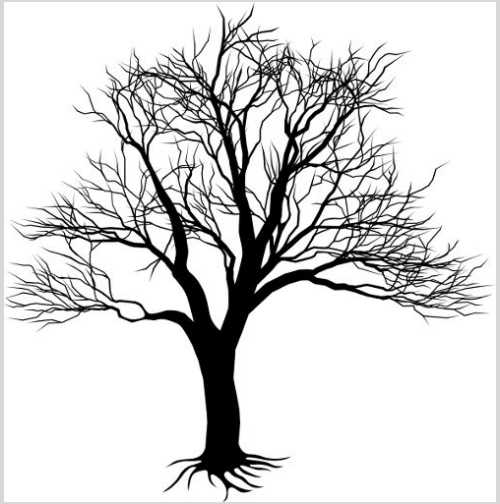


DigitalHouse > Coding School

DATA SCIENCE

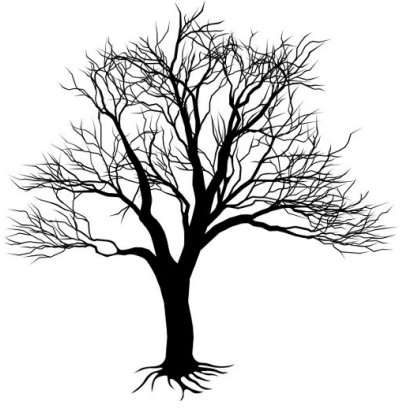
Clase 42

Intro a CARTs

2017

INTRO a CARTs

1. **Describir que es un árbol de decisión**
2. **Explicar cómo funciona un árbol de clasificación**
3. **Explicar cómo funciona un árbol de regresión**



INTRODUCCIÓN

¿Qué es un árbol de decisión?

Los árboles de decisión son una técnica de aprendizaje estadístico que puede usarse para **regresión** y para **clasificación**.

- Las técnicas de árbol implican estratificar o segmentar el espacio de predictores en un número de regiones simples. Para hacer una predicción para una observación dada típicamente se usa la media o el modo de la variable de respuesta para la región a la que pertenece la observación. Como el conjunto de reglas de partición usadas para segmentar el espacio de predictores puede resumirse en un árbol entonces estos enfoques se conocen como métodos de árboles de decisión.
- Los métodos basados en árboles son **simples y útiles para la interpretación.** Sin embargo, típicamente no son competitivos con los mejores enfoques basados en aprendizaje supervisado en términos de precisión predictiva. Por lo tanto más adelante veremo técnicas como bagging, random forests y boosting.
- Cada una de esas técnicas implica producir múltiples árboles que luego son combinados para obtener una única predicción de “consenso”. Veremos que la combinación de un gran número de árboles puede llevar a mejoras sustanciales en la precisión de las predicciones a cambio de cierta pérdida en la interpretabilidad de los resultados. .

¿Qué es un árbol de decisión?

- **No paramétrica** porque no asume una distribución paramétrica para los datos. La complejidad del modelo crece con el tamaño del training set. Es importante tomar en cuenta que al ser no paramétrico este tipo de métodos de aprendizaje tiene mayores riesgos de overfitting.

- **Jerárquica** significa que el modelo consiste en una secuencia de preguntas que definen una etiqueta de clase cuando se aplica a cualquier dato. En otras palabras, una vez entrenado, el modelo se comporta como una receta, que nos dará un resultado si se sigue exactamente.

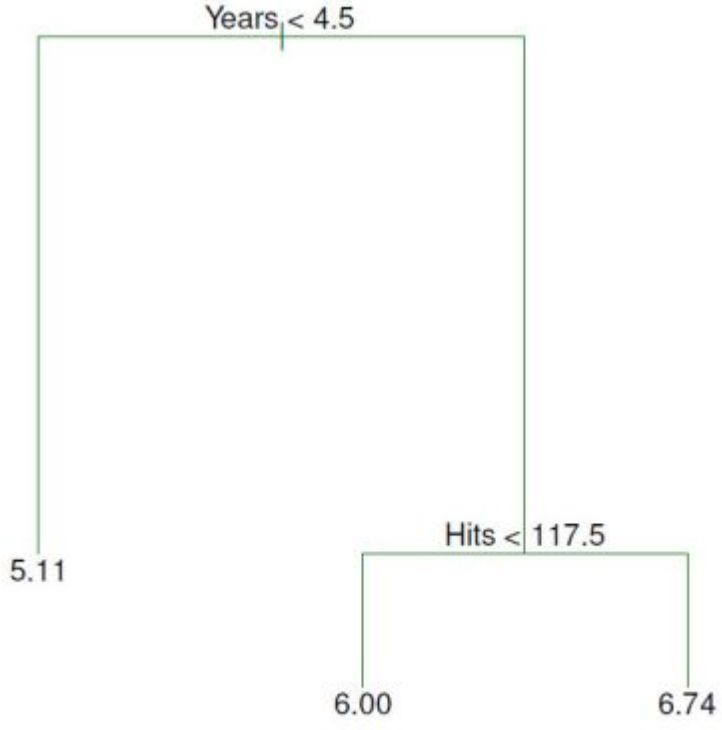
- Se trata de un modelo flexible e interpretable para la relación entre Y y X. Árboles: partir el espacio de atributos en 'rectángulos', y ajustar un modelo simple para Y dentro de cada uno de ellos.

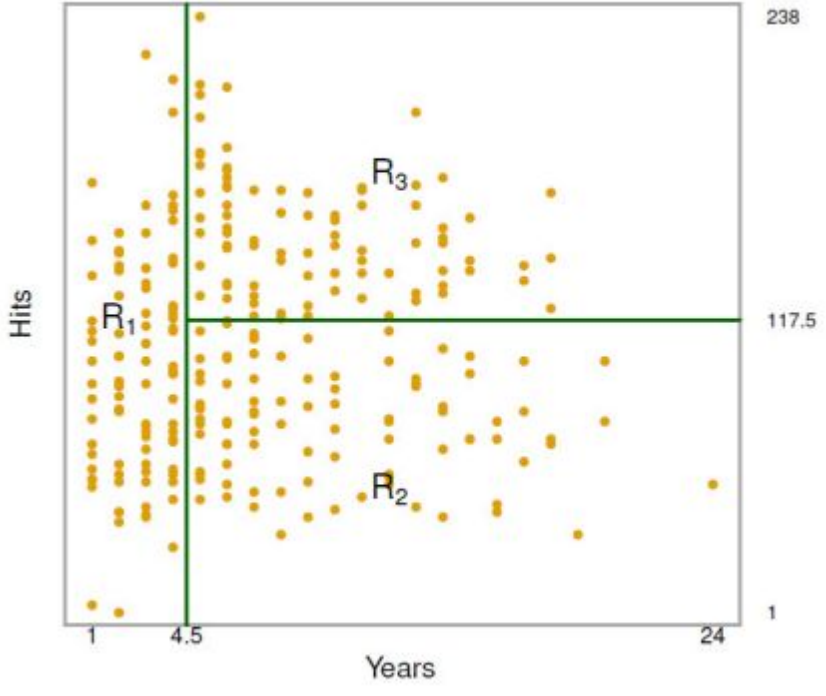
- Partimos el espacio de predictores en base a una sola variable (partición horizontal o vertical).

Ejemplo: aŕboles para regresión

Tratemos de predecir el salario de un jugador de baseball en base a su experiencia (años en la liga) y registros de hits la temporada pasada.

- Vamos a aplicar logaritmo a la variable de ingresos para reducir la dispersión (los salario se miden miles de dólares)
- En la siguiente imagen vemos un árbol de regresión aplicado a estos datos.


