caracterización de los datos

- Targets balanceados
- Más del 25% de las entradas de Oldpeak se encuentran en 0
- No hay outliers en Age



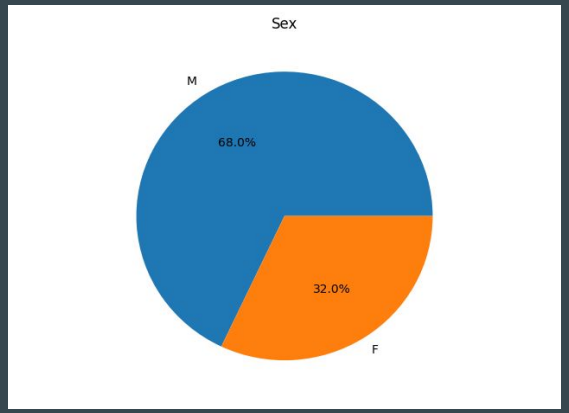


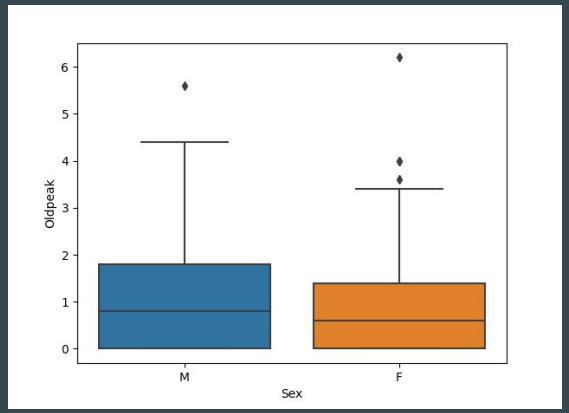


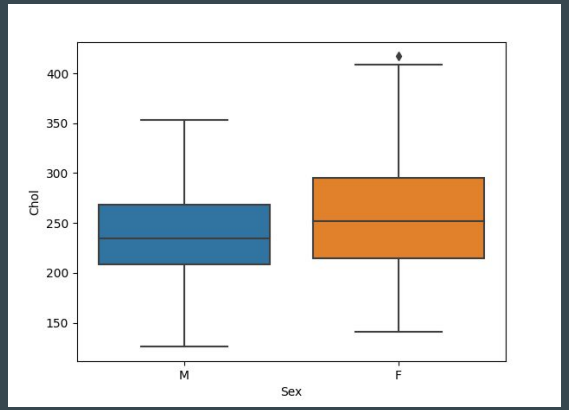


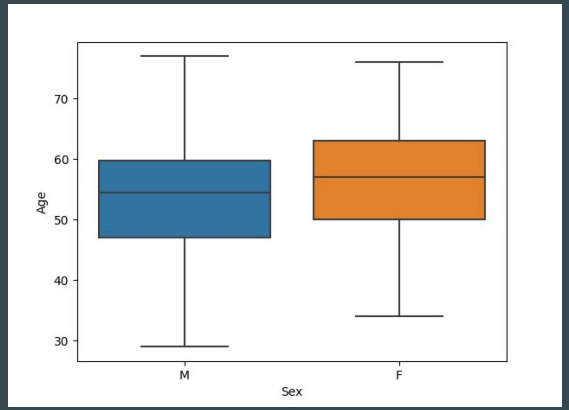
Caracterización de los datos

- Mayor cantidad de hombres
- La variable Oldpeak tiene el mismo problema sin importar el género
- Las mujeres testeadas tienen mayor edad que los hombres









