uc3m | Universidad Carlos III de Madrid

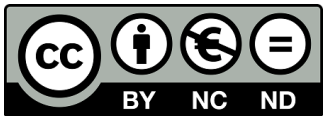
Grado en Ingeniería Informática

2020-2021

Apuntes

Aprendizaje Automático

Jorge Rodríguez Fraile[1]



[1]Universidad: 100405951@alumnos.uc3m.es | Personal: jrf1616@gmail.com

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

1. INFORMACIÓN

1.1. Profesores

Teorías: Raquel Fuentetaja rfuentet@inf.uc3m.es 2.1B19

Prácticas: Rubén Majadas rmajadas@pa.uc3m.es 2.2A12

1.2. Presentación

Vamos a trabajar sobre una modificación del Pac-man.

Los fantasmas no van a por Pac-man, sino que si los toca se los come.

Las prácticas consisten en 4 Tutoriales 0.5 cada uno y 2 Prácticas 1.5 cada una.

El objetivo de la asignatura es conseguir que gane solo el Pac-man sin ayuda nuestra.

El objetivo de las prácticas no es pegar los resultados o con captura, sino documentar bien, incluyendo tablas y gráficas.

Los casos que no funcionan son importantes, no solo poner pruebas con éxito. Así se documentan los cambios realizados.

2. TEMA 1: INTRODUCCIÓN

Definición:

Campo de estudio que da al computador la habilidad de aprender.

Un programa se dice que aprende de una experiencia con respecto a una tarea y alguna medida del rendimiento, si el rendimiento con la tarea al ser medido mejora con la experiencia.

Objetivo: Aprender conocimiento nuevo y mejorar el comportamiento de un sistema.

Se encuentra en las fronteras de la programación.

Programación tradicional: Lo normal, es que el usuario construya el programa que resuelve el problema.

Programación automática: Se crea un programa capaz de generar el programa que es capaz de resolver nuestro problema.

Es útil cuando:

- No existe experiencia humana.
- Los humanos no saben explicar su experiencia.
- Los modelos deben ser personalizados
- Los modelos se basan en enormes cantidades de datos.

Actualmente: Existen muchos algoritmos de aprendizaje automático efectivo y eficientes, recursos computacionales y datos disponibles.

Definición de tarea de Aprendizaje Automático: Mejorar en una **tarea** T, respecto a una **medida de rendimiento** P, basándose en la **experiencia** E (los ejemplos).

Tarea, Medida y Experiencia.

Tipos de tareas de Aprendizaje Automático:

- **Aprendizaje supervisado**: Consiste en etiquetar los datos.
  - **Clasificación**: Determinar de qué clase es un ejemplo.
  - **Predicción – Regresión**: Determinar una etiqueta numérica.

- **Aprendizaje no supervisado**: Agrupa elementos similares, no etiqueta.
    - **Agrupación**.
- **Aprendizaje por refuerzo**: Basado en prueba y error. Dando refuerzo positivo/negativo.

Algunas aplicaciones: Sanidad, Domótica, Banca, Marketing, Personalización, Seguridad, Videojuego, etc.

Problemas no técnicos:

- Las máquinas no son responsables de los diagnósticos, predicciones, o clasificaciones que hacen.
- Los humanos no se fían de los resultados.
- La salida no es entendible por el humano. No explica sus decisiones, se está avanzando para lograr saber cómo piensa.
- Leyes de protección de datos.
- Miedo de los humanos a la pérdida de control.
- Cuestiones éticas.

3. TEMA 2: ÁRBOLES Y REGLAS DE DECISIÓN

3.1. Aprendizaje inductivo (De lo más específico a lo más general)

Encontrar una **función h** (la hipótesis o modelo) que aproxime la **función f** (desconocida) definida por un conjunto de ejemplos.

Los ejemplos normalmente se representan como pares, **(x, f(x))**.

Según como sea la salida de f: Es **clasificación** si es categórica y es de **regresión** si es numérica.

Se basa en inducción, se parte de un ejemplo específico para obtener modelos generales.

Asume que:

- La **hipótesis del aprendizaje es inductiva.**
- Si un modelo o hipótesis **categoriza bien en función a un conjunto de ejemplos grande también lo hará bien para futuros ejemplos.**
- **Siempre hay un sesgo inductivo**, que influye en la decisión, nos lleva más a uno que a otro.
- El lenguaje de representación nos limita si no lo puede expresar bien. Por ejemplo, limitarnos a una función lineal, pero es cuadrática.
- Encontrar una hipótesis adecuada puede ser difícil, f es desconocida y puede ser complicado determinar si h es buena.

El **espacio de hipótesis** es el conjunto de hipótesis que se consideran para aproximar f, **influye mucho para encontrar una buena aproximación**, pueden ser: funciones lineales, polinómicas, lógica de predicados, árboles de decisión, etc.

Los ejemplos tienen atributos/características que los identifican y nos permitirán clasificarlos por clases. Cada ejemplo es una instancia, da valor a los atributos y su clase.

Todos los ejemplos clasificados forman el conjunto de entrenamiento.

3.2. Definiciones

Atributos: Característica que define a un elemento de un conjunto.

Instancia: Colección de valores de atributos.

Clase: Cada uno de los subconjuntos disjuntos.

Ejemplo (positivo): Instancia que pertenece al subconjunto definido por la clase.

Ejemplo negativo: No pertenece al subconjunto definido por la clase.

Generalización de un conjunto de ejemplos de una clase (hipótesis): Descripción que representa al subconjunto de instancias de la clase y no de otras.

3.3. Árboles y reglas de decisión

Usaremos **modelos simbólicos como árboles de decisión**, que usan símbolos para representar lo que los hace más fáciles para ver explicaciones de las decisiones, las podemos entender. Sin embargo, en otros tipos de modelos como los numéricos (redes neuronales), usa números para representar y no es posible que dé explicaciones que entendamos.

3.4. Aprendizaje de árboles de decisión

Nosotros estudiaremos **ID3 (Dicotomizador Iterativo, Quinlan 1986)**, a partir de ejemplos de partida genera árboles de decisión. Normalmente NO son árboles binarios.

Usa búsqueda avara, para encontrar el árbol más sencillo que separa mejor los ejemplos. Utiliza una heurística basada en entropía. Trata de escoger para empezar la clasificación el atributo que mejor separa las distintas clases.

3.5. Algoritmo

1. **Seleccionar el atributo** A_i que maximice la ganancia $G(A_i)$.
2. **Crear un nodo** para ese atributo con **tantos sucesores como valores** tenga.
3. **Dividir los ejemplos** en los sucesores según el valor del atributo.
4. Por cada sucesor,

   sí solo hay **ejemplo de una clase**, entonces se **etiqueta con esa clase**,

   si no, **ejecutar el ID3** con la tabla formada por los ejemplos de ese nodo, pero sin el atributo que todos tiene en común.

3.6. Heurística de ID3 (Heurística de la ganancia de información)

Seleccionar el **atributo que mejor separe los ejemplos de acuerdo con las clases**, que deje subconjuntos más puros (orientados más a un valor).

Para calcular la ganancia se utiliza el concepto de Entropía, como medida de la pureza o impureza de un conjunto ejemplos.

3.7. Entropía

Entropía en clasificación binaria: El conjunto de ejemplos S pertenece a una de las dos clases: $Entropia(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$

- $p_{\oplus}$: **Proporción** de ejemplos **positivos** sobre el total.
- $p_{\ominus}$: **Proporción** de ejemplos **negativos** sobre el total.

$p_{\oplus} + p_{\ominus} = 1$

$p_{\oplus} + p_{\ominus} = 0{,}5$ Entropía máxima

Entropía en múltiples clases: $Entropia(S) \equiv \sum_{c \in C} -p_c \log_2 p_c$

Cuanto más se diferencian las proporciones, cuanto más notablemente hay de una que de otra, más tiende a 0.

Nuestro objetivo es minimizar la entropía.

La ganancia de información: mide la efectividad de un atributo para clasificar, es la **reducción esperada de la entropía** cuando se divide el conjunto de datos original S según el atributo dado A.

- $G(S, A) = Entropia(S) - Entropia\_Atr(S, A)$
- $Entropia\_Atr(S, A) = \sum_{c \in valores(A)} \frac{|S_{A=v}|}{|S|} Entropia(S_{A=v})$
- $Entropia(S_{A=v}) = -\sum_{c \in C} \frac{|S_{A=v, C=c}|}{|_{A=v}|} \log_2 \frac{|S_{A=v, C=c}|}{|_{A=v}|}$

Fig. 3.1: Fórmulas Entropía

Se elige el atributo que maximiza la ganancia de información, ya que menos resta es el que menos entropía tiene, o elegir el que menor **entropía tenga**.

