APRENDIZAJE DE REGLAS

Eduardo Morales y Jesús González

Clasificación con Reglas

- Resultados fáciles de entender
  - En dominios en que se trabaja de cerca con expertos en otras áreas
- Reglas del tipo
  - **If** $\langle att_1 = val_1 \ \& \ att_5 = val_5 \rangle$ **then** $class_i$

Splitting vs. Covering

- Árboles de decisión
  - Basados en splitting
- Splitting
  - Dividir conjunto de datos en subconjuntos
  - Selecciona un atributo con una heurística
  - Se consideran todas las clases en la partición
  - Se añaden atributos al árbol tratando de maximizar la separación entre las clases

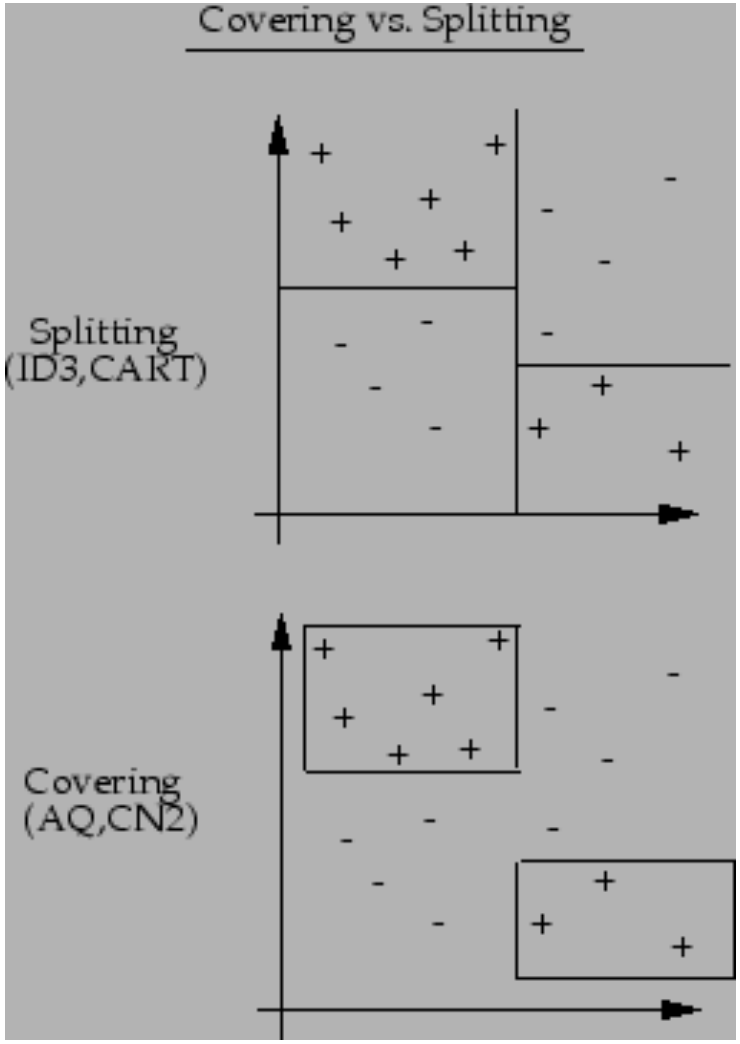
Splitting vs. Covering

- Aprendizaje de Reglas
  - Basado en covering
- Covering
  - Encontrar condiciones (par atributo valor)
    - Cubrir mayor # ejemplos de una clase
    - Cubrir menor # ejemplos del resto de las clases
    - Considera cubrir sólo una clase

Splitting vs. Covering

- Idea básica
  - Añadir pruebas a cada regla que construye
    - Busca maximizar covertura, minimizando errores
- Reglas
  - Pueden expresar ***disjunciones*** más fácil que los árboles
  - Extraer reglas de árboles tiende a producir reglas más complejas de lo necesario
  - Los árboles tienen algo conocido como **replicated subtree problem**
    - Se repiten subárboles en varios lados

Splitting vs. Covering



Listas de Decisión

- Normalmente se generan listas de decisión (***decision lists***)
  - Reglas evaluadas en orden
- Facilita evaluación pero disminuye su modularidad
- Si se permitiese evaluar reglas independientemente del orden
  - Puede haber más de una predicción para un solo ejemplo
  - Dificulta producir un solo resultado