Years < 4.5
5.11
Hits < 117.5
6.00
6.74


Hits
238
117.5
1
R_1
R_2
R_3
1
4.5
24
Years

Ejemplo: árboles para regresión

Tratemos de predecir el salario de un jugador de baseball en base a su experiencia (años en la liga) y registros de hits la temporada pasada.

- Para los jugadores con menos de 4.5 años de experiencia el salario pronosticado está dado por la media de los salarios de jugadores con menos de 4.5 años de experiencia. Para esos jugadores la media de los logaritmos de los salario es5.107 por lo que valor pronosticado para el salario es exp(5.107) miles de dólares o $165,174,
- Para los jugadores con experiencia >=4.5 existen dos grupos posibles en base a sus hits: aquellos con menos de 118 la temporada pasada (con salario esperado de $1,000×exp(5.999) =$402,834) y otro grupo con al menos 118 hits la temporada pasada ($1,000×exp(6.740) =$845,346)
- Estas tres regiones pueden escribirse como R1 ={X | Years<4.5}, R2 ={X | Years>=4.5, Hits<117.5}, and R3 ={X | Years>=4.5, Hits>=117.5}.
- Analogía árboles

Las regiones R1, R2, y R3 se conocen como nodos terminales o hojas del árbol.

Los puntos a lo largo del árbol donde el espacio de predictores es particionado se conocen como nodos internos.

Los segmentos de los árboles que conectan los nodos se denominan ramas.

Ejemplo: árboles para regresión

Tratemos de predecir el salario de un jugador de baseball en base a su experiencia (años en la liga) y registros de hits la temporada pasada.

- Podemos interpretar la regresión de la siguiente manera. Los año de experiencia son el factor más importante para determinar el salario y los jugadores con menos experiencia ganan salarios menores que los jugadores más experimentados. Ahora, *dado que un jugador es menos experimentado s*u número de hits del año anterior parece jugar un rol menor en su salario. Pero entre los jugadores que habían estado en la liga por 4.5 o más años el número de hits del año anterior afecta su salario y los jugadores con más hits en la temporada anterior tienden a tener un salario más alto.

Las regiones R1, R2, y R3 se conocen como nodos terminales u hojas del árbol.

Los puntos a lo largo del árbol donde el espacio de predictores es particionado se conocen como nodos internos.

Los segmentos de los árboles que conectan los nodos se denominan ramas.

¿Como construimos las regiones?

- En teoría las regiones R1,...,RJ podrían tener cualquier forma.
- Sin embargo, elegimos dividir el espacio de predictores en rectángulos o cajas en varias dimensiones por simplicidad y facilidad de interpretación del modelo predictivo resultante. El objetivo es encontrar cajas R1,...,RJ que minimizan la suma de residuos al cuadrado (RSS) dada por

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

- El problema es que no es *factible* ***computacionalmente*** considerar todas las posibles particiones del espacio de atributos en J cajas. Por esta razón se usa un enfoque de arriba hacia abajo "greedy" que es conocido como recursive binary splitting.
- El ***recursive binary splitting*** comienza en la parte de arriba del árbol (donde todas las observaciones pertenecen a una sola región) y sucesivamente particiona el espacio de predictores.
- Se dice que es ***greedy*** porque en cada paso de la construcción del árbol se busca la mejor división en ese punto en particular en lugar de mirar hacia adelante y elegir una división que llevaría a un mejor árbol en un paso futuro.

¿Como construimos las regiones?

- Para hacer el recursive binary splitting primero seleccionados el predictor Xj y el punto de corte s tal que separar el espacio de predictores en las regiones {X|Xj < s} and {X|Xj ≥ s} lleva a la mayor reducción posible de la RSS.
- Para esto debemos considerar todos los predictores X1,...,Xp, y todos los valores posibles de los puntos de corte s para cada uno de los predictores y luego elegir el predictor y punto de corte tal que el árbol resultante tiene el menor RSS.
- En más detalle definimos las regiones como

$$R_1(j,s) = \{X|X_j < s\} \text{ and } R_2(j,s) = \{X|X_j \geq s\},$$

- Buscamos minimizar la siguiente expresión eligiendo Xj y s dado que los valores pronosticados para cada región están dados por el promedio de la y en esa partición.

$$\sum_{i:\, x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:\, x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2,$$

¿Como construimos las regiones?

- A continuación repetimos el proceso buscando el mejor predictor y el mejor punto de corte para volver a dividir los datos con el objetivo de minimizar el RSS para la nueva partición. Pero esta vez en lugar de dividir el espacio de predictores entero, dividimos una de las regiones identificadas previamente.

- Ahora tenemos 3 regiones. Volvemos a buscar una división de estas 3 regiones para minimizar el RSS.

- El proceso continúa hasta que se cumple con algún criterio de detención (cantidad de observaciones en cada región por ejemplo)
- Una vez que tenemos creadas las regiones R1,...,RJ predecimos la respuesta de una observación de test en base la media de las observaciones de training en la región a la que pertenece la observación de test.

Alternativas a CART: podando los árboles...

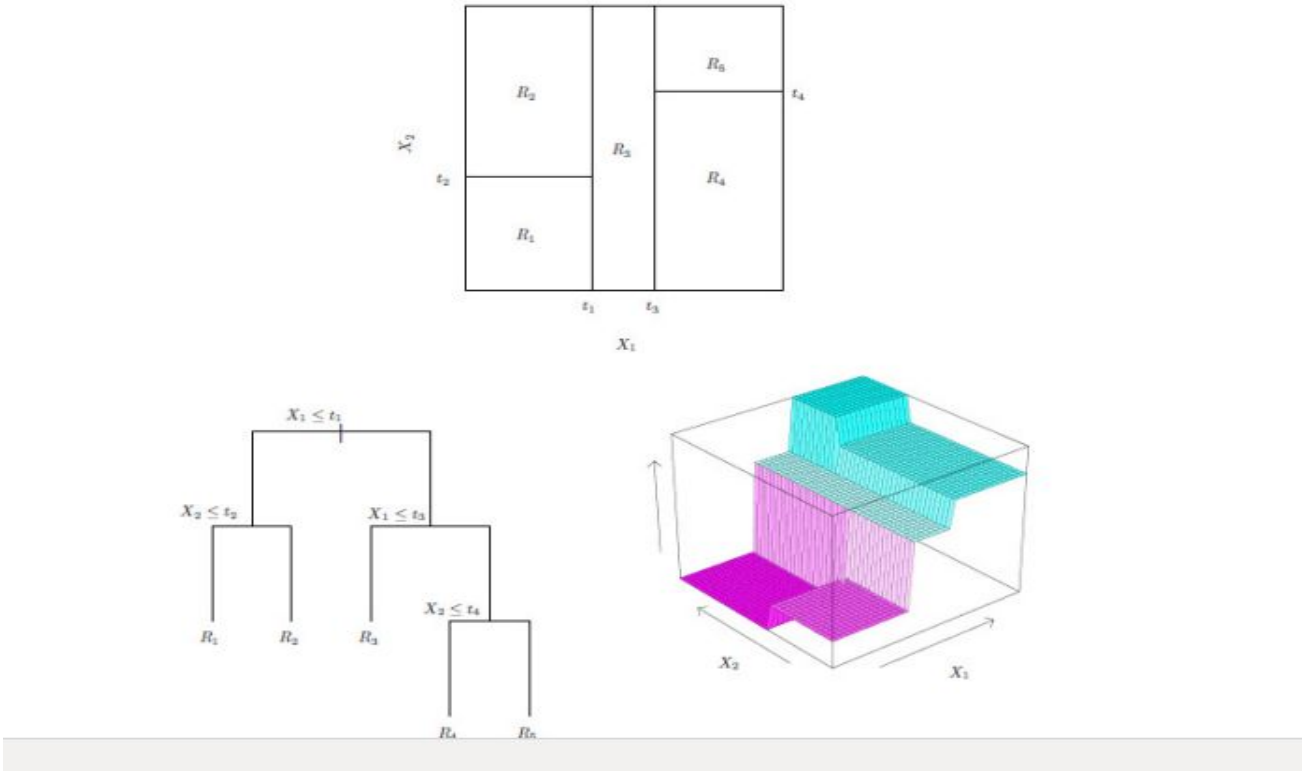
- El proceso anterior puede producir buenas predicciones en el training set pero es probable que haga un overfit a los datos de entrenamiento.
- Esto ocurre porque el árbol resultante puede ser demasiado complejo.
- Un árbol más pequeño con menos divisiones (menos regiones R1, .. RJ) podría tener menos varianza y ser más interpretable al costo de un poco más de sesgo.
- Una posibilidad sería construir los árboles sólo hasta que la reducción en el RSS en cada partición sea mayor que cierto nivel de corte elevado. Esta estrategia genera árboles más chicos pero podría ocurrir que una división aparentemente de poco valor al comienzo del árbol sea seguida por otra muy buena en términos de reducción del RSS más adelante.
- En lugar de considerar cada posible subárbol podríamos incluir una penalización por la complejidad regulada por un parámetro de tunning alpha no negativo. Para cada alpha se corresponde un subárbol T ⊂ T0 tal que