$A = \arg\max_{A \in \mathcal{A}} G(S, A) = \arg\min_{A \in \mathcal{A}} Entropia\_Atr(A, S)$

3.8. Aprendizaje de reglas de decisión

Traducción a reglas:

- Cualquier árbol de decisión se puede convertir a reglas.
- **Reglas**: Estructura del tipo **Si(valor de los atributos)-Entonces(clase a la que pertenece)**

- **Algoritmo**: Por cada rama del árbol, las preguntas y sus valores estarán en la parte de la izquierda de las reglas y la etiqueta del nodo hoja correspondiente será la parte derecha.

Sesgo inductivo en ID3

- Preferir árboles con atributos con más información cerca de la raíz, y árboles cortos.
- **La navaja de Ockham/Occam:** Preferir siempre la hipótesis más sencilla que describa los datos.
- Favorece atributos con muchos valores.

3.9. Ampliación del ID3

En los datos puede haber errores, ruido. Si se ajusta mucho a los datos se produce **sobreajuste**.

No se pueden tratar:

- **Valores continuos de atributos**, números reales, lo que no son discretos.
- **Valores discretos con muchos valores.**

Hay valores de atributos más caros de obtener.

- Hay métodos que los ejemplos vienen incrementalmente, se puede ampliar el modelo, no hace falta volver a empezar: ID4, ID5 y Hoeffding tree.
- Clases continuas: M5
- Representación relacional, la más común es con lógica de predicados: ILP.

Normalmente recibimos los datos como **atributo-valor**, pero también se pueden recibir de forma **relacional**, hay-robot(imagen).

3.10. Ruido

Cualquier cosa que pueda oscurecer la relación entre los atributos y la clase.

- Que los atributos no estén bien descritos o seleccionados.
- Que no sean relevantes los atributos.

Ruido en atributos: Valores erróneos, sin valor (missing values) u outliers (que se salen de sus valores)

Ruido en la clase: Ejemplos con una clase incorrecta, ejemplos exactamente iguales, pero clase distinta (contradictorio)

3.11. Evaluación para Validación de un modelo

No se pueden usar para evaluar ejemplos conocidos, tienen que ser nuevos, para ello se tienen **dos conjuntos de ejemplos, uno para entrenar y otros de prueba**. Tras crear el modelo se pasan los ejemplos de prueba y se calcula el número de errores, se evalúan los resultados.

$Accuracy = \frac{numeroEjemplosClasificadosCorrectamente}{numeroTotalEjemplos}$ Proporción de aciertos sobre el conjunto de prueba.

Un problema de hacer la división de conjuntos es que no se usan todos para entrenar, y esto es problemático si no se tienen muchos.

Validación cruzada k-veces: Método para definir el conjunto de entrenamiento y de pruebas, que permite usar todos los datos para entrenar el modelo. Sirve para **estimar el error del modelo final, no para generar el modelo final**, en el que se usan todos los ejemplos.

Estimación del error del modelo final, en el que se usan todos los ejemplos para entrenarlo.

1. Divide el **conjunto de ejemplos en k partes** iguales.
2. Para cada conjunto, se **entrena con los k-1 conjuntos restantes**, y se pasa por el modelo generado el conjunto seleccionado calculando el error de ese modelo sobre el conjunto. e_i
3. Se estima la **tasa de error total haciendo la media** aritmética de los errores. $r = \sum_{i=1}^{k} \frac{e_i}{k}$

Si k=5, se hacen 5 modelos (clasificadores) para evaluar y otro que es el final que usa todos. k=n entonces n+1 modelos.

El valor de **k típico es 10**, pero la mejor sería que fuese 'Leave one out'(aunque es muy lento y costoso), coger todas menos 1 instancia para entrenar y usar esa para hacer test.

3.12. Medidas adicionales del rendimiento

Tener una precisión alta, no quiere decir que sea bueno, la clase puede estar desbalanceada, que haya más de una clase que otra, aunque se admite cierta diferencia.

Basadas en la Matriz de confusión:

		Clase real	
		Positiva	Negativa
Predicción	Positiva	Positivo Verdadero (TP)	Falso Positivo (FP)
	Negativa	Falso Negativo (FN)	Negativo Verdadero (TN)

Tabla 3.1: Matriz de confusión con fórmulas

$Accuracy = \frac{TP+TN}{TP+TP+FP+FN}$

$Sensibilidad(recall) = \frac{TP}{TP+FN}$ Proporción de las que son positivas acierta. Cuanto más es mejor.

Cuando el coste de FN es alto.

$Especificidad = \frac{TN}{TN+FP}$ En qué medida acierta cuando la clase es negativa.

$Tasa\ de\ falsos\ positivos = 1 - Especificidad = \frac{FP}{TN+FP}$

$Precision\ de\ clase = \frac{TP}{TP+FP}$ En qué medida el clasificador acierta cuando predice clase positiva.

Importante cuando el coste FP es alto.

$F1\ score = 2 \cdot \frac{precision \cdot recall}{precision+recall}$ Medida armónica entre sensibilidad y precisión de clase.

Cuando se busca un buen balance entre recall y precisión de clase, y las clases no balanceadas.

Espacio ROC: Mide como de útil es un clasificador para distinguir entre clases. Cuanto más cerca de la esquina superior izquierda mejor, recoge más área y está más cerca del punto óptimo.

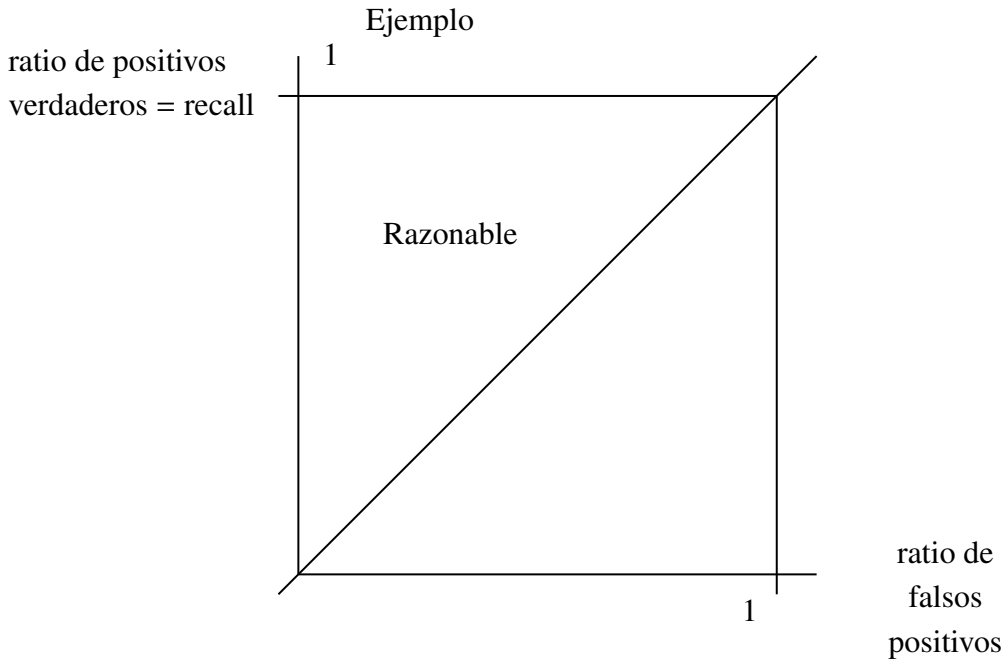


Fig. 3.2: Espacio ROC

Curva ROC: Es una curva en el espacio ROC, en la que cada punto de la recta representa un clasificador, el más cercano al punto óptimo será el mejor, pero teniendo cuidado de los falsos positivos.

En el caso de árboles de decisión no lo utilizaremos porque no tiene sentido, pero si en los casos que hay que definir un umbral (Clasificador Bayesiano) a partir del cual se considera de clase positiva o negativa. Los puntos representan los distintos umbrales.

Cuanto más Área por debajo de la curva mejor.

3.13. Proceso de análisis de datos

Es un proceso iterativo, tras la última etapa se vuelve al principio.

1. **Preprocesamiento**.
   - **Tratamiento de datos imperfectos:**
     - **Reducción del ruido:** Mediante filtros que eliminan instancias clasificadas mal.
     - **Valores desconocidos:** Eliminar instancias/atributos o asignar el valor más probable.
   - **Reducción de datos:** Reducir la variedad.
     - **Normalización**: Valores entre 0 y 1.
     - **Discretización**: En rangos.

