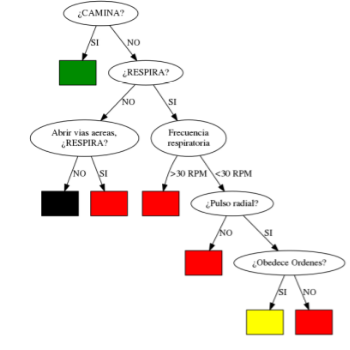
**CART (Classification & Regression Trees)**

**Árbol de Decisión:** Herramientas de Machine Learning muy potentes que nos permiten resolver problemas complejos con buena performance. Son **fáciles de visualizar y comunicar** y esto ayuda cuando el remitente es alguien no técnico.

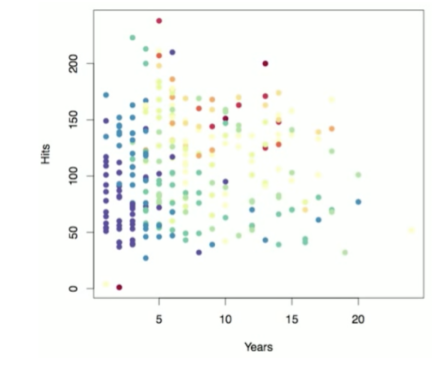
IE: **Triage** es un método de **selección y clasificación de pacientes** que se usa en la enfermería y medicina de emergencias y desastres. Lo que hace este modelo es evaluar las prioridades de atención, privilegiando la posibilidad de supervivencia en función de las necesidades terapéuticas y los recursos disponibles. Lo que busca es evitar que se retrase la atención de aquél paciente que empeoraría su pronóstico si no se lo atiende rápido. En este ejemplo, los colores determinan la gravedad del paciente:



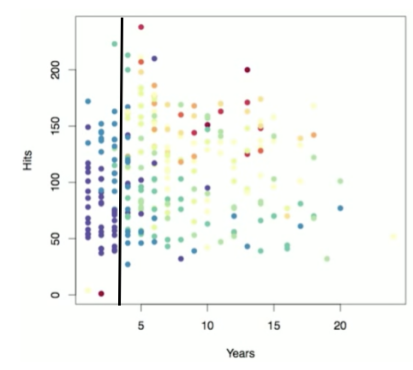
A partir de los valores que vayan tomando las variables, con el árbol tenemos una regla precisa para decidir la gravedad del paciente.

Los árboles de decisión son una **técnica de aprendizaje estadístico** que **puede usarse** tanto en la **regresión**, como en la **clasificación**. Implican **estratificar el espacio de predictores** en **regiones simples**. Con la **media o moda** de la **variable de respuesta en cada región** se pude **predecir** si una observación pertenece a una región determinada. Los árboles de decisión son **simples y útiles para interpretar**, pero son **menos competitivos en precisión predictiva** que los mejores enfoques basados en aprendizaje supervisado. Para contrarrestar esto, podemos combinar un gran número de árboles: Vamos a mejorar sustancialmente la precisión de las predicciones, pero se va a perder un poco la claridad para interpretar resultados.

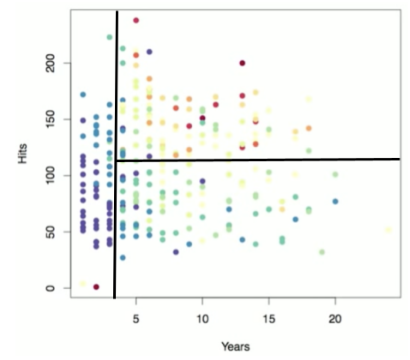
**Árboles de Regresión:** En este ejemplo, se predicen los salarios de jugadores en función de sus años de experiencia y de sus goles. Los salarios más azules son más bajos. Los salarios más rojos son más altos:



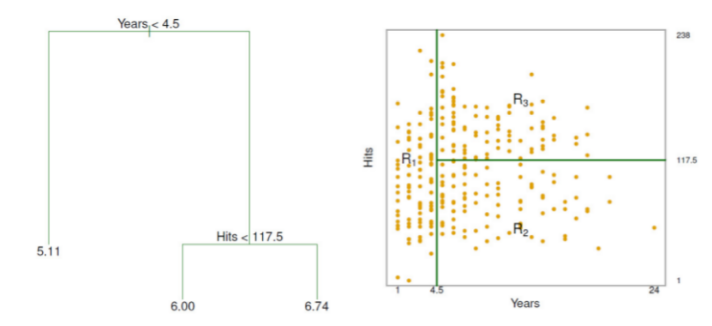
Se hace una primera partición en menos o más de 5 años:



Luego, se hace una segunda segregación por cantidad de goles:



Ahora vamos a representar esto mismo gráficamente en un árbol:

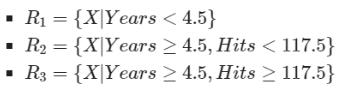


En un nodo interno dado, la etiqueta, de la forma Xj < tk indica la rama izquierda; mientras que la rama derecha corresponde a Xj >= tk.

IE: La división de la parte superior del árbol se da como resultado de que Años < 4.5 (izquierda) o Años >= 4.5 (derecha).

Los números en cada hoja indican la media de la respuesta de las observaciones que caen ahí; es decir: Para los jugadores con menos de 4.5 años de experiencia, la media del salario de los logaritmos es 5.107. Entonces el valor pronosticado para el salario es e5.107 miles de dólares.

Podemos escribir estas 3 regiones matemáticamente como:

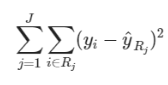


A las regiones R1, R2 y R3 se las llama **nodos terminales / hojas del árbol**.

Los nodos del árbol donde se particiona el espacio de predictores se llaman **nodos internos**.

Las aristas del árbol que conectan los nodos se llaman **ramas.**

Se busca encontrar las Cajas R1,…,Rj de forma tal que minimicen la suma de residuos al cuadrad (RSS):

, siendo y la media de de la caja Rj.

Como **computacionalmente** **no es posible considerar todas las posibles particiones** del espacio de atributos en J cajas, lo que se hace es usar un **enfoque de arriba hacia abajo “greedy”**, conocido como **recursive binary splitting**: Empieza en la parte superior del gráfico, y va particionando sucesivamente el espacio de predictores. Con cada nueva división se generan dos nuevas ramas que siguen hacia abajo del árbol.

Este es un modelo:

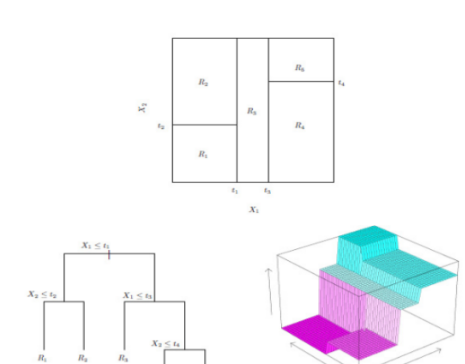
* **Recursivo:** Divide el trabajo en partes. Y resuelve cada parte subdividiéndola a su vez en partes más pequeñas.
* **Voraz (greedy):** En cada paso de la construcción del árbol se busca la mejor división en ese punto, sin mirar lo que pasa más abajo.
* **Óptimo Local:** No alcanza la mejor solución de todas las posibles, si no una solución localmente óptima.

Lo que termina haciendo el algoritmo en definitiva es dividir recursivamente los registros en subconjuntos cada vez más chicos.

1. Se **selecciona** el **predictor Xj** y el **punto de corte s**. de forma tal que va a separar el espacio en las regiones {X|Xj < s} y {X|Xj >= s}, reduciendo lo máximo posible la métrica RSS. Entonces considera todos los predictores X1,…,Xp y todos los posibles valores de corte s; y va a terminar eligiendo aquél par que resulte en el menor valor de RSS.:



1. A partir de la nueva partición, se repite el paso 1), pero trabajando sobre una de las regiones particionadas en el paso 1.
2. Repite paso 2 trabajando con las 3 particiones por separado. Y sigue hasta que se cumpla algún criterio de detención (IE: cantidad de observaciones en cada región). **Una vez generadas las regiones** R1,…,Rj, con la **media de las observaciones de training en la región** a la que pertenece la observación del test, **podemos predecir la respuesta** de una observación en test.