$$\sum_{m=1}^{|T|} \sum_{i:\, x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

Alternativas a CART, podando los árboles de decisión...

$$\sum_{m=1}^{|T|} \sum_{i:\, x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

- Aquí |T | indica el número de nodos terminales del árbol. El parámetro de tunning alpha controla el trade-off entre la complejidad del subárbol y su ajuste a los datos de entrenamiento.
- Con alpha= 0, el subárbol T será igual a T0. Al aumentar alpha el precio a pagar por tener más nodos terminales aumenta por lo que la expresión anterior tenderá a ser minimizada para un subárbol más pequeño. ¿Lasso?
- Podemos elegir alpha por CV y luego usamos el data set entero y obtenemos el subárbol correspondiente a alpha.
- En Python no está implementada esta técnica pero se puede replicar la intuición con los parámetros min_samples_leaf max_depth que regulan la cantidad mínima de hojas (observaciones) en cada región y la profundidad del árbol.


R_1
R_2
R_3
R_4
R_5
t_1
t_2
t_3
t_4
X_1
X_2
$X_1 \leq t_1$
$X_2 \leq t_2$
$X_1 \leq t_3$
$X_2 \leq t_4$
R_1
R_2
R_3
R_4
R_5
X_2
X_1

Árboles para clasificación

- Un árbol de clasificación es muy similar a uno de regresión excepto que se utiliza para predecir una variable cualitativa en lugar de una cuantitativa.
- La predicción se obtiene como la etiqueta mayoritaria para las observaciones de training dentro de la región a la que pertenece una observación.
- Para construir el árbol usamos recursive binary splitting pero con un criterio distinto a RSS. Una alternativa natural es la tasa de error de clasificación. Cómo asignamos una observación a la clase más frecuente en su región la tasa de error de clasificación es simplemente la fracción de las observaciones de training en esa región que no pertenecen a la clase mayoritaria.

$$E = 1 - \max_k(\hat{p}_{mk}).$$

Árboles para clasificación

- La tasa de error de clasificación no es suficientemente sensible para construir el árbol y en la práctica se prefieren otras dos medidas.
- El índice de Gini.

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

- El índice de Gini mide la varianza total a lo largo de las K clases. ¿Cuándo tomará valores pequeños el índice Gini ?
- Por esto el índice de Gini se considera como una medida de pureza de los nodos, un valor pequeño indica que un nodo contiene predominantemente observaciones de una misma clase.

Árboles para clasificación

- Otra alternativa está dada por la entropía.

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}.$$

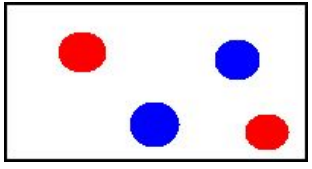
- Como el índice de Gini la entropía tomará un valor pequeño si los nodos son “puros”.
- Cuando se construye un árbol de clasificación típicamente el Gini index o la entropía usados para evaluar la calidad de una división en particular porque estas medidas son más sensibles a la pureza de los nodos que la tasa de clasificación.
- Al podar el árbol cualquiera de estas 3 medidas puede usarse pero la tasa de error de clasificación es preferible si el objetivo es la precisión de predicción del árbol final podado.

Métricas de evaluación

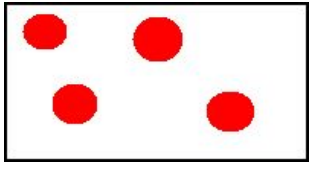
Métricas de evaluación

- Recordemos del algoritmo anterior, que iterativamente crea condiciones de prueba para dividir los datos.
- **¿Cómo determinamos la mejor división entre todas las divisiones posibles?**
- Considerando que no es necesario dividir cuando todos los registros pertenecen a la misma clase pura, buscaremos en cada paso crear la partición con la **pureza más alta posible**.
- Para hacer esto, necesitaremos una función objetivo para optimizar, que mida la ganancia en pureza de una división particular. Por lo tanto, queremos que dependa de la distribución de clases sobre los nodos (antes y después de la división).

Supongamos que queremos hacer una clasificación según el color de las figuras.



Impureza Máxima



Impureza Mínima

- Definamos *p(i|t)* como la probabilidad de la clase *i* en el nodo *t* (por ejemplo, la fracción de registros con la etiqueta *i* en el nodo *t*).
- Por lo tanto, para un problema de clasificación binaria (0/1), la distribución máxima de impureza, donde ambas clases están presentes de igual manera, viene dada por la distribución:

$$p(0|t) = p(1|t) = 0.5$$

- Por otra parte, la mínima de impureza (o máxima pureza) se obtiene cuando está presente sólo una clase, es decir:

$$p(0|t) = 1 - p(1|t) = 0 \vee 1$$

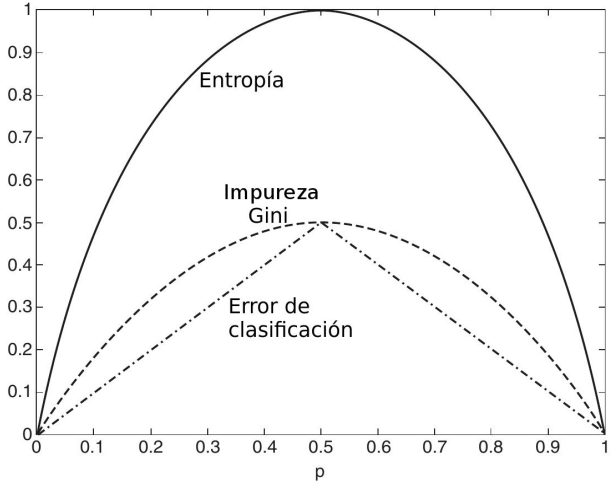
- Por lo tanto, en el caso de una clasificación binaria necesitamos definir una función de impureza que variará suavemente entre los dos casos extremos de impureza mínima (una clase u otra solamente) y el caso de impureza máxima (igual cantidad de ambas clases).