- **Selección de instancias**: Aleatoriamente, los más parecidos entre sí, los más diferentes entre sí o según alguna distribución.
- **Selección de atributos:**
  - Reducción de la dimensionalidad de los datos, mediante filtrado según algún criterio o seleccionando un subconjunto.
  - **Búsqueda**: Cualquier técnica.
  - **Evaluación**: Correlación, entropía, etc.
  - **Criterio de parada:** Porcentaje, umbral, iteraciones, etc.
  - **Técnicas de wrapper**: Se genera el modelo con todos los atributos, se evalúa el modelo y se ve el subconjunto de atributos mejores.
  - **Búsqueda en el espacio de estado de los conjuntos de atributos.**
    - Se puede comenzar por el conjunto completo (por arriba) o por el conjunto vacío (por abajo).
    - La búsqueda puede ser de cualquier tipo.
    - La evaluación de cada nodo, que son subconjuntos de atributos, se realiza llamando al algoritmo inductivo seleccionado, una función de evaluación. Tras evaluar se opera.
  - **PCA (No lo usaremos):** Análisis de componentes principales. Es una solución algebraica, describe los datos en términos de nuevos atributos que no están correlados entre sí.
- **Datos no balanceados:** Crear ejemplos sintéticos de la clase desbalanceada.

2. **Diseño**: Selección de algoritmo y Selección de parámetros.
3. **Ejecución de algoritmo.**
4. **Postproceso.**
   - Análisis de resultados.
   - Visualización.

3.14. Aspectos avanzados

3.14.1. Sobreajuste (overfitting)

Fenómeno que se produce al hacer aprendizaje, porque **el modelo se ajusta demasiado a los ejemplos, y eso hace que no generalice bien.**

Mejora con los que se entrena, pero con aquellas que no conoce falla más.

Para evitar sobreajuste se debe generalizar, hay **2 métodos**:

- **Pre-poda**: Mientras se construye se poda.
  - **Método de χ^2 (chi-cuadrado)**: Si en un nodo la diferencia entre clases no es significativa, no se divide, para con la mayoría. Muy conservador.
  - **Mediante curvas de error**: Cross-validation. De esta manera podemos detectar el punto de inflexión, donde se empieza a producir overfitting (en el punto que se separan las curvas).
- **Post-poda**: Después de generar poda.
  - **Eliminando subárboles**: Eliminando nodos del árbol empezando por las hojas, se evalúa el error tras eliminarlo, si no mejora probamos otro.
  - **Eliminando precondiciones o reglas**: Tras haberlo pasado a reglas, eliminamos condiciones o reglas completas. Tras la eliminación se calcula el error, si mejora, el modelo principal pasa a ser el que lo eliminó y se sigue probado. Si no se prueba otra condición o regla.

3.14.2. Atributos con valores continuos

1. Se **ordenan los valores del atributo**, y se especifica la clase a la que pertenecen.
2. Se **observan los puntos en los que pasa de una clase a otra** y se hace media con los puntos de corte.
3. Los **nodos de decisión se crean según:**
   - Se trata como **atributo binario**: Se escoge **una sola regla de distinción** entre valores, el de mayor ganancia de información, que se eligen con los valores medios. Las reglas son del tipo atrib<punto1 o atrib<punto2.
   - Se trata como **atributos multivaluado**: Se crean **tanto grupos como queramos**, lo más preciso es uno por cada intervalo de valores que tengan la misma clase.

3.14.3. Atributos con muchos valores

ID3 prefiere atributos con mayor número de valores.

Problema: Que cada uno puede tener un valor único, y no se podrá usar para clasificar.

Alternativas:

- Por cada valor v del atributo A, se puede **crear un atributo binario**, de sí ese atributo toma ese valor o no.

- **Razón de ganancia (GainRatio, GR)**: Ganancia calculada como la ganancia partida por una penalización por el número de valores del atributos.

$$GR(S,A) = \frac{G(S,A) = maxEntropia(S) - EntropiaAtrib(S,A)}{Split\_Information(S,A) = -\sum_{v \in valores(A)} \frac{|S_v|}{|S|} \cdot log_2(\frac{|S_v|}{|S|})}$$

Fig. 3.3: Razón de ganancia

$$Split\_Information(S,A) = -\sum_{v \in valores(A)} \frac{|S_v|}{|S|} \cdot log_2(\frac{|S_v|}{|S|})$$

3.14.4. Atributos con costes variables

Como podrían ser pruebas médicas caras, obtener los ejemplos es caro. **Consiste en penalizar la entropía de los atributos con coste variable.**

$\frac{G(S,A)}{unidaddecoste(A)} O \frac{G(S,A)}{unidaddecoste(A)^2}$

3.15. Otras alternativas a ID3

Clasificadores débiles:

- **ZeroR**: En clasificación, dará siempre la **clase más frecuente**, la moda de la clase. En regresión, devuelve la media.
- **OneR**: Construye un conjunto de **reglas de decisión con un solo atributo**. Se elige aquel atributo con menor error, para cada valor del atributo se crea una regla, donde el valor que toma es el más frecuente.

3.16. Implementaciones

ID3, 1986

C4.5, 1993 Sucesor de ID3, atributos numéricos, conversación a reglas, poda.

C5.0 Versión comercial.

J48 Implementación JAVA de C4.5.

4. TEMA 3: REGRESIÓN

Edad	Salario	Indefinido	Cliente	Crédito
25	10.3	No	No	20.3
54	0.0	Sí	Sí	40.6
43	53.0	No	No	70.2
35	15.8	No	No	12.5
26	97.9	Sí	Sí	89.3
45	37.8	Sí	Sí	32.1
56	13.4	No	Sí	56.7
26	12.9	Sí	Sí	7.9

La clase en regresión es numérica.

4.1. Regresión como clasificación

Discretizando la clase, pero se pierde información y hay que elegir un método de discretización.

Soluciones:

- **Discretización de clase.**
- **Discretización fija.**
- **Discretización dinámica.**

4.2. Función de regresión

4.2.1. Regresión lineal

Hallar una **función que se ajuste lo mejor posible a la nube de puntos**. En el caso de la **lineal es una recta.**

Aproximar una función $f(x)$ que no tiene por qué ser lineal en regresión, con una función $\hat{f}(x) = w_0 + w_1a_1(x) + w_2a_2(x) + ... + w_na_n(x)$ donde

- a_i denota el **atributo** i-ésimo del ejemplo x.
- w_i **peso** del atributo

Objetivo: Encontrar aquellos w_i que minimicen el error entre la función clase y el valor de la aproximación.

Equivalente a **minimizar el error cuadrático sobre el conjunto de entrenamiento total,** C: $E = \sum_{c \in C}(f(x) - \hat{f}(x))^2$ Lo que queremos es minimizar.

4.2.2. Error de Regresión

La **suma de todas las diferencias de los valores de la función y su aproximación**. Buscamos aquella aproximación que minimice E.

Minimizando el error

El problema de definir la función se traslada a un problema de definir el vector de pesos w. Se debe encontrar el vector $\vec{w}$ que minimice la función de error, problema de búsqueda en el espacio de pesos.

Aproximación: Descenso de gradiente.

4.2.3. Descenso de gradiente

Sobre la función de errores por variable **vamos calculando la tangente y desplazando los valores de los pesos**.

- Consisten en movernos poco a poco en los pesos, para no pasarnos, y se parte de pesos aleatorios.
- Si la **tangente** tiene pendiente **positiva**, nos desplazamos a la **izquierda**.
- Si la **tangente** tiene pendiente **negativa**, vamos a la **derecha**.

Gradiente del error respecto a w: Derivada parcial de la función de error de cada uno de los pesos. $\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_n}\right]$

Regla de entrenamiento: $\Delta\vec{w} = -\eta\nabla E[\vec{w}]$

Derivada del error: $\frac{\partial E}{\partial w_i} = \sum_e (t_e - o_e)(-x_{i,e})$

- t_e: Valor **verdadero** para la instancia e.
- o_e: Valor de **salida del modelo** para la instancia e.
- $x_{i,e}$: Valor del **atributo** a para la instancia e.

Procedimiento: Descenso de Gradiente(C, n)

- C conjunto de ejemplo de entrenamiento. n tasa de aprendizaje cuando salta.
  - Inicializar cada w_i a algún valor aleatorio pequeño
  - Hasta que la condición de fin sea alcanzada, hacer
    - Inicializar cada Δw_i a cero
    - Para cada $\langle \vec{x}, \vec{t} \rangle \in C$:
      - $o = \hat{f}(\vec{x})$
      - Para cada peso w_i:
        $\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$
    - Para cada peso, w_i:
      $w_i \leftarrow w_i + \Delta w_i$

4.2.4. Regresión no lineal

Cuando usamos este tipo de funciones hay que tener **cuidado**, la evaluación se debe realizar con ejemplos de un conjunto distinto al de entrenamiento, porque la función no lineal se puede ajustar muy bien a esos, pero no al resto.

Esta función es más difícil de generar.

4.3. Regresión paramétrica

Elegir una función lineal, logística o cualquier tipo, para que se ajuste.

El modelo o predictor adopta una forma predeterminada.

Regresión lineal es regresión paramétrica. También se puede hacer regresión no lineal paramétrica, función logística, polinómica, etc.

4.4. Árboles de regresión

Regresión no lineal y no paramétrica.

4.5. M5

M5 es una variación de CART, donde las hojas son valores numéricos y elige aquel atributo que maximice la reducción esperada en varianza.