Preprocesamiento

- Al igual que en el TP2, como vamos a utilizar árboles como estimadores, vamos a categorizar las variables continuas de acuerdo a sus cuantiles (cuartiles para todos excepto Oldpeak)
- Pasamos todas las variables no numéricas (M/F en sexo, tipo de dolor de pecho) a una categorización:

```
{
    "Fbs": { "<=120": 0, ">120": 1 },
    "Sex": { "F": 0, "M": 1 },
    "ChestPain": { "typical": 0, "asymptomatic": 1, "nonanginal": 2, "nontypical": 3 },
    "RestECG": { "normal": 0, "abnormal": 1 },
    "ExAng": { "No": 0, "Yes": 1 },
    "Slope": { "down": 0, "level": 1, "up": 2 },
    "Thal": { "normal": 0, "fixed": 1, "reversable": 2 },
    "HDisease": { "No": 0, "Yes": 1 }
}
```

	column	q1	q2	q3	q4
1	Age	48.0	56.0	61.0	77.0
2	RestBP	120.0	130.0	140.0	200.0
3	Chol	211.0	241.0	274.5	417.0
4	MaxHR	133.5	153.0	166.0	202.0
5	Oldpeak	0.0	0.8	1.6	6.2

- El dataset no tiene repetidos. Tampoco valores nulos.

Métricas

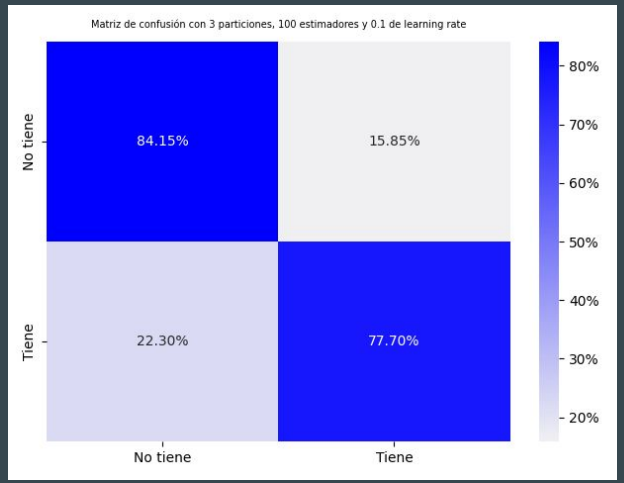
Métricas Adaboost

- Shuffle previo del dataset
- Cross-validation, con particiones 70/30, 80/20, 90/10
- Se presenta la precisión promedio y el desvío estándar de todas las corridas
- Para la matriz de confusión, se toma la suma de todas las iteraciones
- Cantidad de estimadores [50, 100, 150]
- Learning rate [0.1, 0.5, 0.7]

Resultados: variando N° particiones

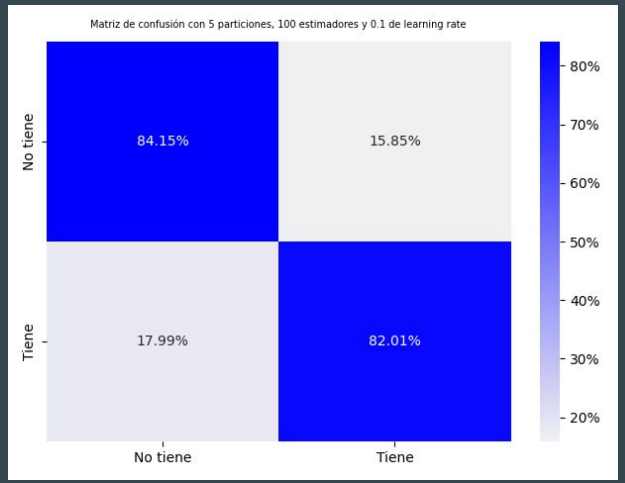
N° estimadores = 100 , Learning Rate = 0.1