- Se pueden definir varias funciones que satisfagan las condiciones anteriores.
  Aquí las tres más comunes:

$$\text{Entropía}(t) = -\sum_{i=0}^{c-1} p(i|t) log_2 p(i|t)$$

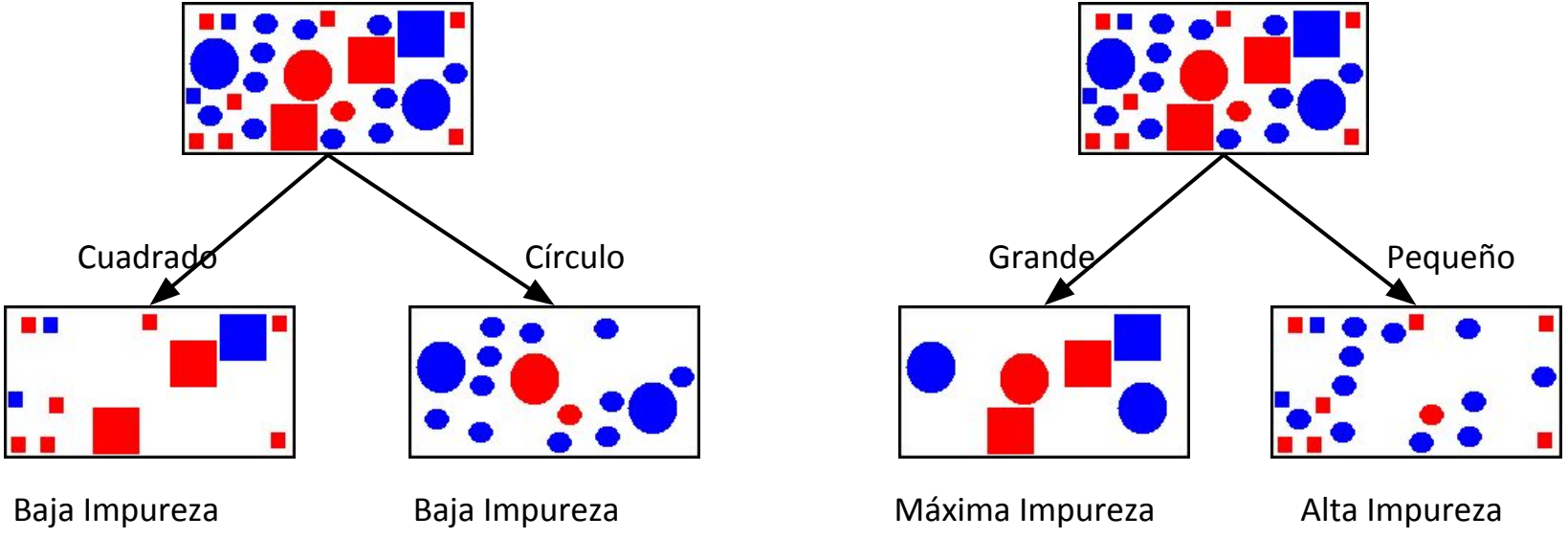
$$\text{Impureza Gini}(t) = 1 - \sum_{i=0}^{c-1} [p(i|t)]^2$$

$$\text{Error de clasificación}(t) = 1 - \max_i [p(i|t)]$$



- Observar que las tres funciones tienen su máximo en 0.5 y sus mínimos en 0 y 1.

Siguiendo con el ejemplo de la clasificación según el color de las figuras. Si podemos seleccionar la forma y el tamaño, ¿que partición conviene realizar?



- Las medidas de impureza, por sí solas, no son suficientes para decirnos cómo funcionará la división. Todavía tenemos que mirar la impureza antes y después de la división. Podemos hacer esta comparación usando la ganancia:

$$\Delta = I(\text{padre}) - \sum_{j \in \text{hijos}} \frac{N_j}{N} I(\text{hijo}_j)$$

- Donde *I* es la medida de impureza, *Nj* es el número de registros en el nodo hijo *j* y *N* es el número de registros en el nodo padre.
- Cuando *I* es la entropía, esta cantidad se llama **ganancia de información**.
- En general, una condición de prueba con un alto número de resultados puede conducir a sobreajuste (por ejemplo: una división con un solo resultado por registro). Una forma de tratar esto es restringir el algoritmo a únicamente divisiones binarias (CART). Otra forma es utilizar un criterio de división que penalice explícitamente el número de resultados (C4.5)

Previniendo el sobreajuste (overfitting)

- Además de determinar las divisiones, también necesitamos un criterio de detención para decirnos cuándo terminamos. Por ejemplo, podemos detenernos cuando todos los registros pertenecen a la misma clase, o cuando todos los registros tienen los mismos atributos. Esto es correcto en principio, pero probablemente conduciría a un sobreajuste.

- Una posibilidad es hacer una prepoda, que implica establecer un umbral mínimo en la ganancia, y detenerse cuando ninguna división logra una ganancia por encima de este umbral. Esto evita el sobreajuste, pero es difícil de calibrar en la práctica.

- Alternativamente podríamos construir el árbol completo, y luego realizar una poda como un paso de post-procesamiento. Para podar un árbol, examinamos los nodos desde abajo hacia arriba y simplificamos las ramas del árbol (de acuerdo con algunos criterios). Los subárboles complicados pueden ser reemplazados con un solo nodo o con un subárbol más simple (secundario).

Árboles de Regresión

- En el caso de regresión, la variable de resultado no es una categoría sino un valor continuo. Por lo tanto, no podemos usar la misma medida de impureza que utilizamos para la clasificación.
- No resulta difícil ver que si el objetivo es predecir un valor numérico, la varianza de dicho valor en un nodo nos dá una medida de impureza de dicho valor.
- Por ello utilizaremos el Error Cuadrático Medio como medida de impureza y la función a maximizar seguirá siendo la ganancia.

$$\Delta = ECM(\text{padre}) - \sum_{j \in \text{hijos}} \frac{N_j}{N} ECM(\text{hijo}_j)$$

- El objetivo es buscar la máxima ganancia, donde ECM es el Error Cuadrático Medio, Nj es el número de registros en el nodo hijo j y N es el número de registros en el nodo padre.

Árboles de decisión, pros y contras

CARTs

Ventajas

- Los árboles son muy fáciles de explicar a las personas
- Parecen más cercanos a la forma en la que las personas toman decisiones
- Pueden ser representados gráficamente, pueden ser interpretados por no expertos fácilmente (especialmente si son pequeños)

Desventajas

- En general no tienen el mismo nivel de precisión en la predicción comparados con otros enfoques para regresión y clasificación vistos previamente.
- Además pueden ser poco robustos. Un pequeño cambio en los datos puede generar un gran cambio el árbol final estimado. Sin embargo al agregar muchos árboles de decisión usando métodos como bagging, random forests, y boosting la performance predictiva de los árboles puede mejorarse sustancialmente. *To be continued..@Clase 45 @Clase46.*

ANEXO

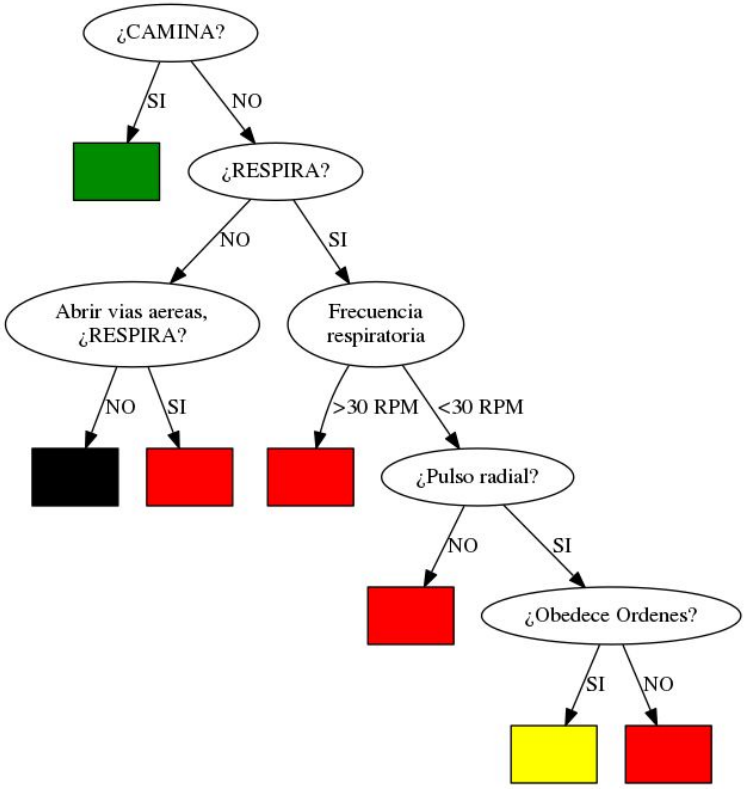
¿Cómo construir un árbol de decisión?

