# Árboles y reglas de decisión

Joaquín Jesús Pineda Gutiérrez

Luis Garrido Morillo

# Procesamiento de datos sobre la supervivencia en el Titanic

- Datos sin procesar, en los cuales pueden faltar ciertos campos.
- Se han extraído los datos referentes al sexo del pasajero, la clase en la que iba y con la edad.
- Para la edad, se ha considerado ‘adulto’ al pasajero si la edad es mayor a 13 años, para cualquier otro caso ‘niño’.
- En los casos en los que la edad es desconocida (NA) se ha considerado la edad media de los pasajeros.

```
"row.names","pclass","survived","name","age","embarked","home.dest","room","ticket","boat","sex"
"1","1st",1,"Allen, Miss Elisabeth Walton",29.0000,"Southampton","St Louis, MO","B-5","24160 L221
"2","1st",0,"Allison, Miss Helen Loraine", 2.0000,"Southampton","Montreal, PQ / Chesterville, ON"
"3","1st",0,"Allison, Mr Hudson Joshua Creighton",30.0000,"Southampton","Montreal, PQ / Chesterv
"4","1st",0,"Allison, Mrs Hudson J.C. (Bessie Waldo Daniels)",25.0000,"Southampton","Montreal, PQ
"5","1st",1,"Allison, Master Hudson Trevor", 0.9167,"Southampton","Montreal, PQ / Chesterville, C
"6","1st",1,"Anderson, Mr Harry",47.0000,"Southampton","New York, NY","E-12","","3","male"
"7","1st",1,"Andrews, Miss Kornelia Theodosia",63.0000,"Southampton","Hudson, NY","D-7","13502 L7
"8","1st",0,"Andrews, Mr Thomas, jr",39.0000,"Southampton","Belfast, NI","A-36","","","male"
"9","1st",1,"Appleton, Mrs Edward Dale (Charlotte Lamson)",58.0000,"Southampton","Bayside, Queens
"10","1st",0,"Artagaveytia, Mr Ramon",71.0000,"Cherbourg","Montevideo, Uruguay","","","(22)","mal
"11","1st",0,"Astor, Colonel John Jacob",47.0000,"Cherbourg","New York, NY","","17754 L224 10s 6d
"12","1st",1,"Astor, Mrs John Jacob (Madeleine Talmadge Force)",19.0000,"Cherbourg","New York, NY
"13","1st",1,"Aubert, Mrs Leontine Pauline",NA,"Cherbourg","Paris, France","B-35","17477 L69 6s",
"14","1st",1,"Barkworth, Mr Algernon H.",NA,"Southampton","Hessle, Yorks","A-23","","B","male"
"15","1st",0,"Baumann, Mr John D.",NA,"Southampton","New York, NY","","","","male"
"16","1st",1,"Baxter, Mrs James (Helene DeLaudeniere Chaput)",50.0000,"Cherbourg","Montreal, PQ",
"17","1st",0,"Baxter, Mr Quigg Edmond",24.0000,"Cherbourg","Montreal, PQ","B-58/60","","","male"
"18","1st",0,"Beattie, Mr Thomson",36.0000,"Cherbourg","Winnipeg, MN","C-6","","","male"
"19","1st",1,"Beckwith, Mr Richard Leonard",37.0000,"Southampton","New York, NY","D-35","","5","m
"20","1st",1,"Beckwith, Mrs Richard Leonard (Sallie Monypeny)",47.0000,"Southampton","New York, N
"21","1st",1,"Behr, Mr Karl Howell",26.0000,"Cherbourg","New York, NY","C-148","","5","male"
"22","1st",0,"Birnbaum, Mr Jakob",25.0000,"Cherbourg","San Francisco, CA","","","(148)","male"
"23","1st",1,"Bishop, Mr Dickinson H.",25.0000,"Cherbourg","Dowagiac, MI","B-49","","7","male"
"24","1st",1,"Bishop, Mrs Dickinson H. (Helen Walton)",19.0000,"Cherbourg","Dowagiac, MI","B-49",
"25","1st",1,"Bjornstrm-Steffansson, Mr Mauritz Hakan",28.0000,"Southampton","Stockholm, Sweden /
"26","1st",0,"Blackwell, Mr Stephen Weart",45.0000,"Southampton","Trenton, NJ","","","(241)","mal
"27","1st",1,"Blank, Mr Henry",39.0000,"Cherbourg","Glen Ridge, NJ","A-31","","7","male"
"28","1st",1,"Bonnell, Miss Caroline",30.0000,"Southampton","Youngstown, OH","C-7","","8","female
"29","1st",1,"Bonnell, Miss Elizabeth",58.0000,"Southampton","Birkdale, England Cleveland, Ohio",
"30","1st",0,"Borebank, Mr John James",NA,"Southampton","London / Winnipeg, MB","D-21/2","","","m
```