70/30



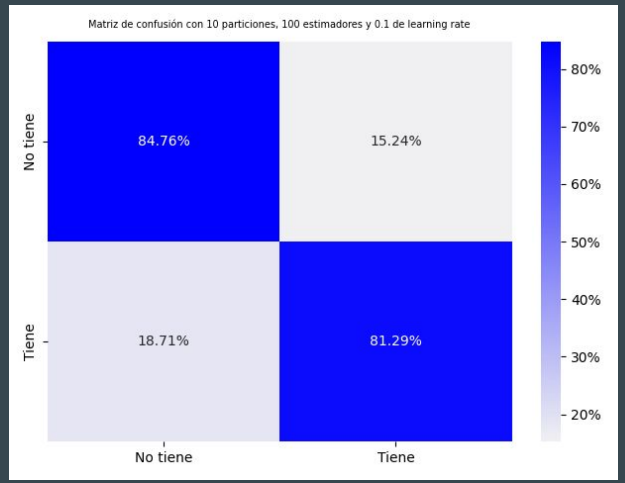
Precisión media	0.81
Desvío standard	± 0.04

80/20



Precisión media	0.82
Desvío standard	± 0.04

90/10

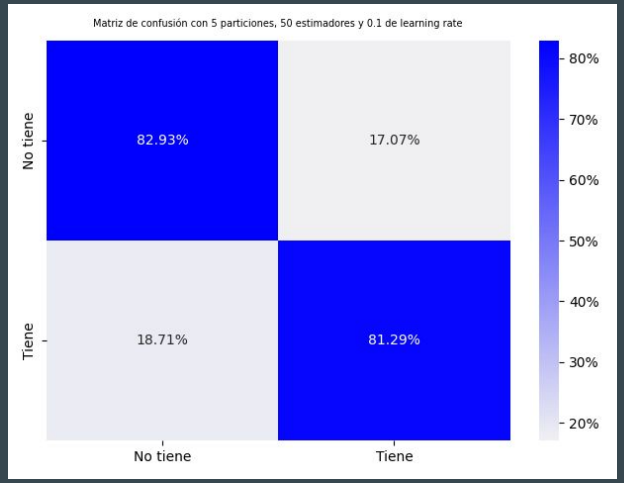


Precisión media	0.82
Desvío standard	± 0.08

Resultados: variando N° estimadores

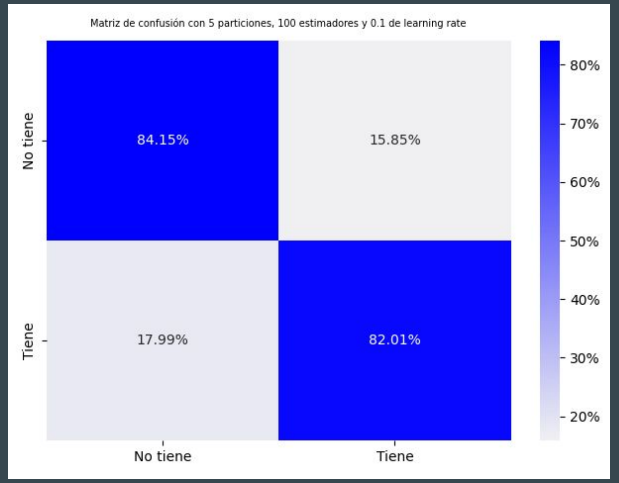
80/20, N° particiones = 5 , Learning Rate = 0.1

50



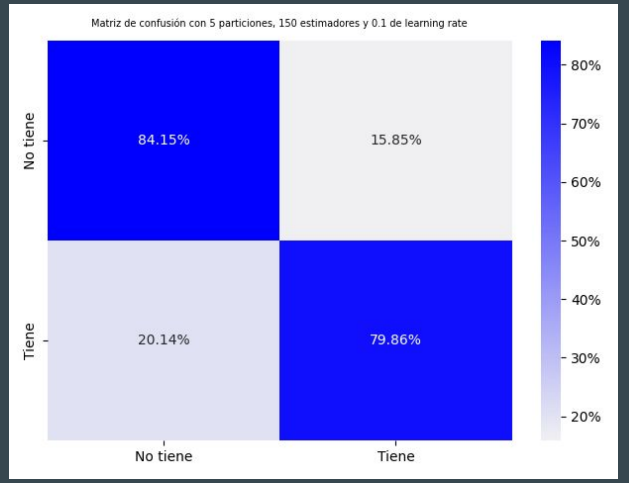
Precisión media	0.8
Desvío standard	± 0.05