Estrategia: Se divide el espacio en partes y hace regresión lineal por partes, no global.

Algoritmo: Parecido a ID3, pero en este caso se trata de reducir la variación interna de los valores de la clase de cada subconjunto.

- Elige aquel atributo que maximice la reducción del error (en vez de entropía o desviación típica). Nos quedamos con la menor media de desviaciones.

$$\Delta\,\text{error}(S, A) = \text{sd}(S) - \sum_{v \in \text{ valoresTestNodo } (A)} \frac{|S_{A=v}|}{|S|} \times \text{sd}\,(S_{A=v})$$

Fig. 4.1: Formula desviación M5

  - **S** conjunto de ejemplos en el nodo a dividir.
  - $S_{A=v}$ Ejemplos con valor v en el atributo A.
  - **sd(S)** Desviación típica de los valores de la clase para los ejemplos en S.

Criterio de parada: Pocos ejemplos o poca variación de los valores (desviación típica pequeña)

Hojas: Se calcula un modelo lineal y se utiliza regresión estándar.

Salida: En las hojas tiene funciones de regresión, en los nodos no hoja se tienen atributos acotados <. Cada nodo tiene un conjunto de entrenamiento y se hace su regresión lineal.

En los árboles de regresión, los nodos hoja tienen modelos lineales y para llegar a esos nodos se usa como heurística la Desviación típica (cuanto menor mejor), que se calcula haciendo una media ponderada ($S_{S=si} \cdot \frac{|S_{S=si}|}{|S|}$) de las desviaciones típicas de cada uno de los conjuntos resultantes.

Tras la construcción del árbol hay un proceso de poda, que consiste en Simplificar los modelos lineales de las hojas, Simplificar el árbol para que sea más pequeño y por último el Suavizado.

4.6. Poda

Paso 0: Antes de realizar la poda se crean modelos lineales de los nodos intermedios, que se utilizaran para ver si se pueden simplificar los subárboles por ese modelo del nodo intermedio. Para estos modelos se usan solo los atributos que aparecen en el subárbol.

Para decidir si simplificamos por un modelo lineal necesitamos una medida del error. Si el modelo lineal proporciona menor error con respecto al subárbol podemos simplificarlo utilizando el modelo.

La medida de error que empleamos es el **Error absoluto medio**, que es la media del error al clasificar las instancias el subconjunto de instancias de entrenamiento del subárbol. Se calcula como $residuo(T) = \frac{1}{n} \sum_{i \in T} \|f(i) - \hat{f}(i)\|$, dado un subárbol con un subconjunto de n instancias de entrenamiento T.

El residuo subestima el error en instancias nuevas (que no ha visto nunca), para solucionar esto se multiplica por $\alpha = \frac{n+v}{n-v}$. Tal que n es el número de ejemplos del subárbol y v el número de atributos del modelo.

Por lo que el error aumenta cuando hay muchos parámetros o hay pocas instancias.

Fórmula del error: $error\_estimado(T) = \alpha \times residuo(T)$ Es una proporción del residuo.

1. **Simplificación de los modelos lineales** (Busca eliminar atributos)

   Se realiza en cada modelo lineal. Se eliminan atributos, que se seleccionan utilizando escalada para reducir el error estimado. En el extremo, deja solo una constante, como termino independiente.

   $M = 0{,}25a_1 + 0{,}12a_2 + 300a_5 - 40 \Rightarrow M = 0{,}12a_2 + 300a_5 - 40$

2. **Simplificación del subárbol** (Busca eliminar subárboles por modelos lineales)

   Cada nodo interno del árbol tiene un modelo lineal simplificado y un modelo subárbol, y se elige aquel que minimice el error estimado. Si el modelo da mejor resultado, se sustituye el subárbol por el nodo del modelo.

3. **Suavizar el árbol**

   Algunos trabajos han comprobado que realizar un suavizado mejora la predicción final, dado que la predicción en cada nodo puede variar mucho. Lo que hace es en lugar de devolver la predicción del modelo en el nodo hoja, se consideran también los modelos en los nodos intermedios entre el nodo hoja y el nodo raíz.

5. TEMA 4: OTRAS TÉCNICAS

5.1. Aprendizaje Bayesiano

Funcionan bien en la clasificación de textos, text mining. Permite recibir los datos de manera incremental y va creciendo el modelo.

5.1.1. Hipótesis más probable (MAP)

Se busca la hipótesis que explique mejor los datos.

$$P(h_i/E) = \frac{P(E/h_i)P(h_i)}{P(E)}$$

Fig. 5.1: Formula Hipótesis más probable

Es aquella que arg máx $P(h_i/E)$ o arg máx $P(E/h_i)$. Cogemos el h_i más probable, pero puede haber otros muchos modelos que dicen que no y que habría que considerar, entonces lo que queremos es la clase más probable para esos datos.

5.1.2. Clasificación más probable

Probabilidad de la clase C dados los ejemplos E:

$$P(C/E) = \sum_{h_i \in H} P(C/h_i) \cdot P(h_i/E)$$

Fig. 5.2: Formula Clase más probable

A esto se le llama **Clasificador Bayesiano Óptimo**, en la práctica el tamaño de H hace que sea imposible, además no tenemos un modelo como tal sino la probabilidad de la clase más probable para esos datos.

5.1.3. Naïve Bayes

$$\text{arg máx } P(C/a_1, ..., a_n) = \text{arg máx}_{c \in C} \frac{P(a_1,...,a_n/C)P(C)}{P(a_1,...,a_n)} = \text{arg máx}_{c \in C} P(a_1, ..., a_n/C)P(C)$$

Fig. 5.3: Fórmula Naïve Bayes

Podemos quitar el denominador porque todas las clases se evalúan con los mismos atributos. Y $P(a_1, ..., a_n/C)$ es muy complejo de calcular, exponencial, se necesitaría una tabla de todas las combinaciones.

Para evitar esta complejidad se hace una simplificación, **se asume que los valores de los atributos una vez se conoce la clase son condicionalmente independientes**, lo que quiere decir que se puede calcular como un producto. $P(a_1, ..., a_n/C) = P(a_1/C) \cdot P(a_2/C) \cdot ... \cdot P(a_n/C) \cdot P(C)$. El problema es que ahora es una aproximación y no un clasificador bayesiano óptimo. **Suele funcionar bastante bien** esta aproximación.

$P(c = c_1) = \frac{n\_casos\_c=1}{n\_casosTotales}$ $P(a = v_1/c = c_1) = \frac{n\_casos\_c=1\_y\_v=v_1}{n\_casos\_c=1}$

5.2. Redes de neuronas

5.2.1. Neurona

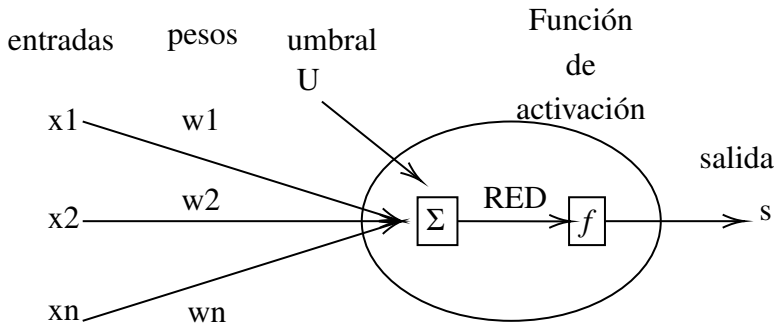


Fig. 5.4: Diagrama de una neurona

Las **entradas son los valores de los atributos** $(x_1, x_2, .., x_n)$ y un umbral, se hace **combinación lineal de las entradas con unos pesos** $(w_0 + w_1x_1 + ... + w_nx_n)$ y se pasa el **resultado por una función de activación** que nos da la salida.

Tipos de función de activación:

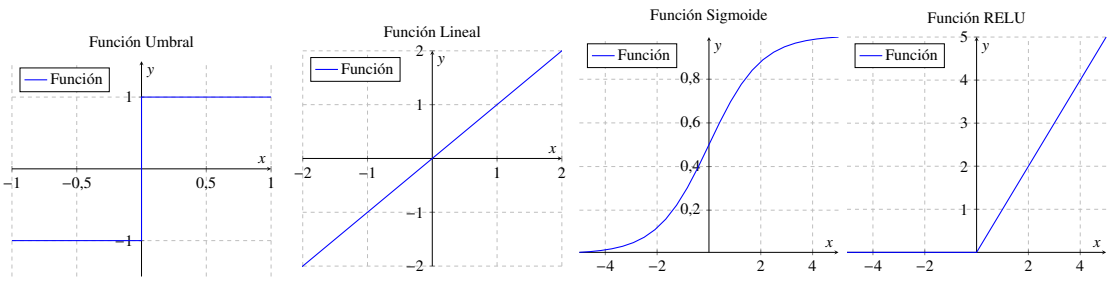


Fig. 5.5: Función Umbral

El aprendizaje consiste en determinar los pesos que se aplican a las entradas.