# Procesamiento de datos sobre la supervivencia en el Titanic

- Para el procesamiento de datos, se ha usado la librería csv de Python, que ofrece una mayor simpleza y limpieza en el código respecto a usar splits y procesar el texto por cadenas.
- Para los conjuntos de entrenamiento, validación y prueba se ha guardado el 60%, 20% y 20% respectivamente del total de ejemplos.
- Todos los conjuntos están correctamente estratificados.

```
atributos = [('clase',['1st','2nd','3rd']),('edad',['niño','adulto']),('genero',['male','female'])]
clasificacion = 'Supervivencia'
clases = ['1','0']
entrenamiento = [['1st', 'adulto', 'female', '0'], ['1st', 'adulto', 'female', '0'], ['1st', 'adulto', 'female', '0'], ['1st', 'adulto', 'female', '0']
validacion = [['1st', 'adulto', 'female', '0'], ['1st', 'adulto', 'female', '1'], ['1st', 'adulto', 'female', '1'], ['1st', 'adulto', 'female', '1'], [
prueba = [['1st', 'adulto', 'female', '0'], ['1st', 'adulto', 'female', '0'], ['1st', 'adulto', 'female', '1'], ['1st', 'adulto', 'female', '1'], ['1st
```

# Árboles de decisión - prepoda

- Durante la fase de entrenamiento, se realiza una poda a partir del grado de clasificación elegido, realizados de la siguiente forma:

$$E(\mathcal{S}) = -\sum_{j=1}^{k} p_j \log_2(p_j)$$

$$T(\mathcal{S}) = 1 - \frac{p_d}{|\mathcal{S}|}$$

$$G(\mathcal{S}) = 1 - \sum_{j=1}^{k} p_j^2$$

```python
def medidas(medida, conjunto):
    result = 0.0
    if medida=="entropia":
        proporcionClases = proporcionClase(conjunto,True)
        for p in proporcionClases:
            result = result + proporcionClases[p]*math.log2(proporcionClases[p])
        result = -result

    elif medida=="error":
        proporcionClases = proporcionClase(conjunto)
        result = 1 - proporcionClases[max(proporcionClases, key=proporcionClases.get)] / len(conjunto)

    elif medida=="gini":
        proporcionClases = proporcionClase(conjunto,True)
        for p in proporcionClases:
            result = result + proporcionClases[p]*proporcionClases[p]
        result = 1 - result

    return result
```

# Árboles de decisión - prepoda

- Para realizar el grado de clasificación de un atributo, se realiza llamando a la función anterior para cada valor y el subconjunto de ejemplos que aborde, dependiendo de la medida que se aplique:

$$E(\mathcal{S} \Rightarrow \mathcal{S}_1 \dots \mathcal{S}_r) = \sum_{i=1}^{r} \frac{|\mathcal{S}_i|}{|\mathcal{S}|} E(\mathcal{S}_i)$$

$$T(\mathcal{S} \Rightarrow \mathcal{S}_1 \dots \mathcal{S}_r) = \sum_{i=1}^{r} \frac{|\mathcal{S}_i|}{|\mathcal{S}|} T(\mathcal{S}_i)$$

$$G(\mathcal{S} \Rightarrow \mathcal{S}_1 \dots \mathcal{S}_r) = \sum_{i=1}^{r} \frac{|\mathcal{S}_i|}{|\mathcal{S}|} G(\mathcal{S}_i)$$

```
for valor in valoresAtributo:
    subconjunto = subconjuntoValorAtributo(conjunto,indices[nombreAtributo],valor)
    medidaValor = medidaValor + medidas(medida,subconjunto)
    medidaValor = medidaValor*len(subconjunto)/len(conjunto)
```