100



Precisión media	0.82
Desvío standard	± 0.04

150

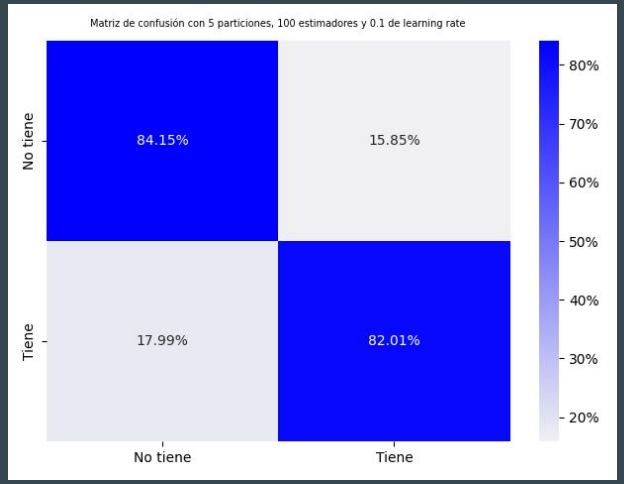


Precisión media	0.81
Desvío standard	± 0.06

Resultados: variando learning rate

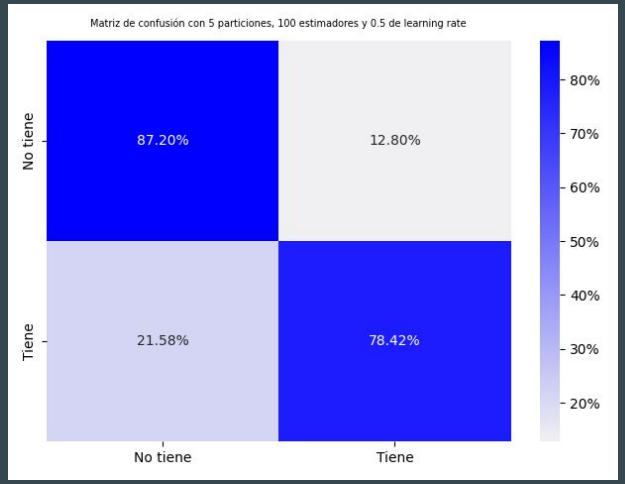
80/20, N° particiones = 5 , N° estimadores = 100

0.1



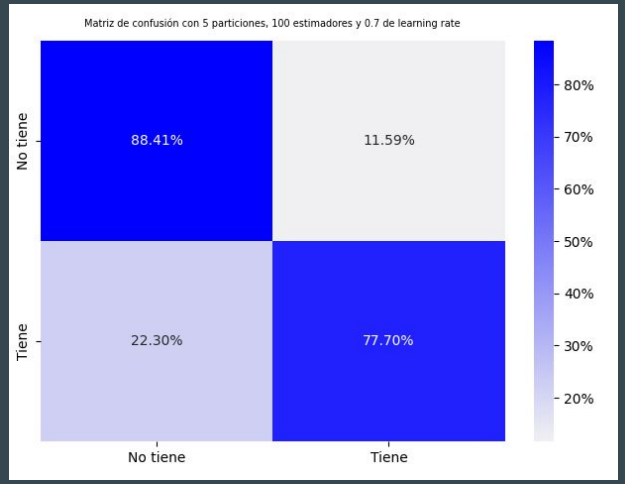
Precisión media	0.82
Desvío standard	± 0.04