La regla que se aplica de forma habitual es la **Regla delta**, que es parecido al descenso del gradiente. Se empieza con peso aleatorio y con las iteraciones del entrenamiento se van actualizando los pesos. $w_i \leftarrow w_i + \Delta w_i$ El delta es una pequeña variación es $\eta(t-o)x_1$ donde η es una tasa de aprendizaje, t la salida real y o el valor que damos.

En general para hacer descenso del gradiente necesitamos que la función sea derivable y con un único mínimo.

Se van metiendo los datos de entrada, se recibe la salida y según el error se van actualizando los pesos, para cuando los pesos se estabilizan.

5.2.2. Perceptrón

Es una forma más simple de red de neuronas, formada por una sola neurona. Usa la función escalón como función de activación.

Se usa en tareas de clasificación lineal, es capaz de determinar el hiperplano capaz de discriminar los ejemplos en dos clases.

5.2.3. Redes

Red neuronal multicapa

Es un **conjunto de neuronas conectadas entre sí** que se distribuyen en capas.

Ventajas:

- Son robustas ante el ruido.
- Trabajan con datos complejos (difíciles de clasificar, como sensores)
- Éxito en reconocimiento del habla y visión. Deep learning
- Dan buenos resultados.

Desventaja: El aprendizaje es lento.

En las redes multicapa el cálculo del error se complica, hay varias salidas y capas ocultas.

5.2.4. Backpropagation o Retropropagación

Se basa en: $Error \sim \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$. Para cada ejemplo:

1. Se propaga desde la **entrada hasta la salida** (calcular la salida de la red)

2. **Propagación del error hacia atrás**: En las que están en la última capa es fácil, pero para las que están en las capas intermedias no tanto.

    1. Se calcula el **error en la unidad de salida**: $\delta_k \leftarrow o_k(1-o_k)(t_k-o_k)$ Este valor es el (t-o) de la variación delta, en este caso para una función sigmoide en vez de lineal
    2. **En las capas ocultas**: Se calcula con las unidades que están conectadas posteriormente, $\delta_n \leftarrow o_n(1-o_n)\sum_{k \in output} w_{kn}\delta_k$

5.2.5. Deep Learning

Redes neuronales convolucionales. Se usan para procesado de imágenes.

Puede recibir datos en crudo, no necesita que se los pasemos como atributo valor, lo que es una ventaja no hay que elegir los atributos relevantes y organizar los datos. Recibe el grueso de los datos y los va cogiendo por partes y agrupando, Convolution. Después se reduce la resolución de la imagen, Pooling. Se repite el proceso hasta que tenemos los datos para clasificar, extracción de características, y podemos generar una red neuronal totalmente conectada, classification.

5.3. Algoritmos genéticos

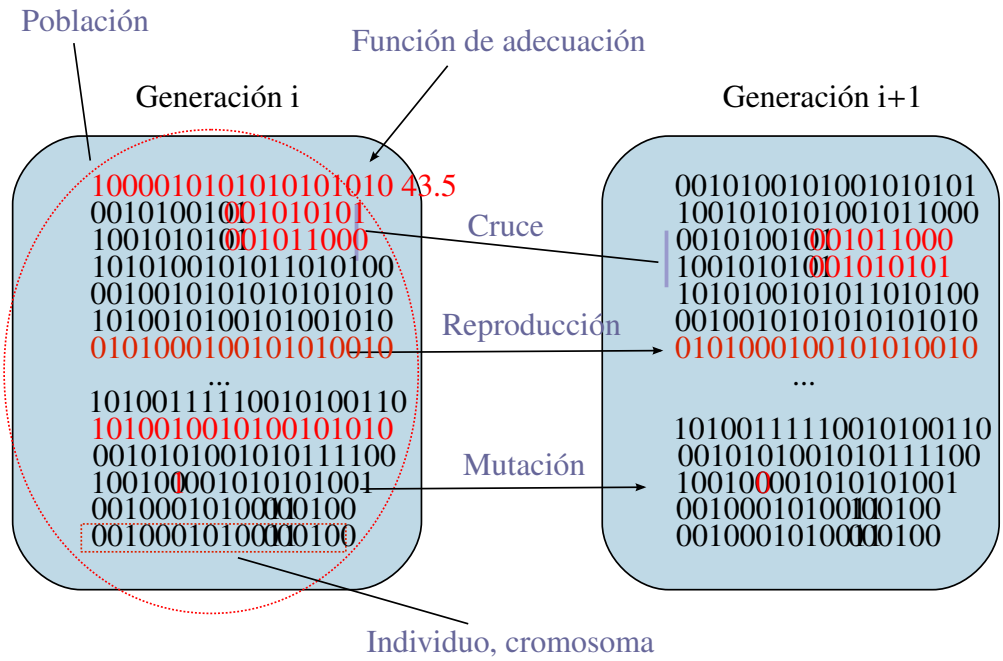


Fig. 5.6: Diagrama Genético

Aprendizaje basado en evolución simulada, se mantiene una colección de soluciones, se evoluciona a través de generaciones por recombinación de las más adecuadas y elimina-

ción de las menos adecuadas, y se termina cuando se obtiene una solución suficientemente adecuada.

Son algoritmos de búsqueda local estocástica, que las acciones tienen una probabilidad de ocurrir, son buenos para la optimización.

Hacen **búsqueda en el espacio de soluciones**, se usan cuando es fácil encontrar una solución, pero no la óptima. Funcionan por generaciones, en la que cada **generación** tiene una **población**, que está compuesta por p **individuos**, cada uno es una solución al problema.

Factores clave:

- Elegir como representar los datos, se pueden utilizar cadenas de bits (llamado Genético) o alguna estructura de datos más compleja.
- Las operaciones para combinar soluciones y generar soluciones nuevas.
- La **función de adecuación (fitness)** que indica como de buena es una población, cuanto más es mejor. Transforma el genotipo (cadena de bits) en el fenotipo (significado de la cadena bits).

Proceso:

1. Se selecciona un conjunto de individuos de la población de soluciones. Se pueden elegir por Ruleta (por número aleatorio) o por Torneo (se coge un par de individuos y se escoge el mejor, dejando al otro fuera).
2. A la nueva población se le realizan una serie de modificaciones como: reproducción, cruce, mutaciones y clone. Y quedan unos sobrevivientes, siempre tiene el mismo número de individuos. Cada operación tiene una probabilidad, por eso es búsqueda estocástica.

   Cruce Una parte de un par de individuos, mezcla de los progenitores.

   Reproducción Pasa a la nueva generación/población igual.

   Mutación Cambios muy pequeños, un solo bit.
3. Se evalúa la nueva población. Si es mejor pasa a ser la población base (generación), si no se descarta y se intentan otras alternativas.

6. TEMA 5: TÉCNICAS VAGAS Y NO SUPERVISADAS

6.1. Aprendizaje basado en instancias (IBL)

No genera modelo, lo que hace es que cuando recibe un dato nuevo busca algunos similares de los conocidos.

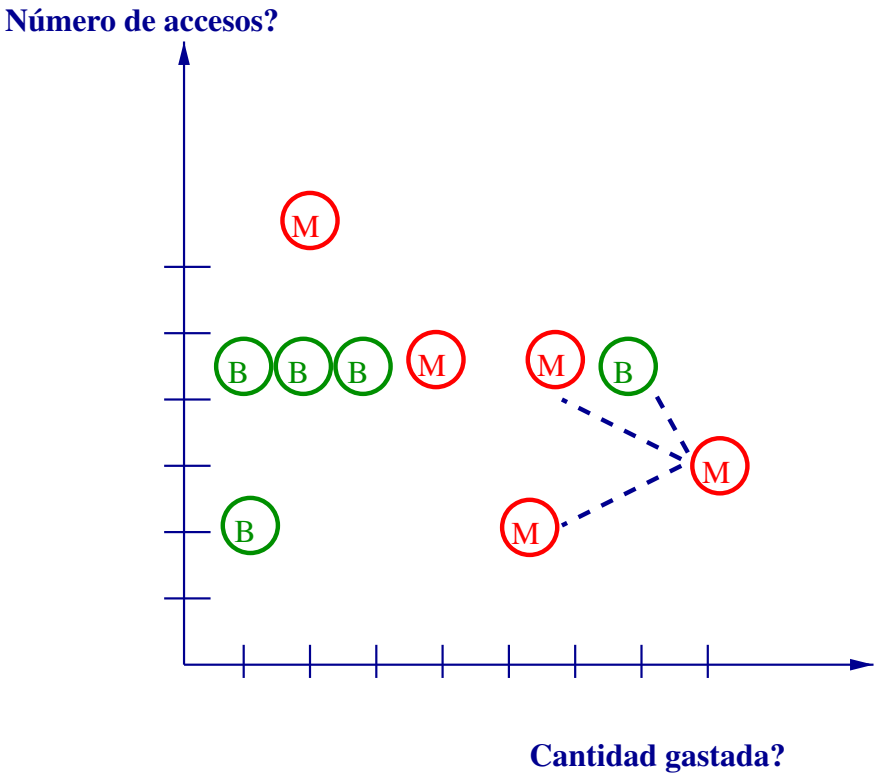


Fig. 6.1: Estrategia IBL

Fases:

1. **Entrenamiento:** Almacenamiento de todo el conjunto de datos disponibles.
2. **Generalización:** Cuando recibe un nuevo dato, se cogen un conjunto de datos similares, que son utilizados para clasificar el nuevo dato. Todo el cómputo se realiza en tiempo de clasificación, Aprendizaje Perezoso.