Splitting vs. Covering

- Reglas (continuación…)
  - Construir árboles a partir de reglas
    - No trivial
    - Tiende a dejar árboles incompletos
  - Ventaja de las reglas sobre los árboles
    - Tienden a representar "**pedazos**" de conocimiento relativamente *independiente*

1R

- 1R es un sistema muy simple, equivalente a un ***decision stump***, o árbol de decisión de un solo nivel
  - Se hacen reglas que prueban un solo par atributo-valor
  - Se prueban todos los pares atributo-valor
  - Se selecciona el que ocasione menos errores
  - En la tabla 2.2
    - Atributo Ambiente, 4 errores
    - Atributo Temperatura, 5 errores
    - Atributo Humedad, 4 errores
    - Atributo Viento, 5 errores
  - Empates se rompen arbitrariamente

1R

Tabla 2.2: Tabla de ejemplos para decidir si jugar o no golf.

Ambiente	Temp.	Humedad	Viento	Clase
soleado	alta	alta	no	N
soleado	alta	alta	si	N
nublado	alta	alta	no	P
lluvia	media	alta	no	P
lluvia	baja	normal	no	P
lluvia	baja	normal	si	N
nublado	baja	normal	si	P
soleado	media	alta	no	N
soleado	baja	normal	no	P
luvia	media	normal	no	P
soleado	media	normal	si	P
nublado	media	alta	si	P
nublado	alta	normal	no	P
lluvia	media	alta	si	N

1R

- Nos quedamos con las siguientes reglas:
- If Ambiente = soleado

  Then Clase = N (cubre 5 y tiene 2 errores)
- If Ambiente = nublado

  Then Clase = P (cubre 4 y tiene 0 errores)
- If Ambiente = lluvioso

  Then Clase = P (cubre 5 y tiene 2 errores)

1R

- Valores faltantes se tratan como un nuevo valor
- Para valores continuos se hace una división simple
  - Se ordenan atributos respecto a la clase
  - Se sugieren puntos de partición en cada lugar donde cambia la clase
  - Si hay dos clases diferentes con el mismo valor
    - Se mueve el punto de partición a un punto intermedio con el sig. valor hacia arriba o abajo dependiendo de dónde está la clase mayoritaria
  - Problema más serio, el algoritmo tendería a construir reglas para cada una de las particiones
    - Da una clasificación perfecta
    - Con muy poca predicción futura
  - Para solucionar el problema
    - Se exige un # mínimo de ejemplos de la clase mayoritaria en cada partición
  - Si hay clases adyacentes con la misma clase mayoritaria
    - Se juntan

Ejemplo

64	65	68	69	70	71	72	72	75	75	80	81	83	85
P	N	P	P	P	N	N	P	P	P	N	P	P	N

- Tomando puntos intermedios
  - 64.5, 66.5, 70.5, 72, 77.5, 80.5 y 84
- Considerando los ejemplos de dif. clase, podemos mover la frontera de 72 a 73.5

64	65	68	69	70	71	72	72	75	75	80	81	83	85
P	N	P	P	P	N	N	P	P	P	N	P	P	N

Ejemplo

- Si tomamos al menos 3 elementos por partición

64	65	68	69	70	71	72	72	75	75	80	81	83	85
P	N	P	P	P	N	N	P	P	P	N	P	P	N

- Si juntamos clases con la misma clase mayoritaria

64	65	68	69	70	71	72	72	75	75	80	81	83	85
P	N	P	P	P	N	N	P	P	P	N	P	P	N

Ejemplo

- Obtenemos una regla del tipo:
  - If Temperatura ≤ 77.5

    Then Clase = P (cubre 10 y tiene 3 errores)
  - If Temperatura > 77.5

    Then Clase = N (cubre 4 y tiene 2 errores)