- Para construir un árbol de decisión necesitamos un algoritmo que sea capaz de determinar las decisiones óptimas para cada nodo. Uno de estos algoritmos es el algoritmo de Hunt: un algoritmo recursivo, voraz que conduce a una solución localmente óptima.
  - **recursivo**: divide el trabajo en partes, y resuelve cada parte dividiéndolas a su vez en partes más pequeñas.
  - **voraz** (greedy): el algoritmo toma una decisión localmente óptima en cada paso
  - **óptimo local**: no alcanza la mejor solución de todas las posibles, sino una solución localmente óptima.

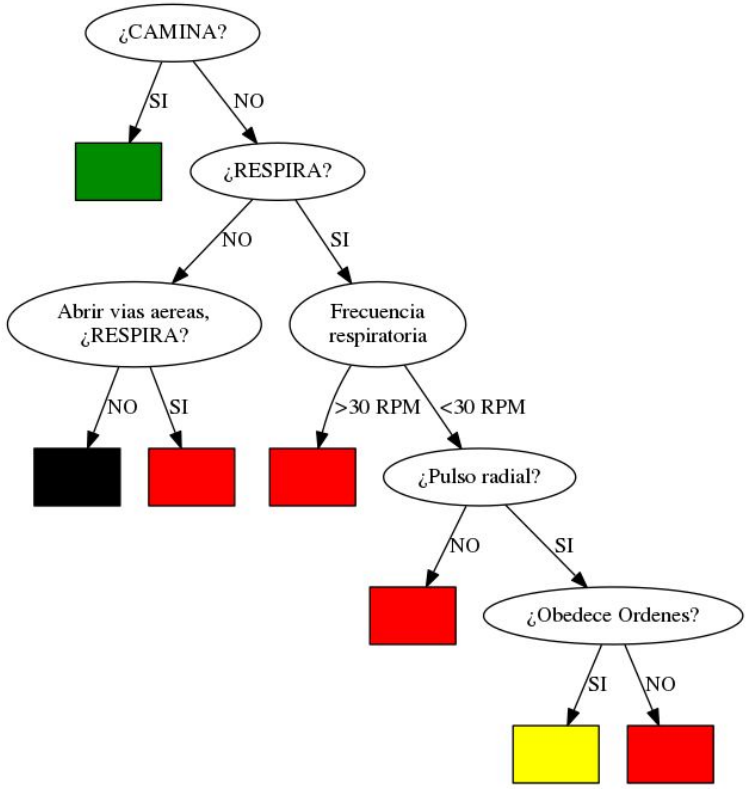
- El algoritmo funciona dividiendo recursivamente los registros en subconjuntos cada vez más pequeños. La decisión de partición se realiza en cada nodo de acuerdo con una métrica llamada pureza. Se dice que un nodo es 100% puro cuando todos sus registros pertenecen a una sola clase (o tienen el mismo valor).

Consideremos como ejemplo el método Triage para la clasificación de heridos según su gravedad. Aquí los colores determinan la gravedad del paciente.

- El árbol da una regla precisa para decidir la gravedad del paciente, dependiendo de los valores tomados por las variables.
- La interpretación del modelo es extremadamente simple lo que facilita su comunicación a gente no técnica.



- El árbol es un caso de *Grafo Dirigido Acíclico* , y como tal, tiene Nodos y Aristas. Los nodos corresponden a preguntas, mientras que las aristas corresponden a posibles respuestas.
- El nodo superior se denomina **nodo raíz**, éste no tiene aristas entrantes y tiene 2 o más aristas salientes.
- Los **nodos internos** tienen 1 arista entrante y 2+ aristas salientes.
- Finalmente, los nodos que tienen 1 arista entrante y ninguna saliente, se denominan **nodos hoja** y contienen una etiqueta de clase (o un valor de regresión).



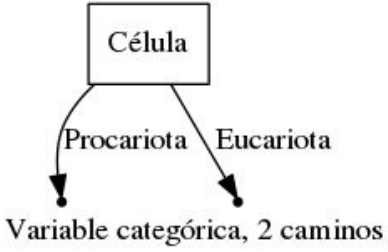
El algoritmo de Hunt

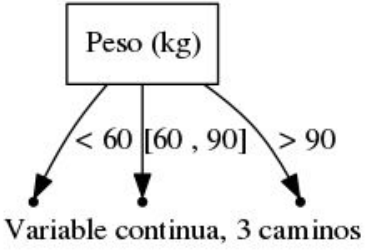
Sea Dt el conjunto de registros de entrenamiento para el nodo-t
Sea Y = {y1, y2, ..., yn} el conjunto de etiquetas de las clases.

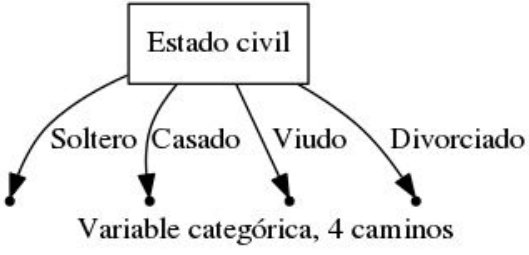
- Si todos los registros en Dt pertenecen a la misma clase Yt, entonces el nodo-t es un nodo hoja de clase Yt

- Si Dt tiene registros de más de una clase, entonces:
  - Crear una condición de prueba para particionar los registros de Dt.
  - Por cada partición crear una arista y nodo hijo.
  - Aplicar recursivamente el algoritmo a los nodos hijos.

Las particiones pueden ser en 2 o más caminos. Las variables pueden ser categóricas o continuas.







Demo del algoritmo de Hunt

Ejemplo del algoritmo de Hunt

Vamos a construir el árbol de decisión para poder decidir si se puede jugar al golf.
El set de entrenamiento es la tabla de la derecha.

Pronóstico	Temperatura	Humedad	Viento	Jugar
Soleado	Alta	Alta	Débil	No
Soleado	Alta	Alta	Fuerte	No
Nublado	Alta	Alta	Débil	Si
LLuvia	Media	Alta	Débil	Si
LLuvia	Baja	Normal	Débil	Si
LLuvia	Baja	Normal	Fuerte	No
Nublado	Baja	Normal	Fuerte	Si
Soleado	Media	Alta	Débil	No
Soleado	Baja	Normal	Débil	Si
LLuvia	Media	Normal	Débil	Si
Soleado	Media	Normal	Fuerte	Si
Nublado	Media	Alta	Fuerte	Si
Nublado	Alta	Normal	Débil	Si
LLuvia	Media	Alta	Fuerte	No

Ejemplo del algoritmo de Hunt

Vamos a construir el árbol de decisión para poder decidir si se puede jugar al golf.
El set de entrenamiento es la tabla de la derecha.