# Aprendizaje de árboles

- Recorriendo los ejemplos y generando el árbol, se establece que se llega a un nodo hoja con valor de clasificación cuando:
  - Hay uniformidad en el valor de clasificación del subconjunto de ejemplos.
  - Cuando no quedan más atributos que recorrer.
  - Cuando no quedan más ejemplos que recorrer.
- En caso contrario, es un nodo interior y se crean los siguientes nodos de forma recursiva.

```
arbol = ClasificadorDT(Titanic.clasificacion,Titanic.clases,Titanic.atributos)

arbol.imprime()
clasificadores.clasificador.ClasificadorNoEntrenado(Exception)

arbol.entrena(Titanic.entrenamiento)
```

# Clasificador de ejemplos

- Con un ejemplo de entrada, se devuelve su valor de clasificación a partir del árbol previamente aprendido.

```
arbol.clasifica(['1st','adulto','female'])
'1'

arbol.clasifica(['2nd','niño','female'])
'1'


arbol.clasifica(['s','s','s','n','n','?','s','s','s','s','n','n','n','n','s','?'])
'demócrata'

arbol.clasifica(['?','s','n','n','n','n','s','s','s','s','s','n','n','s','s','s'])
'demócrata'


arbol.clasifica(['laboral','uno','una','ninguno','viudo','altos'])
'conceder'

arbol.clasifica(['parado','uno','una','uno','viudo','bajos'])
'no conceder'
```

# Evaluación del árbol

- A partir de un conjunto de ejemplos de entrada, se determina el rendimiento del árbol con el número de ejemplos cubiertos por el árbol entre el número total de ejemplos.

```
arbol.evalua(Titanic.validacion)
0.8326996197718631

arbol.evalua(Titanic.prueba)
0.8275862068965517


arbol.evalua(Votos.validacion)
0.9710144927536232

arbol.evalua(Votos.prueba)
0.896551724137931


arbol.evalua(Prestamos.validacion)
0.9382716049382716

arbol.evalua(Prestamos.prueba)
0.9079754601226994
```

# Representación de árboles

- La impresión se realiza de manera recursiva dependiendo de la profundidad de los nodos en el árbol.
- La representación es la siguiente:
  - atributo: (valor del atributo) -> [valor de clasificación]
  - A medida que se va bajando y hacia la derecha, se representan los nodos interiores hasta llegar a los nodos hoja, representados por [valor de clasificación]

```
In [44]: arbol.imprime()

edad: (niño)
        clase: (1st)
                genero: (male) -> [1]

                genero: (female) -> [0]

        clase: (2nd) -> [1]

        clase: (3rd)
                genero: (male) -> [0]

                genero: (female) -> [0]

edad: (adulto)
        clase: (1st)
                genero: (male) -> [0]

                genero: (female) -> [1]

        clase: (2nd)
                genero: (male) -> [0]

                genero: (female) -> [1]

        clase: (3rd)
                genero: (male) -> [0]

                genero: (female) -> [0]
```