64	65	68	69	70	71	72	72	75	75	80	81	83	85
P	N	P	P	P	N	N	P	P	P	N	P	P	N

- En la segunda regla se hace una selección al azar por haber un empate

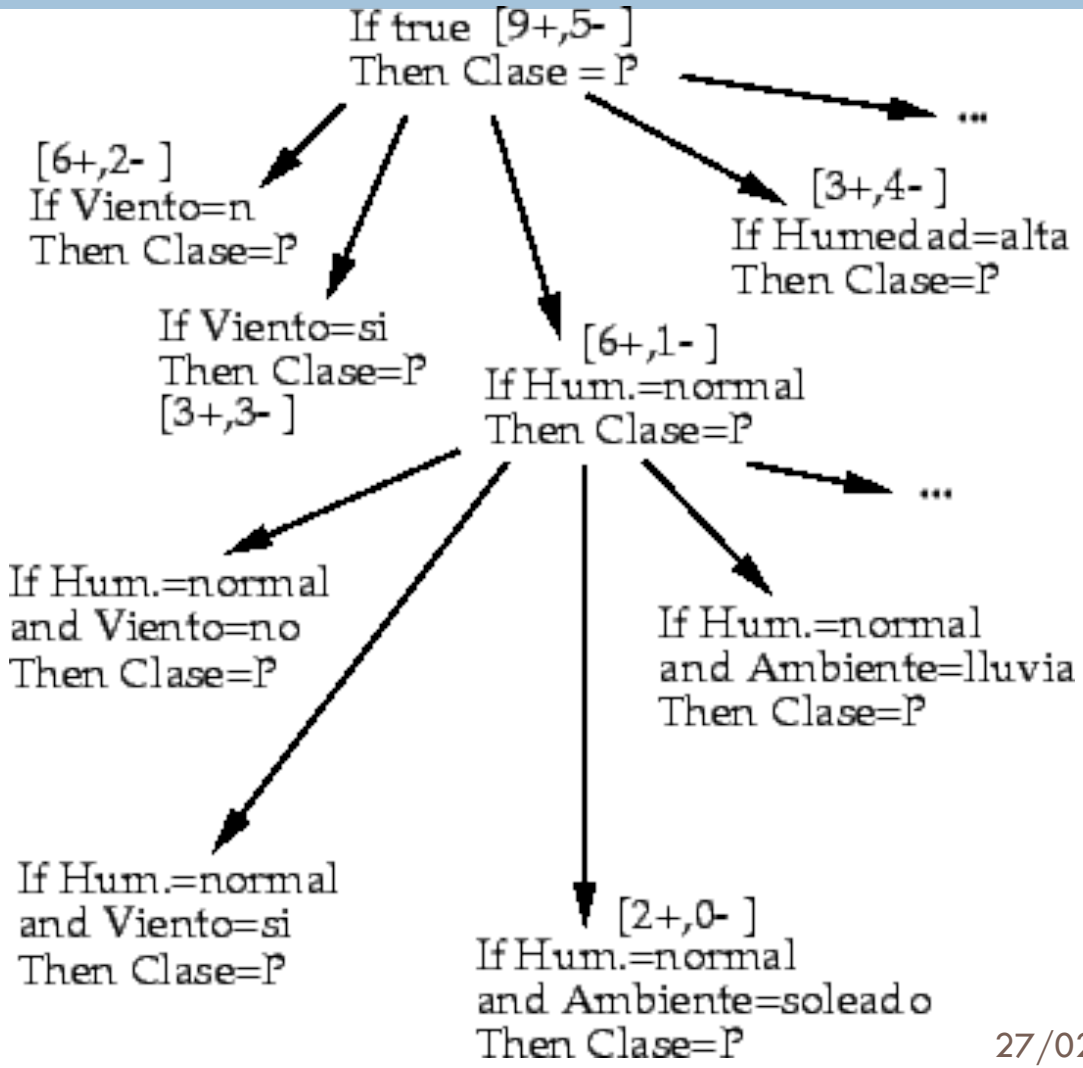
Algoritmo 1R


```
Para cada atributo
    Para cada valor de cada atributo, crea una regla:
        cuenta cuántas veces ocurre la clase
        encuentra la clase más frecuente
        asigna esa clase a la regla
    Calcula el error de todas las reglas
Selecciona las reglas con el error más bajo
```

PRISM

- PRISM es un algoritmo básico de aprendizaje de reglas
  - Asume que no hay ruido en los datos