**Árboles de Clasificación:** **Similar a los árboles de Regresión**, pero **se usan para predecir** una **variable cualitativa**. La **predicción** la obtiene a partir de la **etiqueta mayoritaria para las observaciones de training en la región.** Se usa también **recursary binary splitting**, pero usa otro criterio en lugar de RSS. Por ejemplo, la **tasa de error de clasificación** (**fracción de** **observaciones en training que no pertenecen a la clase mayoritaria**).

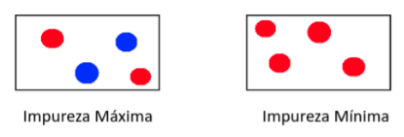
mkrepresenta la **proporción** de **observaciones de entrenamiento** en la **m-ésima región** que pertenecesn a la **k-ésima clase**. O sea, la cantidad de observaciones de la clase k en la región m, divido la cantidad de observaciones totales en la región m.

Si medimos max (mk) obtenemos el valor de la proporción de la clase mayoritaria en la región m. Entonces todo lo que no pertenezca a esta clase mayoritaria en la región m, será error con el que estemos trabajando al clasificar en la región m:

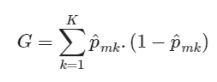


La **tasa del error de clasificación NO es lo suficientemente sensible** para construir el árbol. Por eso se prefieren otras 2 medidas en la práctica:

1. **Impureza:**

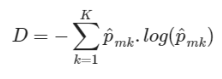


Para medir la impureza, se usa el índice de Gini:



Lo que mide este índice es la varianza total a lo largo de las K clases para cada nodo terminal m. Si las proporciones tienen valores cercanos a 1 o 0, va a tener valores chicos. Se lo considera como una medida de pureza de los nodos terminales. Si da chica, entonces se puede asumir que hay una clase predominante mayoritaria, caso contrario, tenemos un problema porque la distribución de clases es más pareja.

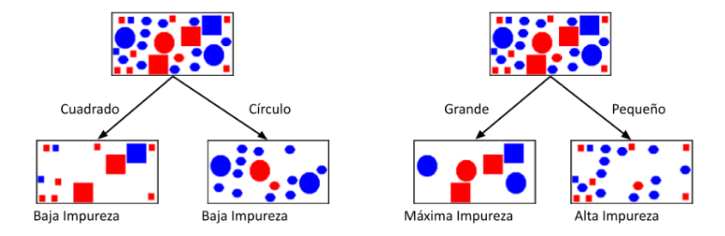
1. **Entropía:**



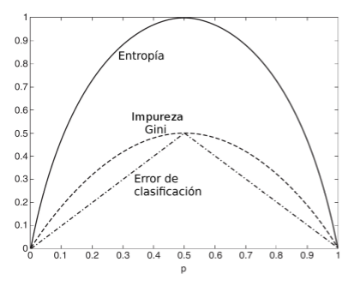
Similar al índice de Gini, la entropía será pequeña para nodos más “puros”. Al construir un árbol de clasificación, se usa el **índice de Gini o la Entropía** para **evaluar una división en particular**, ya que **estas medidas son más sensibles a la pureza** de los nodos **que la tasa de error de clasificación**.

**Optimización:**

Ejemplo: Tenemos el siguiente algoritmo y podemos seleccionar por forma o por tamaño. En este caso conviene particionar por forma, ya que las particiones resultantes son más puras:

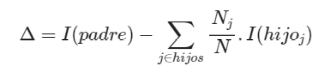
****

En cada paso de partición se buscará crear la partición con la pureza más alta posible. Esto vuelve necesario usar una **función objetivo** que mida la **ganancia en pureza de una división particular**, para maximizarla. Queremos que esta función **dependa de la distribución de clases en los** **nodos antes y después de la división**:



Estas 3 funciones tienen su máximo en 0.5 y su mínimo en 0 y 1.

Esta comparación se hace usando la **ganancia:**



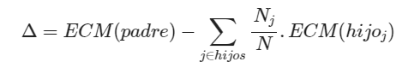
I es la medida de impureza. Nj es el número de registros en el Nodo Hijo j; N es el número de registros en el Nodo Padre. Es decir, la ganancia de pureza surge de restarle a la medida de impureza del nodo padre las medidas de impurezas de los nodos hijos resultantes, ponderadas por la forma en que se distribuyen los registros en los nodos hijos resultantes.