- En primer lugar debemos verificar si todos los registros pertenecen a la misma clase, ya que en ese caso deberíamos construir un nodo hoja con esa clase como etiqueta.

Pronóstico	Temperatura	Humedad	Viento	Jugar
Soleado	Alta	Alta	Débil	**No**
Soleado	Alta	Alta	Fuerte	**No**
Nublado	Alta	Alta	Débil	**Si**
LLuvia	Media	Alta	Débil	**Si**
LLuvia	Baja	Normal	Débil	**Si**
LLuvia	Baja	Normal	Fuerte	**No**
Nublado	Baja	Normal	Fuerte	**Si**
Soleado	Media	Alta	Débil	**No**
Soleado	Baja	Normal	Débil	**Si**
LLuvia	Media	Normal	Débil	**Si**
Soleado	Media	Normal	Fuerte	**Si**
Nublado	Media	Alta	Fuerte	**Si**
Nublado	Alta	Normal	Débil	**Si**
LLuvia	Media	Alta	Fuerte	**No**

Ejemplo del algoritmo de Hunt

Vamos a construir el árbol de decisión para poder decidir si se puede jugar al golf.
El set de entrenamiento es la tabla de la derecha.

- En primer lugar debemos verificar si todos los registros pertenecen a la misma clase, ya que en ese caso deberíamos construir un nodo hoja con esa clase como etiqueta.
  Como no es el caso, vamos a particionar nuestro set según la variable Pronóstico.

Pronóstico	Temperatura	Humedad	Viento	Jugar
Soleado	Alta	Alta	Débil	No
Soleado	Alta	Alta	Fuerte	No
Nublado	Alta	Alta	Débil	Si
LLuvia	Media	Alta	Débil	Si
LLuvia	Baja	Normal	Débil	Si
LLuvia	Baja	Normal	Fuerte	No
Nublado	Baja	Normal	Fuerte	Si
Soleado	Media	Alta	Débil	No
Soleado	Baja	Normal	Débil	Si
LLuvia	Media	Normal	Débil	Si
Soleado	Media	Normal	Fuerte	Si
Nublado	Media	Alta	Fuerte	Si
Nublado	Alta	Normal	Débil	Si
LLuvia	Media	Alta	Fuerte	No

Ejemplo del algoritmo de Hunt

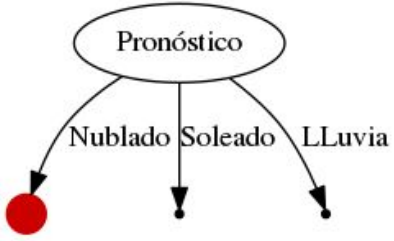
Vamos a construir el árbol de decisión para poder decidir si se puede jugar al golf.
El set de entrenamiento es la tabla de la derecha.

- En primer lugar debemos verificar si todos los registros pertenecen a la misma clase, ya que en ese caso deberíamos construir un nodo hoja con esa clase como etiqueta.
  Como no es el caso, vamos a particionar nuestro set según la variable Pronóstico.

- Por cada | n nodo hijo.

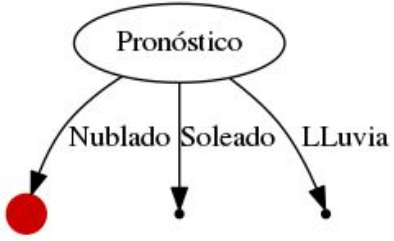
Pronóstico	Temperatura	Humedad	Viento	Jugar
Soleado	Alta	Alta	Débil	No
Soleado	Alta	Alta	Fuerte	No
Nublado	Alta	Alta	Débil	Si
LLuvia	Media	Alta	Débil	Si
LLuvia	Baja	Normal	Débil	Si
LLuvia	Baja	Normal	Fuerte	No
Nublado	Baja	Normal	Fuerte	Si
Soleado	Media	Alta	Débil	No
Soleado	Baja	Normal	Débil	Si
LLuvia	Media	Normal	Débil	Si
Soleado	Media	Normal	Fuerte	Si
Nublado	Media	Alta	Fuerte	Si
Nublado	Alta	Normal	Débil	Si
LLuvia	Media	Alta	Fuerte	No

- Ahora aplicamos recursivamente el algoritmo. En primer lugar analizamos la partición correspondiente a Pronóstico=Nublado.



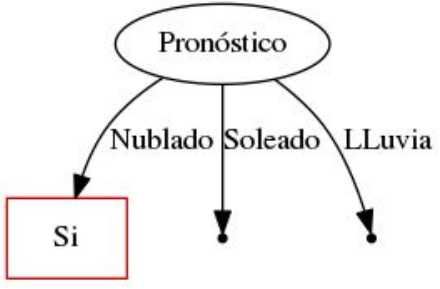
Pronóstico	Temperatura	Humedad	Viento	Jugar
Nublado	Alta	Alta	Débil	Si
Nublado	Baja	Normal	Fuerte	Si
Nublado	Media	Alta	Fuerte	Si
Nublado	Alta	Normal	Débil	Si

- Ahora aplicamos recursivamente el algoritmo. En primer lugar analizamos la partición correspondiente a Pronóstico=Nublado.

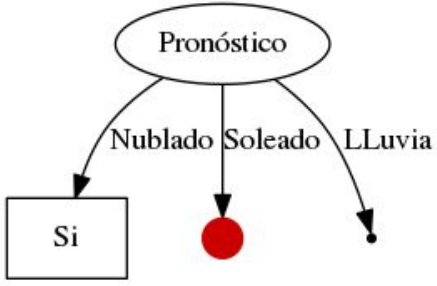


Pronóstico	Temperatura	Humedad	Viento	Jugar
Nublado	Alta	Alta	Débil	Si
Nublado	Baja	Normal	Fuerte	Si
Nublado	Media	Alta	Fuerte	Si
Nublado	Alta	Normal	Débil	Si

- Al analizar a qué clase pertenecen los registros, vemos que todos corresponden a "Si" por lo tanto este será un nodo hoja con la etiqueta "Si".

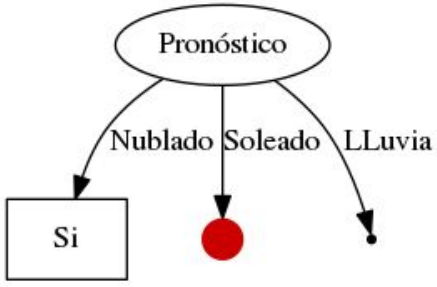


- Ahora analicemos la partición correspondiente a Pronóstico=Soleado.



Pronóstico	Temperatura	Humedad	Viento	Jugar
Soleado	Alta	Alta	Débil	No
Soleado	Alta	Alta	Fuerte	No
Soleado	Media	Alta	Débil	No
Soleado	Baja	Normal	Débil	Si
Soleado	Media	Normal	Fuerte	Si

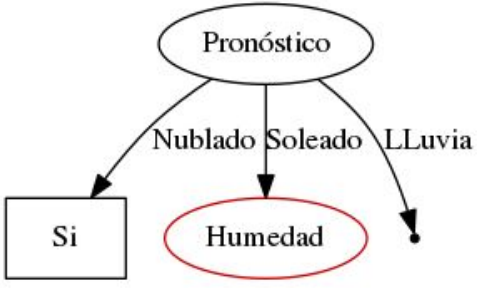
- Ahora analicemos la partición correspondiente a Pronóstico=Soleado.