Para definir similitud, se utilizan medidas de distancia (más cercano, más similares).

El algoritmo recibe como entrada k, que indica el número de vecinos más cercanos que se van a considerar (K-NN, K-Nearest Neighbors).

6.1.1. K-Nearest Neighbors

Las instancias corresponden a puntos en un espacio de dimensión n (n atributos) y los vecinos más cercanos se calculan con la distancia euclídea ($d(\vec{x}, \vec{z}) = \sqrt{\sum_{i=1}^{n} (x_i - z_i)^2}$).

Si los atributos tienen diferentes rangos, se normalizan, y si son valores simbólicos, se define el valor por posible valor.

6.1.2. Función Objetivo

Dada una instancia x y sus k vecinos, nos indica cuál es la clase. Dependiendo del tipo de valores se usan distintos métodos:

Caso discreto (clasificación) Valor más común de entre los k vecinos más cercanos.

$$\hat{f}(\vec{x}) = \arg\max_{c \in C} \sum_{i=1}^{k} \delta(c, f(\vec{z^i})) \tag{6.1}$$

$$\hat{f}(\vec{x}) = \arg\max_{c \in C} \sum_{i=1}^{k} w_i \times \delta(c, f(\vec{z^i})) \tag{6.2}$$

Fig. 6.2: Función objetivo K-Nearest: Clasificación

Caso continuo (regresión) Valor medio de entre los k vecinos.

$$\hat{f}(\vec{x}) = \frac{\sum_{i=1}^{k} f(\vec{z^i})}{k} \tag{6.3}$$

$$\hat{f}(\vec{x}) = \frac{\sum_{i=1}^{k} w_i f(\vec{z^i})}{\sum_{i=1}^{k} w_i} \tag{6.4}$$

Fig. 6.3: Función objetivo K-Nearest: Regresión

También se puede ponderar la importancia de cada vecino en el valor de la función objetivo, en función de su distancia. Donde $w_i = \frac{1}{d(\vec{x}, \vec{z^i})^2}$.

6.1.3. Selección de los K vecinos

El parámetro k identifica cuantos vecinos se utilizan para la decisión, su valor es el que mejor resultados aporta, no hay una regla que siempre funcione. Las regiones que se forman con 1-NN se denominan regiones de Voronoi.

6.1.4. Medidas de distancia

Distancia de Hamming (atributos no numéricos) $(x_m - z_m) = \begin{cases} 0 & \text{si } x_m = z_m \\ 1 & \text{si } x_m \neq z_m \end{cases}$

Manhattan $d(\vec{x}, \vec{z}) = \sum_{m=1}^{n} |x_m - z_m|$

Minkowski $d(\vec{x}, \vec{z}) = \left(\sum_{m=1}^{n} |x_m - z_m|^k\right)^{\frac{1}{k}}$

6.1.5. Propiedades

- Las instancias son puntos de un espacio n dimensional, donde n es el número de atributos.
- No hay muchos atributos por instancias, 20 aprox.
- Hay muchos datos de entrenamiento.

Ventajas

- Simple e intuitivo.
- Fácil de usar.
- Funciona bien cuando la función objetivo es muy compleja.
- El entrenamiento es muy rápido.
- Efectivo si hay muchos ejemplos. Consistente asintóticamente: Con infinitos ejemplos y un k suficientemente grande, es capaz de aproximarse al mejor clasificador (Bayes óptimo).

Problemas y sus soluciones

- Muy **sensible a atributos irrelevantes**
  - **Selección de atributos:** Determinar qué características son las interesantes, y eliminar las restantes. Se pueden utilizar técnicas wrapper, que busca en el espacio de atributos, métodos estadísticos, algoritmos genéticos, etc.
  - **Ponderación de atributos:** Es menos radical que eliminar atributos y consiste en dar el peso que minimice el error a los distintos atributos a la hora de seleccionar vecinos por distancia.
- **Coste de clasificación alto (tiempo)**, no tarda en entrenar, pero cuando clasifica compara todos los ejemplos

- **Indexación:** Organizar el conjunto de instancias
  - **Árboles KD (k dimensional):** Particionar el espacio en regiones marcadas por las instancias que tenemos, en el que cada rama va reduciendo el espacio. Alterna entre dimensiones.

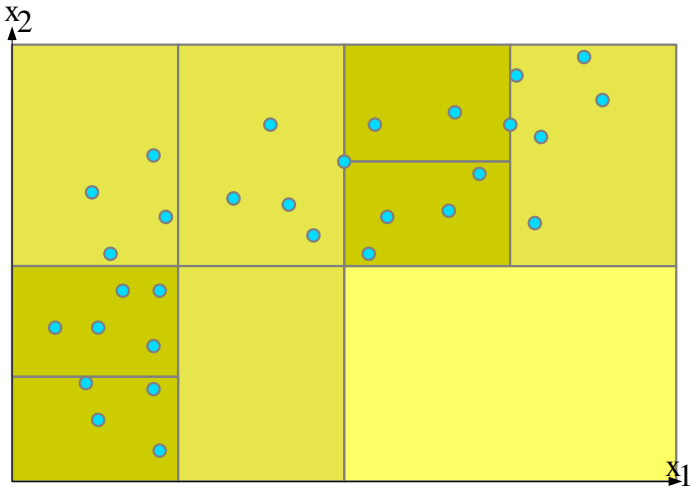


Fig. 6.4: Ejemplo Árbol KD

  - **Agrupación o clustering:** Que es de aprendizaje no supervisado.

- **Almacenamiento (memoria) de todos los ejemplos**
  - **Selección de instancias:** Elegir un grupo reducido de instancias (prototipos, ejemplos representativos) que mantengan la misma información que el conjunto total. Se puede hacer de manera incremental, desde el conjunto vacío e ir añadiendo, o decremental, desde el conjunto completo y eliminando. Se sigue un criterio para sacar o meter una instancia en el conjunto, que siga clasificando correctamente o se reduce las que clasifica mal.
  - **Reemplazo de instancias:** Calcular prototipos a partir del conjunto de entrenamiento, y esos prototipos podrían sustituirse por los ejemplos que lo han generado. Se pueden generar con técnicas de clustering.

6.1.6. Otros métodos

Locally Weighted Regression

En vez de devolver la media de la clase de los k vecinos, se crea un modelo lineal de forma local en zonas del espacio. Por cada consulta se obtienen los k vecinos y se crea el modelo de regresión lineal.

Se podría aplicar cualquier otro método de aproximación, como las redes neuronales.

Case Based Reasoning

Los ejemplos no son de la forma atributo-valor, sino que se representan con cualquier estructura de datos, como pueden ser imágenes, documentos, planos, etc. Consigue adaptar los datos de otros problemas al actual, viendo cómo se han resuelto problemas previos.

6.2. Aprendizaje no supervisado

No conocemos la clase, solo tenemos atributos en los ejemplos.

Lo que se hace es ver que instancias son parecidas y se crean grupos de instancias similares. También se utiliza para reducir el número de instancias y quedarnos con el más representativo del grupo.

6.2.1. Posibles objetivos

- **Agrupación:** Dado un conjunto de ejemplos sin etiquetar, agruparlos según algún criterio.
- **Indexación:** Dado un conjunto de ejemplos, facilitar su acceso organizándolos.
- **Asociación:** Dados unos datos, establecer relaciones y/o asociaciones entre datos.
- **Generación de jerarquías:** Dados unos datos en un mismo nivel, generar jerarquías en dichos datos.
- **Reducción de dimensionalidad:** Dados unos datos, reducir la dimensión o número de atributos que caracterizan los datos.
- **Visualización:** Dados unos datos con representación compleja, permitir su visualización.

6.2.2. Agrupación

El objetivo es dado un conjunto de ejemplos sin etiquetar, agruparlos siguiendo algún criterio predefinido. Usaremos la técnica k-medias. Ese criterio suele venir por:

- **Aprendizaje paramétrico:** Parámetros asumidos, se asume que los datos siguen una distribución.
- **Aprendizaje no paramétrico:** Necesitamos alguna medida de distancia para ver cuáles se parecen más entre ellos. Se basa en agrupar las instancias por similitud; a menor distancia, mayor similitud.