# Podado de árboles

- Con el árbol aprendido en el conjunto de entrenamiento, se realiza una poda a partir de la poda de nodos interiores (reemplazándolos por nodos hoja con el valor de clasificación más frecuente de su distribución de clases) y se mide el rendimiento del árbol podado.
- Si el rendimiento es mejor con respecto al árbol original, se vuelve a realizar la poda sobre el árbol podado, así continuamente hasta que no se encuentre un árbol con mejor rendimiento.
- Para facilitar el proceso de poda, se generan los caminos posibles de los nodos interiores del árbol y se trabaja a partir de ellos.

```
In [44]: arbol.imprime()

edad: (niño)
        clase: (1st)
                genero: (male) -> [1]

                genero: (female) -> [0]

        clase: (2nd) -> [1]

        clase: (3rd)
                genero: (male) -> [0]

                genero: (female) -> [0]

edad: (adulto)
        clase: (1st)
                genero: (male) -> [0]

                genero: (female) -> [1]

        clase: (2nd)
                genero: (male) -> [0]

                genero: (female) -> [1]

        clase: (3rd)
                genero: (male) -> [0]

                genero: (female) -> [0]
```

```
[['niño', '1st'],
 ['niño', '3rd'],
 ['adulto', '1st'],
 ['adulto', '2nd'],
 ['adulto', '3rd'],
 ['niño'],
 ['adulto']]
```

# Podado de árboles

- Se generan árboles más reducidos y/o con mejor rendimiento en los conjuntos:

```
In [44]: arbol.imprime()

edad: (niño)
        clase: (1st)
                genero: (male) -> [1]

                genero: (female) -> [0]

        clase: (2nd) -> [1]

        clase: (3rd)
                genero: (male) -> [0]

                genero: (female) -> [0]

edad: (adulto)
        clase: (1st)
                genero: (male) -> [0]

                genero: (female) -> [1]

        clase: (2nd)
                genero: (male) -> [0]

                genero: (female) -> [1]

        clase: (3rd)
                genero: (male) -> [0]

                genero: (female) -> [0]
```

```
In [77]: arbolPoda = ClasificadorDTPoda(Titanic.clasificacion, Titanic.clases, Titanic.atributos)

In [78]: arbolPoda.entrena(Titanic.entrenamiento,Titanic.validacion)

In [79]: arbolPoda.imprime()

edad: (niño)
        clase: (1st) -> [1]

        clase: (2nd) -> [1]

        clase: (3rd) -> [0]

edad: (adulto)
        clase: (1st)
                genero: (male) -> [0]

                genero: (female) -> [1]

        clase: (2nd)
                genero: (male) -> [0]

                genero: (female) -> [1]

        clase: (3rd) -> [0]
```

```
arbol.evalua(Titanic.validacion)
0.8326996197718631

arbol.evalua(Titanic.prueba)
0.8275862068965517
```

```
In [80]: arbolPoda.evalua(Titanic.validacion)
Out[80]: 0.8326996197718631

In [81]: arbolPoda.evalua(Titanic.prueba)
Out[81]: 0.8275862068965517
```

# Podado de árboles

```
In [87]: arbolPoda = ClasificadorDTPoda(Votos.clasificacion, Votos.clases, Votos.atributos)

In [88]: arbolPoda.entrena(Votos.entrenamiento,Votos.validacion)

In [89]: arbolPoda.imprime()

voto4: (s) -> [republicano]

voto4: (n) -> [demócrata]

voto4: (?) -> [demócrata]
```

```
arbol.evalua(Votos.validacion)
0.9710144927536232

arbol.evalua(Votos.prueba)
0.896551724137931
```

```
In [90]: arbolPoda.evalua(Votos.validacion)
Out[90]: 0.9855072463768116

In [91]: arbolPoda.evalua(Votos.prueba)
Out[91]: 0.9080459770114943
```

# Podado de árboles

```
arbol.evalua(Prestamos.validacion)
0.9382716049382716

arbol.evalua(Prestamos.prueba)
0.9079754601226994
```