Si como medida de pureza se usa la entropía, a esta cantidad se la llama **Ganancia de Información.**

Una medida de impureza debe:

* Ser cero cuando hay una sola clase.
* Ser máxima cuando hay igual cantidad de observaciones de cada una de las clases en una región determinada.

Como la variable resultado de un caso de **regresión** es una categoría y no un valor continuo, **no podemos usar la misma medida de impureza** que con **clasificación**. Como el objetivo es predecir un valor numérico, podemos tener una **medida de impureza del valor a predecir** a partir de su **varianza**. Entonces podemos usar el **Error Cuadrático Medio como medida de impureza**. Entonces la ganancia en los casos de regresión será:



Siendo el error cuadrático medio:



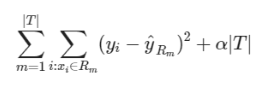
**Si un árbol resulta demasiado complejo**, es probable que termine **sobreajustando** el modelo. Si este es el caso, **se podría usar un árbol más pequeño**, con menos divisiones. Esto probablemente va a **implicar menor varianza a cambio de un mayor sesgo**. Tenemos dos caminos para podar un árbol: Antes de armar el árbol completo (**pre-poda**); o armar el árbol lo más grande posible y luego podarlo (**post-poda**)**.**

**Pre-poda**: Se limita el crecimiento del árbol con algunos criterios:

* **Reducción Mínima de Impureza.**
* **Límite máximo de Profundidad.**
* **Número de Observaciones por Hoja.**
* **Número Máximo de Hojas.**

**Post-Poda:** Armar el árbol más grande posible **T0** y después podarlo. Para no considerar cada posible subárbol, lo que se puede hacer es **incluir una penalización a la complejidad**, que se puede regular con un **alpha no negativo:**

Para cada alpha se corresponde un árbol T tal que:



**m** recorre **todas las regiones** definidas por el árbol.

**i** recorre **todas las observaciones (xi) en una región** m particular.

Rm es el **estimador del valor y** **en la región m**.

|T| es el número de nodos terminales del árbol (las hojas).

**α** es el **parámetro de tuning**, que **controla el trade-off entre** la **complejidad del subárbol** y su **ajuste a los datos de entrenamiento**.

**Conclusiones:**

**Ventajas:**

* Los árboles son **muy fáciles de explicar a las personas**.
* Parecen **más cercanos a las formas** en las que las **personas toman decisiones**.
* Se pueden **representar gráficamente**. Pueden **interpretarse fácilmente por** **no expertos.**
* Pueden **manejar fácilmente predictores cualitativos** **sin necesidad de** crear variables **dummy**.

**Desventajas:**

* **No tienen el mismo nivel de precisión** en comparación con los métodos ya vistos (regresión linear simple o múltiple con o sin regularización; knn, Regresión Logística, Naive Bayes).
* **Pueden ser poco robustos:** Un **pequeño cambio en los datos** puede **generar** un **gran cambio** en el árbol final estimado. Esto puede contrarrestarse combinando muchos árboles de decisión usando métodos como bagging, random forests y boosting; mejorando sustancialmente la performance predictiva de los árboles.

**En Python:**

import pandas as pd

import seabron as sns

from sklearn import tree

from sklearn.model\_selection import train\_test\_split, GridSearchCV, KFold, cross\_val\_score

from sklearn.metrics import mean\_squared\_error

data = pd.read\_csv(‘../Data/Hitters.csv’)

data\_complete = data.dropna()

columns = [x for x in data\_complete.columns if x not in [‘Salary’, ‘League’, ‘Division’, ‘NewLeague’]]

X = data\_complete.loc[:,columns]

y = np.log(data\_complete.loc[:,’Salary’])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.3, random\_state = 127)

my\_tree = tree.DecisionTreeRegressor(random\_state = 40)

my\_tree.fit(X\_train, y\_train)

y\_train\_pred = my\_tree.predict(X\_train)

y\_test\_pred = my\_tree.predict(X\_test)

mean\_squared\_error(y\_train, y\_train\_pred)