0.5



Precisión media	0.84
Desvío standard	± 0.08

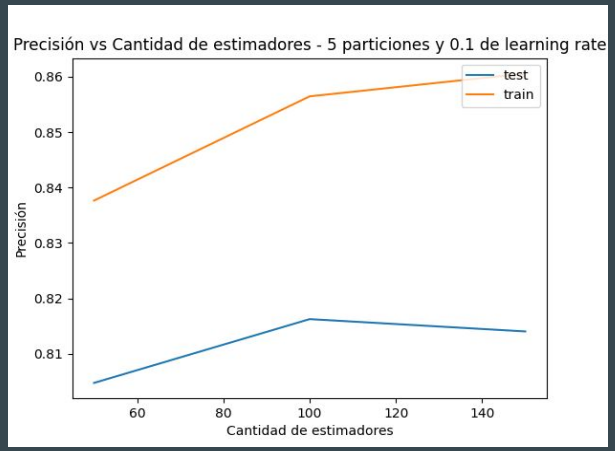
0.7



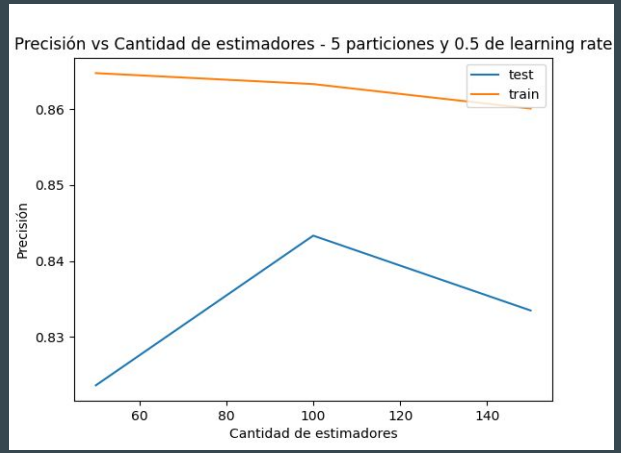
Precisión media	0.86
Desvío standard	± 0.05