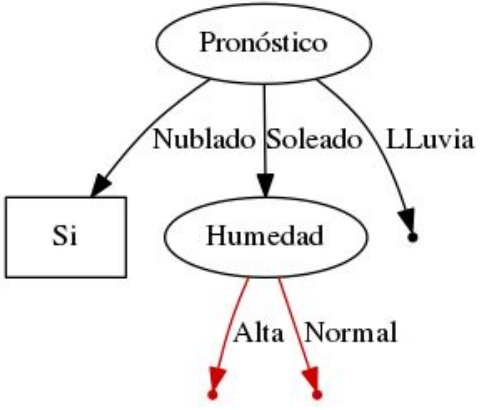
Pronóstico	Temperatura	Humedad	Viento	Jugar
Soleado	Alta	Alta	Débil	No
Soleado	Alta	Alta	Fuerte	No
Soleado	Media	Alta	Débil	No
Soleado	Baja	Normal	Débil	Si
Soleado	Media	Normal	Fuerte	Si

- Como podemos ver, los registros pertenecen a distintas clases, por esta razón tendremos que sub-particionar. Podemos optar por particionar según las siguientes variables: {Temperatura, Humedad y Viento}.
- ¿Cual conviene utilizar?

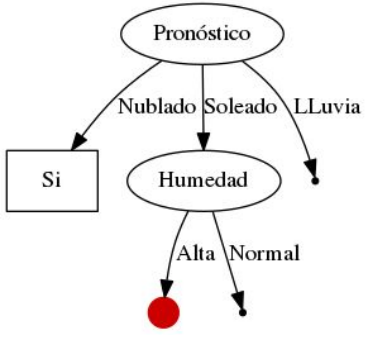
- Al analizar los datos, resulta evidente que la forma más simple es particionar por Humedad, ya que si seleccionásemos Viento o Temperatura, deberíamos hacer una división inferior más.



- Ahora que tenemos seleccionado el criterio, creamos las particiones.

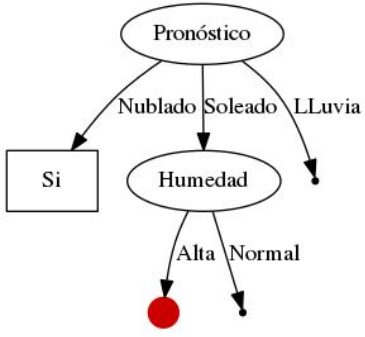


- Ahora debemos aplicar recursivamente el algoritmo.
  Para la sub-partición Humedad=Alta tenemos este caso:



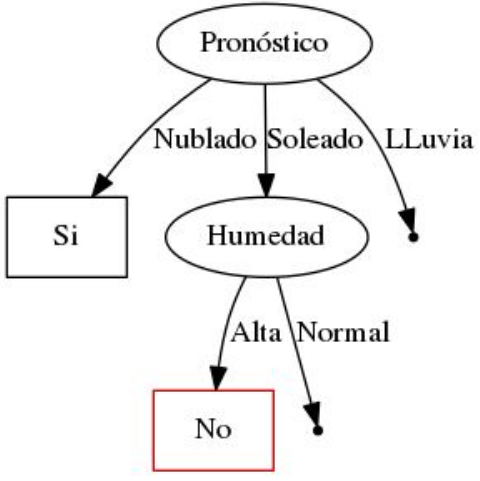
Pronóstico	Temperatura	Humedad	Viento	Jugar
Soleado	Alta	Alta	Débil	No
Soleado	Alta	Alta	Fuerte	No
Soleado	Media	Alta	Débil	No

- Ahora debemos aplicar recursivamente el algoritmo. Para la sub-partición Humedad=Alta tenemos este caso:

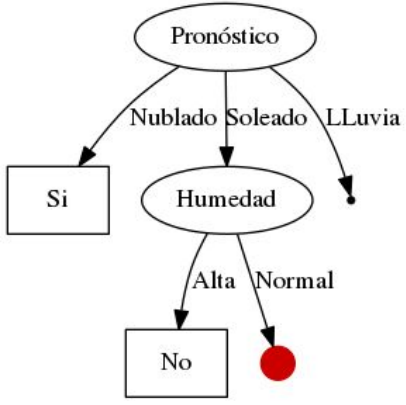


Pronóstico	Temperatura	Humedad	Viento	Jugar
Soleado	Alta	Alta	Débil	No
Soleado	Alta	Alta	Fuerte	No
Soleado	Media	Alta	Débil	No

- Como podemos ver que todos los registros pertenecen a la clase "No", sabemos que será un nodo hoja con la etiqueta "No"

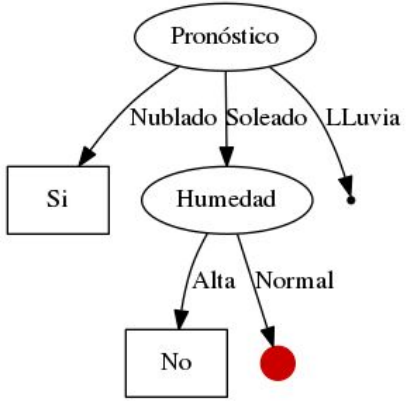


- Para la sub-partición Humedad=Normal tenemos este caso:



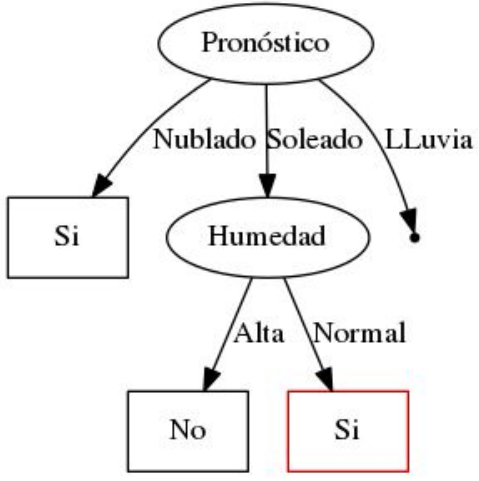
Pronóstico	Temperatura	Humedad	Viento	Jugar
Soleado	Baja	Normal	Débil	Si
Soleado	Media	Normal	Fuerte	Si

- Para la sub-partición Humedad=Normal tenemos este caso:

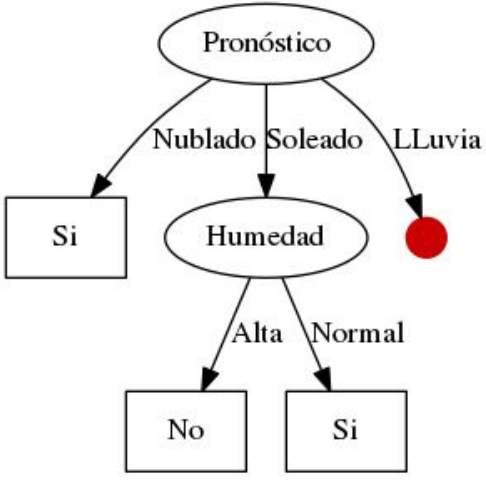


Pronóstico	Temperatura	Humedad	Viento	Jugar
Soleado	Baja	Normal	Débil	Si
Soleado	Media	Normal	Fuerte	Si

- Como podemos ver que todos los registros pertenecen a la clase "Si", sabemos que será un nodo hoja con la etiqueta "Si"



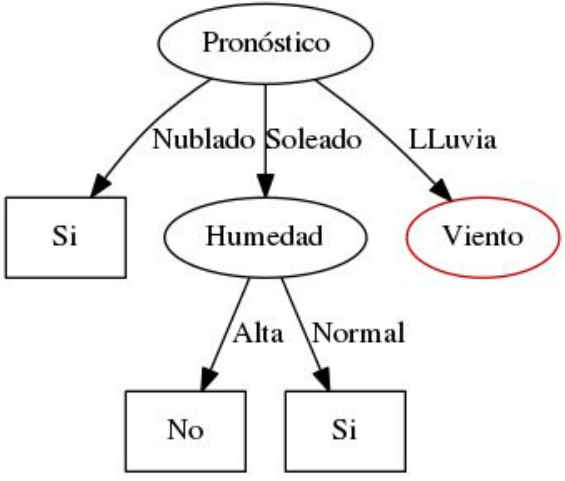
- Ahora queda analizar la partición correspondiente a Pronóstico=LLuvia.