PRISM


If true [9+,5-]
Then Clase = P
...
[6+,2-]
If Viento=n
Then Clase=P
[3+,4-]
If Humedad=alta
Then Clase=P
If Viento=si
Then Clase=P
[3+,3-]
[6+,1-]
If Hum.=normal
Then Clase=P
...
If Hum.=normal
and Viento=no
Then Clase=P
If Hum.=normal
and Ambiente=lluvia
Then Clase=P
If Hum.=normal
and Viento=si
Then Clase=P
[2+,0-]
If Hum.=normal
and Ambiente=soleado
Then Clase=P

Algoritmo de PRISM

Para cada clase C
  Sea E = ejemplos de entrenamiento
  Mientras E tenga ejemplos de clase C
    Crea una regla R con LHS vacío y clase C
    Until R es perfecta **do**
      Para cada atributo A no incluido en R y cada valor v,
        Considera añadir la condición $A = v$ al LHS de R
        Selecciona el par $A = v$ que maximice p/t
        (en caso de empates, selecciona la que tenga p mayor)
      Añade $A = v$ a R
    Elimina de E los ejemplos cubiertos por R

PRISM

- Sea ***t*** el # total de ejemplos que cumplen la condición (par atributo-valor) y ***p*** el # de ejemplos positivos (o la clase en turno) cubiertos por la regla
  - PRISM añade condiciones a reglas que maximicen la relación ***p/t***
    - Relación entre ejemplos ***pos*** cubiertos y ejemplos cubiertos en total **sea mayor**

$$\frac{p}{t} = \frac{\text{positivos cubiertos}}{\text{tobal cubiertos}}$$

PRISM

- El algoritmo va eliminando los ejemplos que va cubriendo cada regla
  - Las reglas construidas tienen que interpretarse en orden
    - Nuevas reglas se diseñan para cubrir los casos faltantes
  - Reglas que dependen del orden para su interpretación → **decision lists**
  - Reglas que no dependen del orden son más modulares, pero pueden producir varias clasificaciones o no predecir nada
  - Con varias clasificaciones se puede seleccionar la regla que cubra más ejemplos
  - Si no hay una clasificación, elegir la clase mayoritaria
  - Las reglas ordenadas son en general más rápidas de producir ya que van reduciendo el # de ejemplos a considerar

Ejemplo de PRISM

A1	A2	A3	A4	Clase
1	X	Triang	a	P
0	X	Circ	a	N
1	Y	Cuadr	a	P
1	Y	Triang	b	P
1	X	Cuadr	b	N
0	Y	Circ	a	P
0	X	Triang	b	N
1	Y	Circ	a	P

Ejemplo de PRISM

- Iniciamos con ***P***
  - Construimos todas posibles combinaciones de atributo valor y evaluamos su predicción sobre la clase ***P***
    - If A_1 = 1 Then Clase = P, 4/5
    - If A_1 = 0 Then Clase = P, 1/3
    - If A_2 = x Then Clase = P, 1/4
    - If A_2 = y Then Clase = P, 4/4
    - If A_3 = triang Then Clase = P, 2/3
    - If A_3 = circ Then Clase = P, 2/3
    - If A_3 = cuadr Then Clase = P, 1/2
    - If A_4 = a Then Clase = P, 4/5
    - If A_4 = b Then Clase = P, 1/3

Ejemplo de PRISM

- Hay una regla perfecta (*If* $A_2 = y$ *Then Clase* = ***P***)
  - La seleccionamos y eliminamos ejemplos cubiertos

Ejemplo de PRISM

A1	A2	A3	A4	Clase
1	X	Triang	a	P
0	X	Circ	a	N
1	X	Cuadr	b	N
0	X	Triang	b	N

Ejemplo de PRISM

- Repetimos con los ejemplos que quedan
  - If $A_1 = 1$ Then Clase = P, 1/2
  - If $A_1 = 0$ Then Clase = P, 0/2
  - If $A_2 = x$ Then Clase = P, 1/4
  - If $A_3 =$ triang Then Clase = P, 1/2
  - If $A_3 =$ circ Then Clase = P, 0/1
  - If $A_3 =$ cuadr Then Clase = P, 0/1
  - If $A_4 = a$ Then Clase = P, 1/2
  - If $A_4 = b$ Then Clase = P, 0/2