Se puede utilizar Distancia euclídea (la más usada) $d(\vec{x}, \vec{z}) = \sqrt{\sum_{i=1}^{n} (x_i - z_i)^2}$, euclídea ponderada $d(\vec{x}, \vec{z}, \vec{w}) = \sqrt{\sum_{i=1}^{n} w_i (x_i - z_i)^2}$, etc.

Tipos de técnicas:

- Técnicas de particionamiento (como k-medias).
  La salida es un conjunto de clases/grupos de instancias.
- Técnicas Aglomerativos (jerárquica).
  La salida es una jerarquía de clases, en cada clase hay instancias, pero hay clases de clases con mayor importancia.

k-medias o Algoritmo de Lloyd

Realiza una discretización o particionamiento del espacio.

Genera regiones de Voronoi y nos proporciona un representante de la región, se definen mediante una medida de distancia y un conjunto de ejemplos representativos.

Objetivo: Generar el conjunto de prototipos/centroides que minimice una determinada medida de error o distorsión.

Algoritmo de Lloyd Generalizado (T, k): Conjunto de entrenamiento T y k número de grupos/clases. Los prototipos se representan con μ.

1. Comenzar con un conjunto inicial de k prototipos $C_1 = \{\vec{\mu_1}, ..., \vec{\mu_k}\}$.
2. Dado un conjunto de prototipos, C_m, ejecutar la Iteración de Lloyd para generar el siguiente conjunto de prototipos, C_{m+1}. Los prototipos varían entre conjuntos, se van centrando.

   Iteración de Lloyd (C_m, T, k): C_m Conjunto prototipo, T Conjunto de entrenamiento y k número de grupos. Crea regiones y recalcula centroide.

   a) Dado un conjunto de prototipos $C_m = \{\vec{\mu_1}, ..., \vec{\mu_k}\}$, partir el conjunto de entrada T en k particiones $R_i, ..., R_k$ usando la siguiente condición: Los ejemplos pertenecen a la región del prototipo más cercano, y solo de ese. (Si se genera una partición vacía, utiliza alguna estrategia para tratar el caso)

   $$R_i = \{\vec{x} \in T : d(\vec{x}, \vec{\mu_i}) \leq d(\vec{x}, \vec{\mu_j}); \forall j \neq i\}$$

   b) Recalcular los centroides de cada partición, pasa a ser el punto medio de la región, y actualiza el conjunto de prototipos con esos centroides.

   $$centroide(R_i) = \frac{1}{|R_i|} \sum_{\vec{x} \in R_i} \vec{x}$$

3. Calcular la distorsión para el nuevo conjunto, C_{m+1}, midiendo como de cerca están las instancias x_i del prototipo μ_j.

$$D = \sum_{i=1}^{|T|} d(\vec{x_i}, \vec{\mu_j})^2$$

$$C_{m+1} = \{\vec{\mu_i} | \vec{\mu_i} = centroide(R_i)\}$$

4. Si el decremento de la distorsión entre dos interacciones no supera un umbral (bastante bajo), parar. Si no ir al paso 2.

Algunas consideraciones:

- Atributos con diferentes rangos es imprescindible normalizarlos.
- Atributos simbólicos: Utilizamos Distancia de Hamming.

Parámetros de k-medias:

Nos indica el número de clases a considerar y nos sirve para la selección de semillas.

Selección de semillas:

- ejecutar varias veces y quedarse con la mejor partición.
- otra alternativa es utilizar K-MEANS++ (que es una selección más inteligente)

Selección de k automáticamente:

- Se puede probar con distintos k y quedarse con el que los conjuntos tengan menos distorsión.
- Otro método es G-MEANS.

Métodos Aglomerativos

Consisten en generar unos árboles denominados dendrogramas.

Clustering jerárquico: No solo se generan grupos, sino también una jerarquía entre ellos.

Búsqueda hacia arriba:

1. Inicialmente, cada ejemplo representa una clase.
2. Se repite N-1 veces (siendo N el número de ejemplos):
   a) Se calcula la similitud entre todo par de ejemplos.
   b) Se agrupan los dos más cercanos.
   c) Se sustituyen los dos nodos por su representante/centroide/prototipo.

Distancia entre clústeres

Distintas definiciones:

- Mínima entre las instancias de los clústeres. $d_{min}(C_i, C_j) = \text{mín}_{\vec{x}\in C_i, \vec{y}\in C_j}\, d(\vec{x}, \vec{y})$
- Máxima. $d_{max}(C_i, C_j) = \text{máx}_{\vec{x}\in C_i, \vec{y}\in C_j}\, d(\vec{x}, \vec{y})$
- Media. $d_{media}(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{\vec{x}\in C_i, \vec{y}\in C_j} d(\vec{x}, \vec{y})$
- Como distancia entre sus centroides. $d_{centro}(C_i, C_j) = d(\vec{\mu_i}, \vec{\mu_j})$

6.3. Aprendizaje semi-supervisado

En algunos dominios etiquetar es pesado (páginas web, imágenes). La solución es etiquetar unas pocas instancias.

6.3.1. co-training [Blum and Mitchell, 1998]

Se tienen dos conjuntos de datos independientes con diferentes atributos, que entrenan clasificadores independientes, y se etiquetan las demás instancias con la clase que diga el clasificador que tenga más confianza y se repite el proceso.

6.3.2. CO-EM

Entrena con las clasificadas, clasifica el resto y el resultado se añade al conjunto de entrenamiento.

Realiza en bucle: Clasificar las instancias no etiquetadas y reconstruir el clasificador.

Es aprendizaje supervisado con las instancias etiquetadas.

7. TEMA 6: CONJUNTOS DE CLASIFICADORES Y REGLAS DE ASOCIACIÓN

7.1. Conjuntos de clasificadores - Emsembles

Se emplea en aprendizaje supervisado, la mayoría se pueden utilizar tanto en Clasificación como en Regresión, pero hay algunos más específicos.

Se tiene un grupo de clasificadores cuyas decisiones individuales se combinan de alguna manera. A menudo son mucho más precisos que cualquiera de los clasificadores individuales.

Los clasificadores individuales deben ser diversos (errores no correlados) y precisos (al menos mejor que una clasificación aleatoria).

Razones por las que se pueden encontrar buenos conjuntos:

- **Estadística:** Si construimos muchos modelos que no tengan mucha precisión, combinando los modelos estadísticamente se reduce el error.
- **Computacional:** Como cada hipótesis se construye de manera diferente, combinar varias búsquedas locales, nos permite encontrar un modelo final más preciso sin la necesidad de explorar todo el espacio de hipótesis.
- **Representacional:** Es posible que combinando distintas hipótesis (en la que alguna puede estar en un espacio de hipótesis que no contenga a la función) lleguemos a poder representar una mejor aproximación a la función real.

Métodos de construcción de conjuntos

- Manipulación de las instancias del conjunto de entrenamiento
  - Bagging
  - Comités de validación cruzada
  - AdaBoost (Boosting)
- Manipulación de los atributos (utilizar atributos distintos en los clasificadores)
  - Random forests
- Manipulación de las salidas/clases
  - Error correcting output coding - ECOC
- Meta-Aprendizaje
  - Stacking

7.1.1. Bagging - Boostrap aggregating [Breiman, 1996]

Construir distintos conjuntos de entrenamientos, generar los clasificadores y la salida será lo que diga la mayoría de los clasificadores.

Generan muestras de m instancias del conjunto de entrenamiento, donde cada muestra se genera haciendo selección aleatoria con remplazo (una instancia se puede coger en varios conjuntos) Se genera un clasificador con cada muestra.

Los clasificadores se pueden generar en paralelo, y se combinan las decisiones de los clasificadores por voto mayoritario.

Algoritmo de Bagging

Procedimiento Bagging (E, B, N), donde E es el conjunto de entrenamiento, B el algoritmo de aprendizaje y N el número de clasificadores que se generan (copias de E).

Para $i = 1$ hasta N:

E' = muestra de E (selección aleatoria con reemplazo).

$C_i = B(E')$ (crea un nuevo clasificador a partir de E').

Devolver clasificadores C_i.

7.1.2. Comités de validación cruzada

Se construyen conjuntos de entrenamiento de forma similar a validación cruzada (básicamente igual, pero cada uno de los clasificadores que se utilizaría para estimar el error es lo que se busca):

- Se dividen los ejemplos de entrenamiento en k conjuntos disjuntos.
- Se crean k conjuntos de entrenamiento en el que en cada uno se deja fuera uno de los conjuntos disjuntos.

7.1.3. Boosting [Shapire, 1990]

La idea es ir construyendo un clasificador (incremental) que basándose en lo que ha fallado previamente vaya mejorando, se especializan los clasificadores en ejemplos mal clasificados.

Se utiliza principalmente en clasificación binaria (+1 -1). Mejora la precisión de clasificadores débiles, como puede ser OneR.