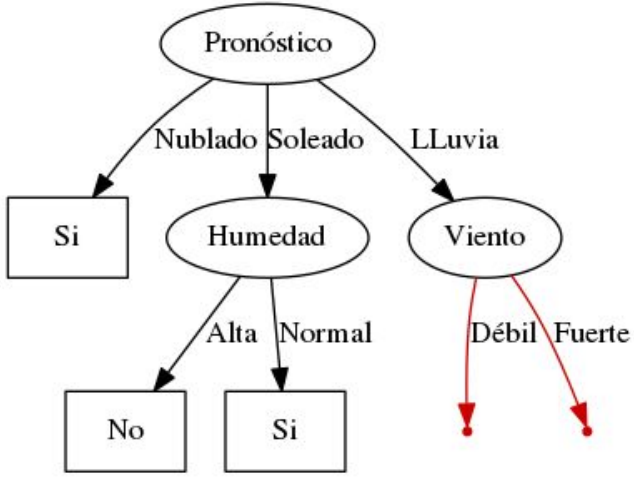
Pronóstico	Temperatura	Humedad	Viento	Jugar
LLuvia	Media	Alta	Débil	Si
LLuvia	Baja	Normal	Débil	Si
LLuvia	Baja	Normal	Fuerte	No
LLuvia	Media	Normal	Débil	Si
LLuvia	Media	Alta	Fuerte	No

- Nuevamente vemos que los registros pertenecen a clases distintas y tendremos que sub-particionar. Podemos optar por particionar según las siguientes variables: {Temperatura, Humedad y Viento}.
- ¿Cual conviene utilizar?

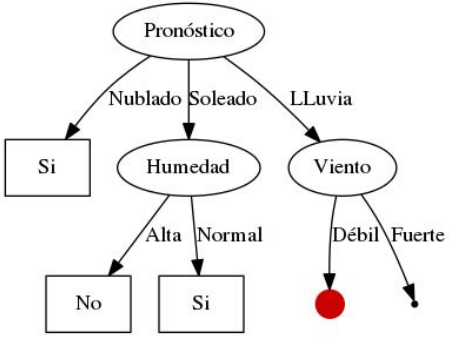
- En este caso resulta más conveniente utilizar como criterio de partición Viento.



- Ahora que tenemos seleccionado el criterio, creamos las particiones.

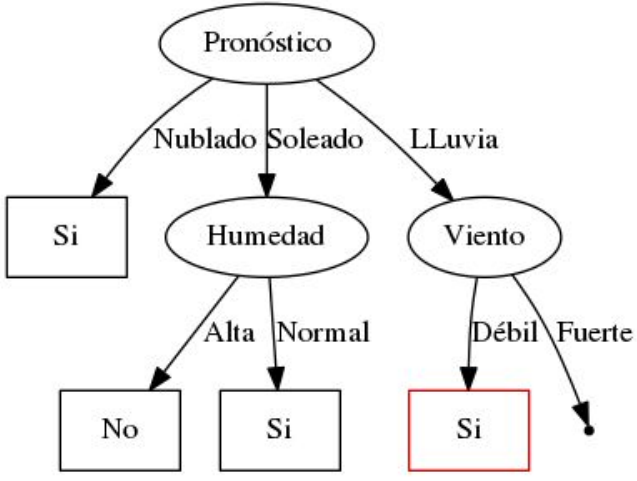


- Aplicamos nuevamente el algoritmo de forma recursiva. Para la sub-partición Viento=Débil tenemos este caso:

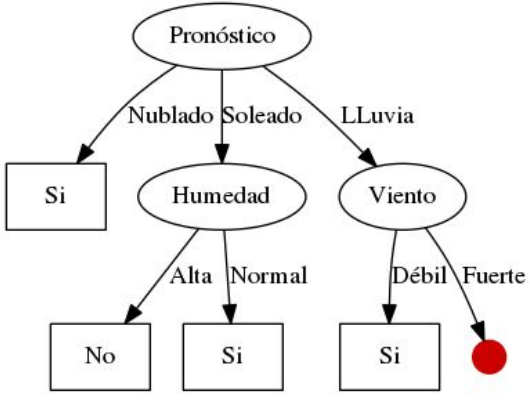


Pronóstico	Temperatura	Humedad	Viento	Jugar
LLuvia	Media	Alta	Débil	Si
LLuvia	Baja	Normal	Débil	Si
LLuvia	Media	Normal	Débil	Si

- Como podemos ver que todos los registros pertenecen a la clase "Si", sabemos que será un nodo hoja con la etiqueta "Si"

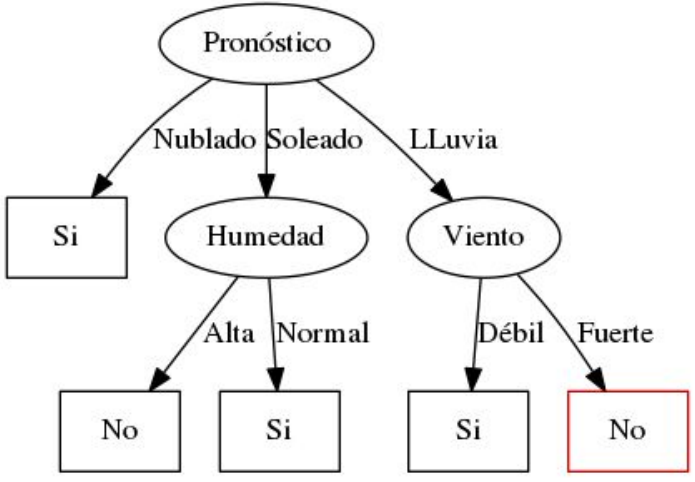


- Para la sub-partición Viento=Fuerte tenemos este caso:



Pronóstico	Temperatura	Humedad	Viento	Jugar
LLuvia	Baja	Normal	Fuerte	No
LLuvia	Media	Alta	Fuerte	No

- Como podemos ver que todos los registros pertenecen a la clase "No", sabemos que será un nodo hoja con la etiqueta "No"



Algoritmo ID3

- Pseudocódigo del algoritmo ID3 de árboles de decisión:

```
ID3 (Ejemplos, Atributo_objetivo, Atributos_canidatos)
  Si todos los ejemplos tienen el mismo valor para el Atributo_objetivo:
    Retornar un solo nodo con etiqueta = ese valor
  Si la lista de Atributos_candidatos está vacía:
    Retornar un solo nodo con etiqueta = valor más común para el Atributo_objetivo en los ejemplos.
  Sino:
    A ← El atributo que mejor clasifica los ejemplos (mayor ganancia de información)
    Crear nodo padre, para la evaluación del atributo A
    Para cada posible valor v_i, de A:
      Agregar una nueva rama debajo del nodo padre, correspondiente a la condición A = v_i.
      Hacer Ejemplos(v_i) el subconjunto de Ejemplos que tienen el valor v_i para A
      Si Ejemplos(v_i) está vacío:
        Debajo de esta nueva rama agregar un nodo hoja con etiqueta = valor más común en los Ejemplos
      Sino:
        Debajo de esta nueva rama agregar un sub-árbol = ID3 (Ejemplos(v_i), Atributo_objetivo, Atributos_candidatos – {A})
    Retornar nodo padre
  Fin
```