Resultados: precisión vs N° estimadores

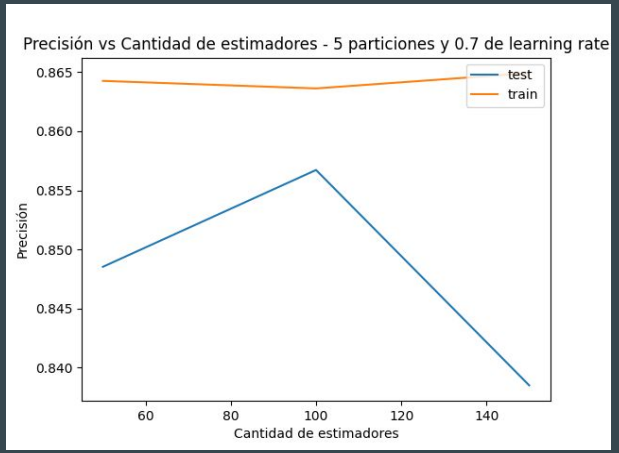
80/20, N° particiones = 5



η=0.1



η=0.5



η=0.7

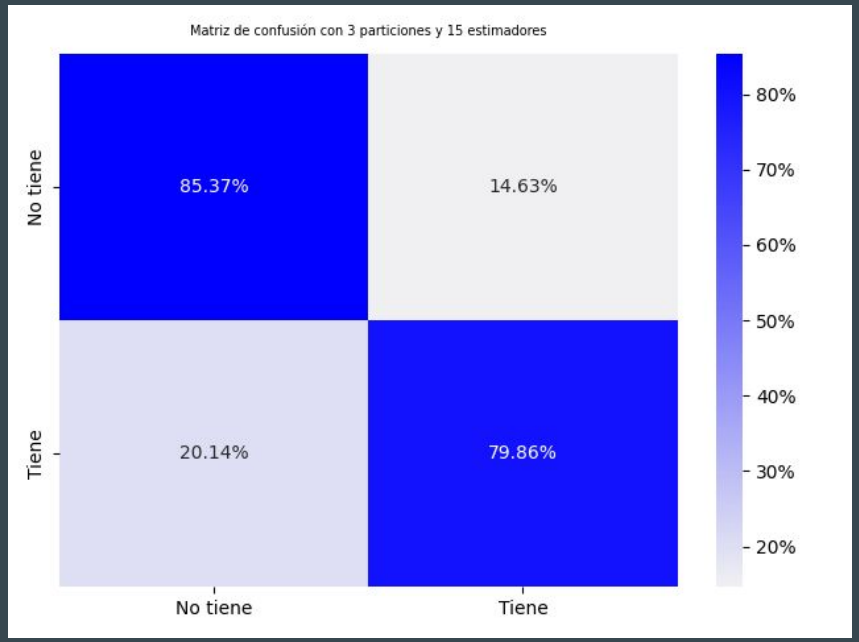
Comparación con RandomForest

Comparación con RandomForest

- Shuffle previo del dataset
- Cross-validation, con particiones 70/30, 80/20, 90/10
- Cantidad de árboles [5, 10, 15]
- Se presenta la mejor matriz de confusión para todas las corridas, con la precisión promedio y desvío estándar