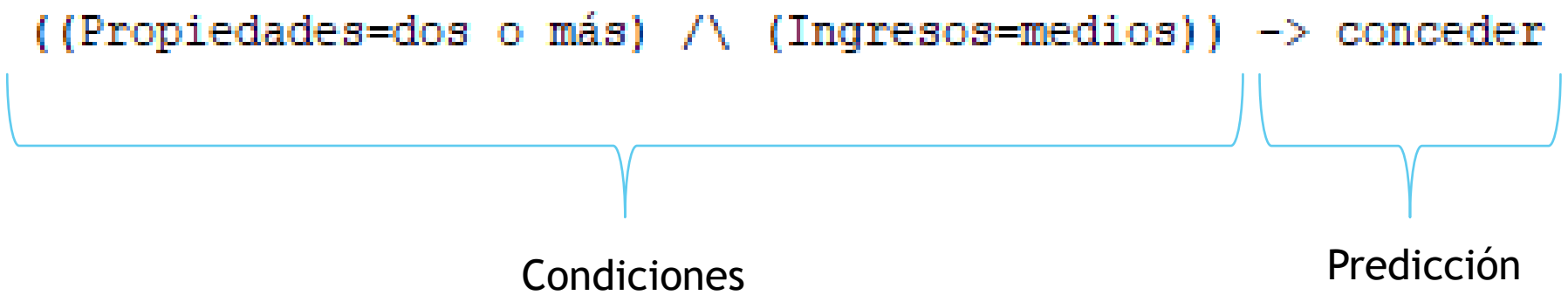


```
arbolPoda.evalua(Prestamos.validacion)
1.0

arbolPoda.evalua(Prestamos.prueba)
0.9877300613496932
```

```
In [84]: arbolPoda.imprime()

Ingresos: (bajos)
        Empleo: (parado) -> [no conceder]

        Empleo: (funcionario)
                Propiedades: (ninguna) -> [no conceder]

                Propiedades: (una) -> [estudiar]

                Propiedades: (dos o más) -> [conceder]

        Empleo: (laboral)
                Productos: (ninguno) -> [no conceder]

                Productos: (uno) -> [no conceder]

                Productos: (dos o más) -> [estudiar]

        Empleo: (jubilado) -> [no conceder]

Ingresos: (medios)
        Propiedades: (ninguna)
                Empleo: (parado) -> [no conceder]

                Empleo: (funcionario) -> [estudiar]

                Empleo: (laboral) -> [estudiar]

                Empleo: (jubilado) -> [no conceder]

        Propiedades: (una)
                Productos: (ninguno) -> [no conceder]

                Productos: (uno) -> [estudiar]

                Productos: (dos o más)
                        Hijos: (ninguno) -> [estudiar]

                        Hijos: (uno) -> [estudiar]

                        Hijos: (dos o más) -> [no conceder]

        Propiedades: (dos o más) -> [conceder]

Ingresos: (altos)
        Empleo: (parado) -> [estudiar]

        Empleo: (funcionario) -> [conceder]

        Empleo: (laboral) -> [conceder]

        Empleo: (jubilado) -> [estudiar]
```

# Reglas

- Las reglas son una forma de clasificador que genera una prediccion dela siguiente forma:



# Reglas - Implementación

- Las reglas han sido implementadas como una clase que encapsula una lista de tuplas con el índice del atributo y el valor escogido.

- Para evaluar si una instancia cumple una regla solo tenemos que mirar si los atributos de la instancia poseen los mismos valores que los de las condiciones de la regla. En caso afirmativo las instancia será de la misma categoría de la regla.

```
class ReglaDR():

    def __init__(self, categoria):
        self.categoria = categoria
        self.reglas = []

    def addRule(self, atributo):
        self.reglas.append(atributo)

    def getRules(self):
        return self.reglas

    def getCategoria(self):
        return self.categoria

    ''' Metodo que dado un elemento de un conjunto evalua las condiciones de
    esta regla para ese elemento.
    Si el parametro positivo es True se mira primero que la clase esperada para el
    elemento sea del mismo tipo que precide esta regla. Esto es importante para el
    calculo de la frecuencia relativa durante la fase de entrenamiento.'''
    def evaluate(self, elemento, positivo=False):
        clase = elemento[-1]
        if positivo and self.categoria != clase:
            return False
        for atributo, valor in self.reglas:
            if elemento[atributo] != valor:
                return False
        return True

    def getFrecuenciaRelativa(self, conjunto):
        return self.getCountPositivos(conjunto) / self.getCountValidos(conjunto)

    ''' Metodo que devuelve el numero de elementos que cumplen la regla y ademas
    poseen la misma clase que precide la regla'''
    def getCountPositivos(self, conjunto):
        count = 0
        for elemento in conjunto:
            if self.evaluate(elemento, True):
                count += 1
        return count

    ''' Metodo que devuelve el numero de elementos de un conjunto que cumplen la regla
    aunque la prediccion de la clase no sea la esperada.'''
    def getCountValidos(self, conjunto):
        count = 0
        for elemento in conjunto:
            if self.evaluate(elemento, False):
                count += 1
        return count
```