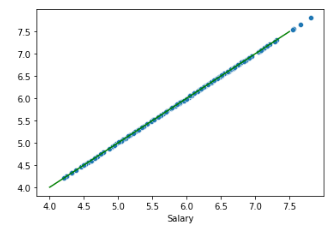
0.0

mean\_squared\_error(y\_test, y\_test\_pred)

0.3817171195740024

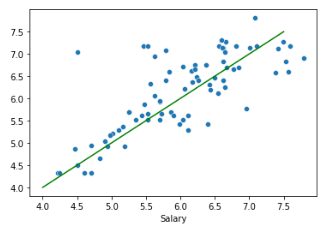
sns.scatterplot(x = y\_train, y = y\_train\_pred)

sns.lineplot(x = [4,7.5]), y = [4, 7.5], color = ‘green’)



sns.scatterplot(x = y\_test, y = y\_test\_pred)

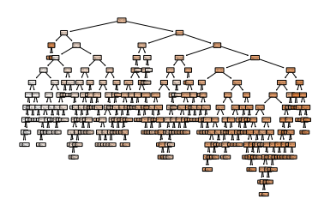
sns.lineplot(x=[4,7.5], y = [4, 7.5], color = ‘green’)



# Observemos la complejidad del árbol resultado:

tree.plot\_tree(my\_tree, feature\_names = X.columns, filled = True, rounded = True)

print()



#Cross Validation:

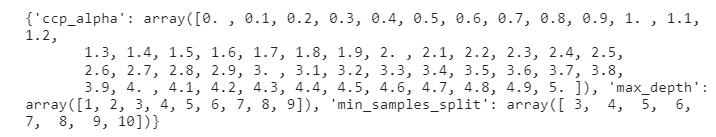
alpha\_range = np.arange(0.0, 5.1, 0.1)

max\_depth\_range = np.arange(1, 10, 1)

min\_samples\_split\_range = np.arange(3, 11, 1)

param\_grid = dict(ccp\_alpha = alpha\_range, max\_depth = max\_depth\_range, min\_samples\_split = min\_samples\_split\_range)

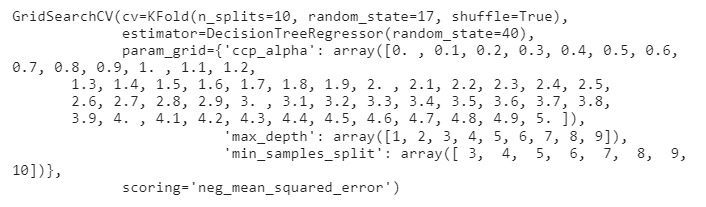
print(param\_grid)



folds = KFold(n\_splits = 10, random\_state = 17, shuffle = True)

grid = GridSearchCV(my\_tree, param\_grid, cv = folds, scoring = ‘neg\_mean\_squared\_error’)

grid.fit(X\_train, y\_train)



grid.best\_params\_



my\_best\_tree = grid.best\_estimator\_

y\_train\_pred = my\_best\_tree.predict(X\_train)

y\_test\_pred = my\_best\_tree.predict(X\_test)

mean\_squared\_error(y\_train, y\_train\_pred)

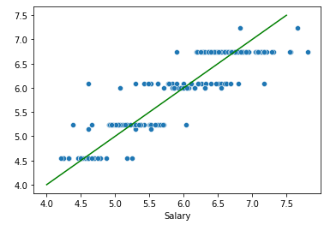


mean\_squared\_error(y\_test, y\_test\_pred)



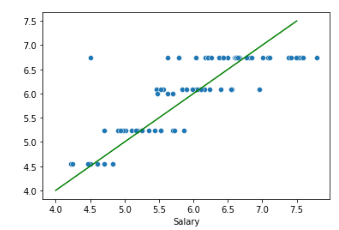
sns.scatterplot(x = y\_train, y = y\_train\_pred)

sns.lineplot(x = [4,7.5]), y = [4, 7.5], color = ‘green’)



sns.scatterplot(x = y\_test, y = y\_test\_pred)

sns.lineplot(x = [4,7.5]), y = [4, 7.5], color = ‘green’)



plt.figure(figsize=(12, 6))

tree.plot\_tree(my\_best\_tree, feature\_names = X.columns, filled = True, rounded = True, fontsize = 10)

plt.show()