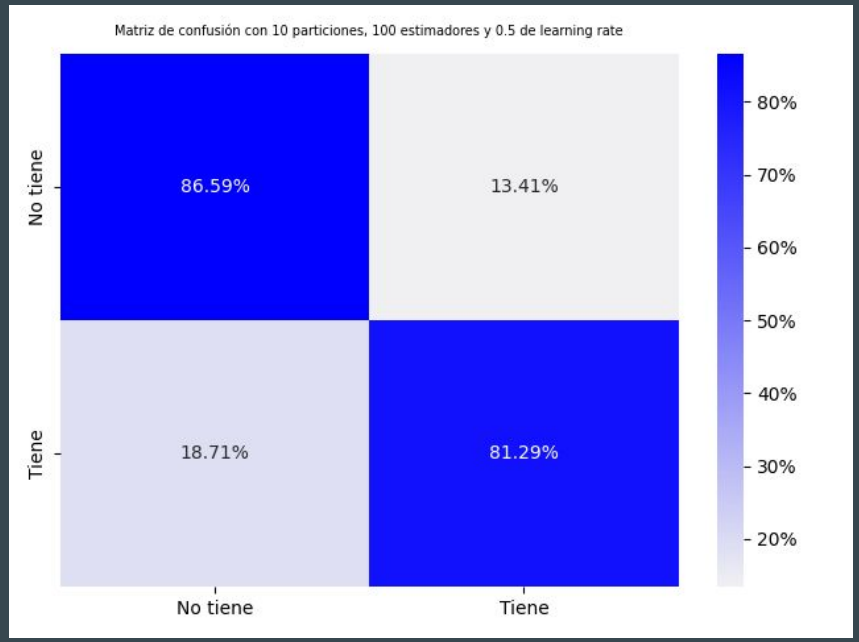
Mejor matriz de confusión

RandomForest



Precisión media	0.82
Desvío standard	± 0.05

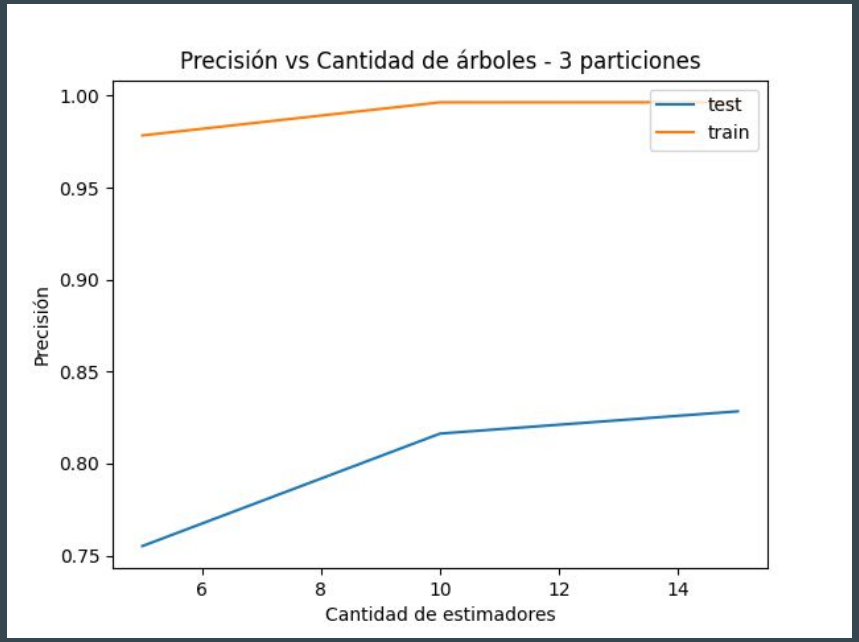
Adaboost



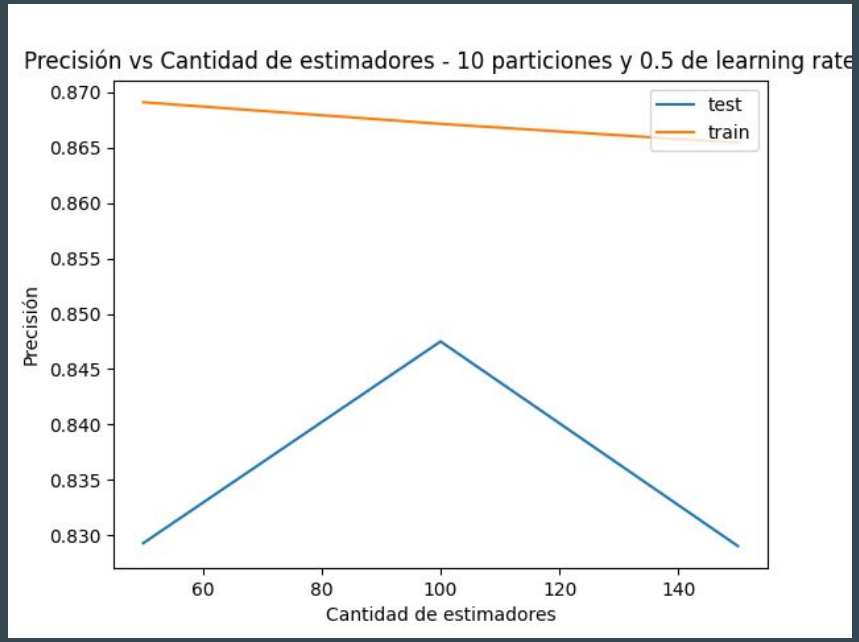
Precisión media	0.85
Desvío standard	± 0.07

Precisión vs N° estimadores

RandomForest

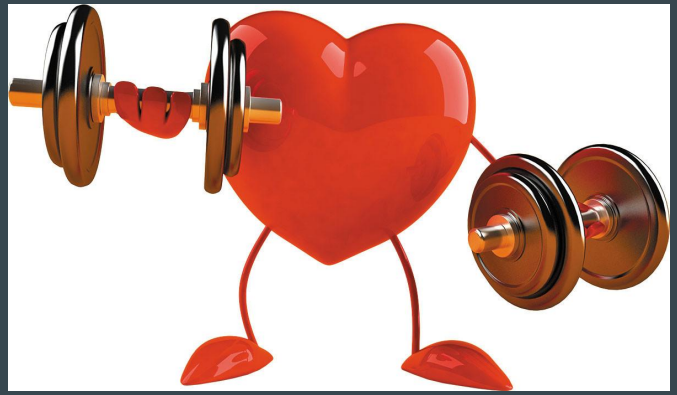


Adaboost



Conclusiones

- Construcción secuencial Vs independiente
- Ponderación a la hora de predecir una clase Vs votos uniformes
- Aprendizaje
  - Árboles de decisión: no paramétrico (se aprende la estructura)
  - AdaBoost agrega el hecho de tener en cuenta los errores del estimador previo



Muchas gracias!

Dey, Patrick
Lombardi, Matías
Vázquez, Ignacio