# Reglas - Aprendizaje

- El entrenamiento de las reglas se basa en los siguientes pasos:
  1. Elegir una categoría para la regla
  2. Generar una regla vacía para dicha categoría
  3. Añadir un par atributo-valor a las condiciones de la regla
  4. Eliminar los elementos no cubiertos por la regla
  5. Repetir 3 y 4 hasta que la frecuencia relativa sea 1 o no queden más pares de atributo-valor que elegir
  6. Devolver la regla y volver a 1 eligiendo otra categoría hasta que lleguemos a la ultima.
  7. En la ultima categoría devolvemos una regla vacía.

# Reglas - Aprendizaje

- Finalmente obtenemos un conjunto de reglas que clasifican todos los casos (bien o mal) del conjunto de entrenamiento. Por ejemplo en el Titanic:

```
Reglas obtenidas:
 ((genero=female) /\ (clase=1st) /\ (edad=adulto)) -> 1
((edad=niño) /\ (clase=2nd)) -> 1
((genero=female) /\ (clase=2nd) /\ (edad=adulto)) -> 1
((genero=female) /\ (edad=adulto) /\ (clase=3rd)) -> 1
((clase=1st) /\ (edad=niño) /\ (genero=male)) -> 1
((clase=1st) /\ (edad=adulto) /\ (genero=male)) -> 1
((genero=female) /\ (clase=3rd) /\ (edad=niño)) -> 1
((clase=3rd) /\ (edad=adulto) /\ (genero=male)) -> 1
((clase=2nd) /\ (edad=adulto) /\ (genero=male)) -> 1
En otro caso -> 0
```

# Reglas - Podado

- No siempre el conjunto de reglas obtenidos es bueno. En el ejemplo anterior del titanic nos decía que los hombres de 3 clase adultos sobreviven lo cual no se ajusta a la realidad de lo que pasó.
- Aquí entra el podado de reglas

# Reglas - Podado

- El podado de reglas se compone de los siguientes pasos:
  1. Guardamos una copia del conjunto original de reglas como la mejor opción.
  2. Cogemos una regla del conjunto (la ultima normalmente).
  3. Eliminamos una condición de la regla escogida. Si era la única condición eliminamos la regla al completo.
  4. Evaluamos el nuevo conjunto de reglas con el conjunto de instancias de validación.
  5. Si estamos mejorando guardamos ese conjunto de reglas como nueva mejor opción. En caso contrario mantenemos el que tuviéramos.
  6. Repetimos 2, 3, 4, 5 hasta que no nos quede ninguna regla salvo la regla por defecto (conjunto vacío).
  7. Devolvemos el conjunto elegido como mejor opción y ese será nuestro nuevo conjunto de reglas para el clasificador.

# Reglas - Podado

- Tras la poda habremos empeorado el porcentaje de clasificación del conjunto de entrenamiento pero a su vez habremos mejorado la clasificación del conjunto de validación y eliminado el sobreajuste en el conjunto de reglas.
- Para el Titanic este es el nuevo conjunto, que se ajusta más a la realidad:

```
Reglas obtenidas:
 ((genero=female) /\ (clase=1st) /\ (edad=adulto)) -> 1
((edad=niño) /\ (clase=2nd)) -> 1
((genero=female) /\ (clase=2nd)) -> 1
En otro caso -> 0
```