Ejemplo de PRISM

- Tenemos tres empates de ½ como valor máximo
- Tomamos uno al azar y construimos todas las posibles reglas añadiendo posibles pares atributo-valor
  - If $A_1 = 1$ & $A_2 = x$ Then Clase = P, 1/2
  - If $A_1 = 1$ & $A_3 = triang$ Then Clase = P, 1/1
  - If $A_1 = 1$ & $A_3 = circ$ Then Clase = P, 0/0
  - If $A_1 = 1$ & $A_3 = cuadr$ Then Clase = P, 0/1
  - If $A_1 = 1$ & $A_4 = a$ Then Clase = P, 1/1
  - If $A_1 = 1$ & $A_4 = b$ Then Clase = P, 0/1

Ejemplo de PRISM

- De nuevo tenemos empates, tomamos la primera regla
- Las reglas entonces para la clase **P** son:
  - If A_2 = y Then Clase = **P**
  - If A_1 = 1 & A_3 = Triang Then Clase = **P**
- Se repite el mismo proceso para el resto de las clases
- Para la clase **N** unas posibles reglas son:
  - If A_2 = x & A_1 = 0 Then Clase **N**
  - If A_2 = x & A_3 = Cuadr Then Clase = **N**

AQ

- AQ fue uno de los primeros sistemas de reglas
  - Desarrollado originalmente por Michalski (79)
  - Reimplementado y mejorado por otros autores
    - AQ11, AQ15
  - Su salida es un conjunto de reglas de clasificación del tipo “if…then…”

AQ

- Idea principal de AQ
  - Seleccionar un ejemplo semilla aleatoriamente
  - Identificar el ejemplo de otra clase más cercano
  - Especializar (añadir condiciones atributo valor) a la regla actual
    - Para no cubrir ese ejemplo negativo y
    - Tratar de cubrir la mayor cantidad de ejemplos positivos

AQ

- Se exploran varias alternativas de reglas (beam-search)
- Algunas heurísticas utilizadas para seleccionar la mejor regla
  - Sumar ejemplos **pos** cubiertos y **neg** excluidos (en empate, preferir la más corta)
  - Sumar el # de ejemplos clasificados correctamente dividido por el # total de ejemplos cubiertos
  - Maximiza el número de ejemplos **pos** cubiertos
- Principal desventaja
  - Sensible a ejemplos con ruido (i.e. si la semilla seleccionada tiene inf. errónea)

CN2

- CN2 (Clark, Niblett, 88)
  - Atacar datos con ruido
  - Evitar el sobreajuste encontrado en sistemas como AQ
- Principal contribución
  - Quitar dependencia de ejemplo específico durante su búsqueda
  - Forma base de muchos alg. de reglas actuales

CN2

- En CN2
  - Se pueden especificar valores ***don't-care***
  - Valores desconocidos
  - Sigue búsqueda tipo beam-search
  - Heurística de búsqueda original basada en entropía
    - $Entr = -\sum_i p_i log_2(p_i)$
    - p_i es la distribución de las clases que cubre cada regla

CN2

- Selecciona la regla de menor entropía
  - La que cubre muchos ejemplos de una clase y pocos de las demás
  - Versión posterior utiliza ***Laplacian error estimate***
  - $Accuracy A(n, n_c, k) = (n - n_c + k - 1)/(n + k)$
    - n = número total de ejemplos cubiertos por la regla
    - n_c = número de ejemplos **pos** cubiertos por la regla
    - *k* = número de clases en el problema

CN2

- Medida de significancia para las reglas
  - Umbral para la medida de significancia
    - Debajo de ese nivel, las reglas son rechazadas
  - Medida basada en la razón de verosimilitud (***likelihood ratio statistic***)
    - Mide una distancia entre dos distribuciones

$$2\sum_{i=1}^{n} f_i log(\frac{f_i}{e_i})$$

Likelihood-ratio test

- Prueba estadística utilizada para comparar dos modelos
  - Uno es el modelo nulo y otro es el modelo alternativo
  - La prueba está basada en el "likelihood ratio"
  - Expresa cuantas veces es más probable que los datos caigan en un modelo que en otro
  - El "likelihood ratio" (o su logaritmo) puede ser usado para calcular el "p-value"

$$D = -2\ln\left(\frac{\text{likelihood for null model}}{\text{likelihood for alternative model}}\right)$$

CN2

- F = (f1, …, fn), distribución de frecuencias observada de ejemplos dentro de las clases que satisfacen una regla dada
  - # ejemplos que satisfacen la regla entre # total de ejemplos que satisfacen la regla
- E = (e1, …, en) es la frecuencia esperada en los mismos ejemplos bajo la suposición de que la regla selecciona ejemplos aleatoriamente
  - # ejemplos cubiertos por la regla siguiendo la distribución de ejemplos del total de los ejemplos
- Entre más baja la medida, más posible que la aparente regularidad expresada en la regla sea por casualidad

Medidas Alternativas de Selección

- La más simple es la frecuencia relativa de ejemplos positivos cubiertos
  - Tiene problemas con muestras pequeñas

$$m(R) = \frac{p}{t} = \frac{p}{p+n}$$

  - t = # tot de ejemplos cubiertos por la regla = p + n
  - p = # tot ejemplos pos cubiertos por la regla

Medidas Alternativas de Selección

- Estimador Laplaciano (CN2)
  - Asume una distribución uniforme de las k clases (k=2)

$$m(R) = \frac{p+1}{p+n+k}$$

Medidas Alternativas de Selección

- Estimador ***m***
  - Considera que las distribuciones a priori de las clases "Pa(C)", son independientes del # de clases y ***m*** es dependiente del dominio
    - Entre más ruido, se selecciona una ***m*** mayor

$$m(R) = \frac{p + m \cdot P_a(C)}{p + n + m}$$

Medidas Alternativas de Selección

- Ganancia de Información

$$log_2 \frac{p}{p+n} - log_2 \frac{P}{P+N}$$

- P y N son los ejemplos cubiertos antes de añadir la nueva prueba

Medidas Alternativas de Selección

- Weighted relative accuracy

$$wla = \frac{p+n}{P+N}\left(\frac{p}{p+n} - \frac{P}{P+N}\right)$$

Valores Desconocidos y Numéricos

- Valores desconocidos en un alg. Covering
  - Hacer como si no aparecieran en ninguna condición (ignorarlos)
  - Algoritmos que aprenden "decision lists" tienen ventaja
    - Ejemplos que parecen difíciles al principio se van quedando y al final se pueden resolver
      - En general, más fácilmente
- Atributos numéricos se pueden tratar igual que con árboles de decisión

Pruning

- Forma de evaluar una regla
  - Considerar la probabilidad de que una regla aleatoria nos de resultados iguales o mejores que la regla a considerar
    - Basados en la mejora en ejemplos **pos** cubiertos
- Idea
  - Generar reglas que cubran solo ejemplos **pos**
  - Probable que esas reglas estén sobre-especializadas
  - Se elimina el último término añadido y se verifica si esa regla es mejor a la anterior
  - Se repite el proceso hasta que no existan mejoras

Algoritmo de Podado de Reglas

Inicializa E al conjunto de ejemplos
Until E sea vacío **do**
    Para cada clase C
        Crea una regla perfecta para la clase C
        Calcula la medida de probabilidad $m(R)$ para la regla
            y para la regla sin la última condición $m(R-)$
        Mientras $m(R-) < m(R)$, elimina la última condición
            de la regla y repite el proceso
    De las reglas generadas, selecciona aquella con $m(R)$ menor
    Elimina las instancias cubiertas por la regla

Algoritmo de Podado de Reglas

- Este algoritmo no garantiza encontrar las mejores reglas por 3 razones principales
  - El alg. para construir las reglas, no necesariamente produce las mejores reglas para ser reducidas
  - La reducción de reglas inicia con la última condición, y no necesariamente es el mejor orden
  - La reducción de reglas termina cuando cambia la estimación, lo cual no garantiza que el seguir recortando pueda mejorar de nuevo la estimación
- Sin embargo, el procedimiento es bueno y rápido

Medida de Evaluación para Reducción de Reglas

- ¿Cuál es la probabilidad de que una regla seleccionada aleatoriamente con la misma cantidad de ejemplos que cubre R tenga el mismo desempeño?
- Una regla que cubra ***t*** casos, de ellos ***i*** son de la clase **C** (sin reemplazo):

$$Pr(t_C) = \frac{\binom{P}{i}\binom{T-P}{t-i}}{\binom{T}{t}}$$

Medida de Evaluación para Reducción de Reglas

- ***p*** es # instancias de la clase que la regla selecciona
- ***t*** es # total de instancias que cubre la regla
- ***P*** es el # total de instancias de la clase
- ***T*** es el # total de instancias
- Si queremos ver la probabilidad de que cubra *p* casos o más, entonces:

$$m(R) = \sum_{i=p}^{min(t,P)} Pr(t_C)$$

Medida de Evaluación para Reducción de Reglas

- Valores pequeños indican que la regla es buena
  - Porque es muy poco probable que la regla sea construida por casualidad
  - Como esto es muy costoso de calcular, se hacen aproximaciones
  - Si el # de ejemplos es grande, la prob. de que exactamente *i* ejemplos de los *t* sean de clase C (con reemplazo) es:

$$Pr(t_C) = \left(\begin{array}{c} t \\ i \end{array} \right) (\frac{P}{T})^i (1 - \frac{P}{T})^{t-i}$$

  - corresponde a una distribución binomial

Medida de Evaluación para Reducción de Reglas

- La función acumulada se puede aproximar a una función beta de la sig. forma:

$$\sum_{i=p}^{t} \binom{t}{i} \left(\frac{P}{T}\right)^{i} \left(1 - \frac{P}{T}\right)^{t-i} = I_{\beta}(p, t-p+1)$$

- Esta simplificación de reglas se puede hacer con un subconjunto del conjunto de ejemplos de entrenamiento
  - REP (***reduced error pruning***)

Medida de Evaluación para Reducción de Reglas

- Variantes: IREP (Incremental REP)
  - Simplifica reglas cada vez que se construyen usando

$$\frac{p + (N - n)}{P + N}$$

  - Maximiza # **pos** cubiertos más el # **neg** no cubiertos
  - Da misma importancia a **neg** no cubiertos y **pos** cubiertos
    - Si una regla cubre 2,000 (p) de 3,000, n=1,000 es evaluada mejor que una regla que cubre 1,000 (p) de 1,001 (n=1)

Medida de Evaluación para Reducción de Reglas

- Otra medida popular

$$\frac{p - n}{p + n}$$

- Sufre de problemas parecidos

RIPPER

- RIPPER es una variante con buenos resultados
  - Construye reglas usando covering
  - Las reduce usando una heurística (como las anteriores) con conjunto de entrenamiento separado
  - Luego “optimiza” al mismo tiempo ese conjunto de reglas

RIPPER

- RIPPER utiliza varias medidas e ideas al mismo tiempo
  - Un conjunto de ejemplos separados para decidir podar reglas
  - Ganancia de información para crecer las reglas
  - Medida IREP para podar reglas
  - Medida basada en MDL como criterio de paro para el conjunto global de reglas

RIPPER

- Cuando construye un conjunto inicial de reglas
  - Toma una regla R_i, del conjunto total de reglas
  - La hace crecer (revision)
  - Hace crecer una nueva regla desde el principio
- Al final
  - Se queda con la regla original o alguna de las otras dos
    - La que hizo crecer o construyó desde cero
    - Tomando en cuenta el error sobre el conjunto total de reglas

Construir Reglas usando Árboles

- Es posible hacerlo directamente
  - Las reglas tienden a ser más complejas de lo necesario
- Se pueden utilizar mecanismos de podado de reglas

Construir Reglas usando Árboles

- Combinar estrategia de ***covering*** con ***splitting***
  - Para construir una regla se construye un árbol podado (splitting)
  - La hoja con mejor covertura se transforma en regla
  - Se eliminan los ejemplos cubiertos por esa regla (covering)
  - Se repite el proceso
  - En lugar de construir un árbol completo, se construyen árboles parciales, se expanden las hojas con mejores medidas de entropía
  - Tiende a producir conjuntos de reglas simples pero con muy buen desempeño

Ripple-down rules

- Construir primero reglas que cubren la mayor cantidad de casos
  - Luego se afinan mediante excepciones
- Primero se toma la clase mayoritaria de entrada
- Todo lo que no se cumpla se toma como una excepción a esta
- Se busca la mejor regla (la que cubra más casos) de otra clase y se añade como una excepción
- Esto divide de nuevo los datos en los que cumplen con esa nueva condición y los que no
- Dentro de los que no cumplen de nuevo se busca la mejor regla de otra clase y se añade como excepción
- Así sucesivamente hasta cubrir todos los casos

Ejemplo de Ripple Down Rules

Default: reprueba
excepto
    Si estudia=si AND memoriza=no
    Entonces pasa
    excepto
        Si copia=si AND descubren=si
        Entonces reprueba
    Else
        Si estudia=no AND copia=si AND descubren=no
        Entonces pasa
        excepto
            Si vecino-sabe=no
            Entonces reprueba

Ripple-down rules

- Ventajas de estas reglas
  - La mayoría de los ejemplos se cubren por las reglas de alto nivel
  - Las de bajo nivel representan realmente las excepciones

Reglas que Consideran Relaciones

- En casos anteriores las reglas consideran la prueba de un atributo contra una constante
  - Son reglas proposicionales
  - Mismo poder que el cálculo proposicional
- Algunas veces se requieren reglas más expresivas
  - Expresar relaciones entre los ejemplos

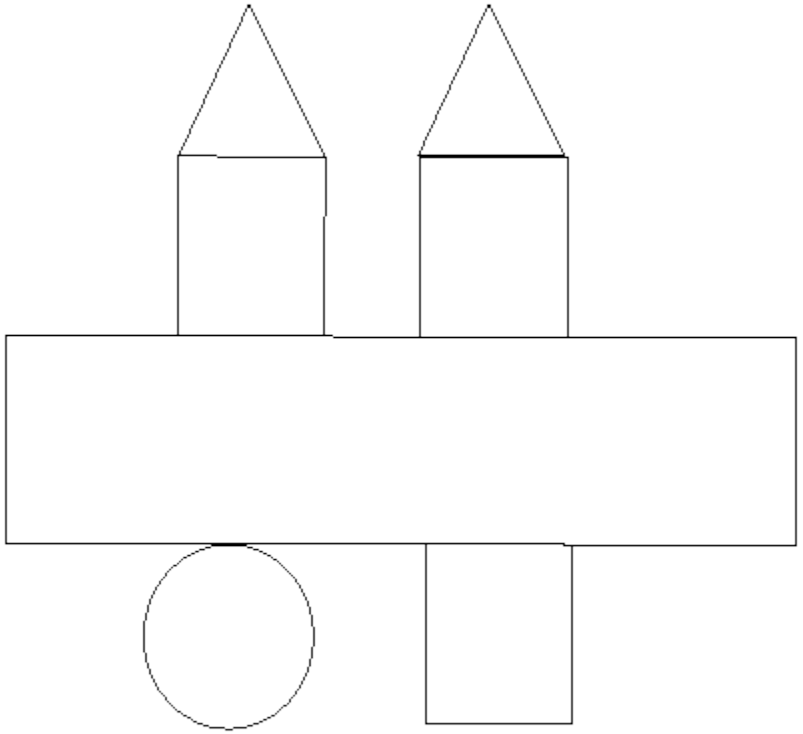
Reglas que Consideran Relaciones

- Dominio de objetos geométricos
  - If width >= 3.5 and weight < 7 then lying
  - If height >= 3.5 then standing
- Si vemos varios ejemplos de este dominio
  - Notamos que los bloques con clase “standing” (de pie) tienen más altura que anchura
  - Es posible generar las reglas
  - If width > heigh then lying
  - If height > width then standing
  - El valor particular de la altura y ancho ya no son importantes, sólo el resultado de su comparación
- Este tipo de reglas se conoce como relacionales
  - Expresan relaciones entre atributos
  - En lugar de referirse a hechos sobre un atributo como las proposicionales

Reglas que Consideran Relaciones

- Otro dominio relacional es el de figuras geométricas
  - Representan una casa como un triángulo sobre un cuadrado
  - La forma de un objeto se representa con un atributo

Reglas que Consideran Relaciones



Reglas que Consideran Relaciones

- Posteriormente veremos como trabajar con dominios relacionales
  - Inductive Logic Programming
  - Graph-based Learning

Ejercicios

- ¿Cuál es el lenguaje de hipótesis en el aprendizaje basado en reglas?
- ¿Cuál es el espacio de hipótesis?
- ¿Cómo acotamos el espacio de hipótesis con el algoritmo PRISM?
- ¿Buscamos hipótesis consistentes?
- Buscar reglas usando el algoritmos PRISM (manualmente) para el dominio del GOLF, lámina 10, para ambas clases.

Resumen: Mapa Mental para Aprendizaje basado en Reglas