El algoritmo de boosting más utilizado es AdaBoost - Adaptative Boosting.

Los clasificadores se generan de manera secuencial y se combina la decisión de los clasificadores por voto ponderado.

Algoritmo de Boosting

Se tienen unas instancias con unos pesos, se utiliza un algoritmo de aprendizaje que genera un modelo, se evalúan las instancias y según las mal clasificadas se recalculan los pesos de las instancias (que lo solucionaran los futuros clasificadores).

Procedimiento AdaBoostM1(E, B, T), donde E es el conjunto de entrenamiento E, B el algoritmo de aprendizaje base y T el número de iteraciones.

Inicializar pesos: $w(x_i) = \frac{1}{n}$. Inicialmente todas las instancias (x) tienen el mismo peso.

Para $t = 1$ hasta T:

- Normalizar pesos $w(x_i)$ (suman todas 1 en total)
- $C_t = B(E_t)$ Se construye un clasificador con el conjunto de entrenamiento con pesos.
- **Error por instancia:** $ierror(x_i)$ (0 si bien clasificado, 1 si mal clasificado)
- **Error del clasificador:** $error_{C_t} = \frac{\sum_i w(x_i) \cdot ierror(x_i)}{\sum_i w(x_i)}$ Medida ponderada, para tener en cuenta los pesos de las instancias y poder darles luego más importancia.
- **Peso del clasificador:** $peso_{C_t} = ln\left(\frac{1-error_{C_t}}{error_{C_t}}\right)$ Da más peso a los clasificados que cometen menos error (son más precisos)
- **Actualización de pesos:** $w(x_i) = w(x_i) \times e^{(peso_{C_t} \times ierror(x_i))}$ Da más peso a las instancias predichas incorrectamente. De esta manera se trata de que se llegue a clasificar bien (se va especializando)

Devolver clasificador C* donde cada clasificador se pondera por su peso.

De esta manera tenemos T clasificadores, a los que pasaremos instancias y tendremos en cuenta el peso del clasificador (ponderar el resultado) para elegir la clase final.

7.1.4. Manipulación de atributos

Se generan distintos clasificadores con distintos subconjuntos de atributos.

Problemas cuando eliminar atributos provoca un mal comportamiento de los clasificadores.

7.1.5. Bosques aleatorios (Random Forests) [Breiman, 2001]

Por un lado, se modifican instancias y por otros atributos.

Aprendizaje: Se construyen k árboles de decisión (regresión) aleatorios.

- elegir aleatoriamente $|E|$ ejemplos (con reemplazo) de los E originales.
- en cada nodo del árbol de decisión (regresión), elegir aleatoriamente $m \ll |A|$ atributos.
- usualmente $m = \sqrt{|A|}$.
- expandir el mejor atributo (de los m), los mejores estarán en la raíz.
- no se podan los árboles.

Clasificación: Se elige la moda de los votos de cada clasificador.

Regresión: Se hace la media.

7.1.6. Error Correcting Output Code (ECOC)

Manipulación de las salidas (clases)

Se utiliza para problemas con muchas clases como Reconocimiento de dígitos o clasificación de artículos de periódicos.

A cada clase se le asigna un código binario (vector de n bits), que se denomina un error correcting output code.

Se construye un clasificador binario para cada bit y para clasificar se utilizan todos los clasificadores para generar un código. Se devuelve la clase cuyo código sea más cercano al generado, para esto se puede usar la distancia de Hamming para cada bit con las clases.

7.1.7. Stacking - Stacked Generalization [Wolpert, 1992]

Genera los clasificadores a partir de algoritmos distintos y se combinan las decisiones de los clasificadores mediante otro clasificador, un meta-clasificador.

Tiene dos niveles de aprendizaje (Nivel-0 y Nivel-1)

- Nivel 0: Entrenar los clasificadores.
- Nivel 1: Entrenar el metaclasificador.

Entrenamiento:

- Divide los datos en k-folds de forma similar a validación cruzada.
- Se realizan k iteraciones. Se utilizan k-1 folds para entrenar los clasificadores de Nivel-0 y el otro para hacer predicciones (probar) de cada clasificador.
- El conjunto de entrenamiento de Nivel-1 se construye utilizando los atributos originales y las k predicciones para cada instancia (utiliza las predicciones de los clasificadores como atributos del metaclasificador, pero mantiene la clase de la instancia).

Al clasificar, primero se pasa por los clasificadores de nivel 0, para construir la instancia que se pasara al metaclasificador, que proporciona la clase final.

7.2. Reglas de asociación

Se emplea en aprendizaje no supervisado.

7.2.1. Conjuntos más frecuentes de elementos

Objetivo:

Dados un conjunto de ejemplos no etiquetados (en forma de conjunto de ítems) y un umbral de cobertura (coverage, support) de subconjuntos.

Trata de determinar el subconjunto de ítems que son más frecuentes (frequent itemsets).

Si un conjunto es **frecuente**, cualquier **subconjunto es también** frecuente.

Si un conjunto **NO es frecuente**, cualquier **superconjunto tampoco** lo es.

Teniendo unos conjuntos de elementos, podemos determinar la cobertura de un subconjunto de elementos según en cuantos de esos conjuntos aparecen esos elementos juntos.

Algoritmo APRIORI [Agrawal and Srikant, 1994]

$i = 1$

L_1 = encontrar todos los ítems que aparecen en al menos un C % de los ejemplos en E.

El proceso consiste en generar subconjuntos de tamaño i, y en L_i metemos aquellos que cumplen la cobertura mínima requerida, y en L_{i+1} se hacen subconjuntos de tamaño i+1 con los L anteriores.

Repetir

$i = i + 1$

S_i = encontrar todos los subconjuntos de i ítems, combinando conjuntos frecuentes previos

L_i = podar aquellos elementos de S_i que aparecen en menos de un C % de los ejemplos en E

Hasta qué $L_i = \emptyset$. Devolver la unión de todos los L_i

Los subconjuntos de salida son aquellos que no son subconjuntos de otros, partiendo de coger el del último L no vacío.

7.2.2. Reglas de asociación

Objetivo: Dados conjuntos de ejemplos, trata de determinar reglas que relacionen los elementos.

Se suelen utilizar como entrada los conjuntos más frecuentes.

Se calcula la precisión (confianza) de cada regla.

Se suelen escoger las que maximicen la precisión, las que sobrepasen un umbral de precisión y las N que tengan mayor precisión.

$$X \Rightarrow Y$$

Soporte de la regla: número de veces que el par antecedente-consecuente se dan juntos en los ejemplos.

$$C = \frac{|X, Y|}{|N|}$$

Precisión (confianza) de la regla: número de veces que el par antecedente-consecuente se dan juntos en los ejemplos sobre el número de veces que el antecedente es cierto.

$$P = \frac{|X, Y|}{|X|}$$

La idea es elegir las reglas cuya cobertura es mayor que un umbral de cobertura y cuya precisión sea mayor que un umbral de precisión.

Si se parte de ejemplos atributo-valor: Se define un elemento por cada valor del atributo (Si salario puede ser alto o bajo, los elementos serán salarioAlto y salarioBajo), de tal manera que un ejemplo/instancia se define como el conjunto de los elementos de sus valores de atributos (sin que haya más de un elemento que se refiera al mismo atributo). Se ejecuta APRIORI: genera reglas de asociación.

Ejemplo:

Si número-habitacionesAlto Y salarioMedio **Entonces** número-hijosAlto

Si salarioAlto **Entonces** precio-viviendaAlto

Ejemplo completo:

- Entrada

  carro1: leche, cereales, jamón, queso, pan-de-molde

  carro2: cerveza, leche, jamón, queso, pan-de-molde

  carro3: panales, leche, cereales, potitos

  carro4: leche, queso, pasta-dientes, peine

- Conjuntos más frecuentes, C = 0,5

  L1 = leche, queso, jamón, cereales, pan-de-molde

  L2 = (leche, queso), (leche, jamón), (leche, cereales), (leche, pan-de-molde), (queso, jamón), (queso, pan-de-molde), (jamón, pan-de-molde)

  L3 = (leche, queso, jamón), (leche, queso, pan-de-molde), (leche, jamón, pan-de-molde), (queso, jamón, pan-de-molde)

  L4 = (leche, queso, jamón, pan-de-molde)

  L5 = ∅;

- Salida (los que no son subconjuntos de otros):

  leche, cereales

  leche, queso, jamón, pan-de-molde

- Reglas de asociación

  R1: leche → cereales (C=2/4, P=2/4)

  R2: cereales → leche (C=2/4, P=2/2)

  R3: leche, queso, jamón → pan-de-molde (C=2/4, P=2/4)

  R4: leche, queso, pan-de-molde → jamón (C=2/4, P=2/4)

  R5: leche, jamón, pan-de-molde → queso (C=2/4, P=2/4)

  R6: queso, jamón, pan-de-molde → leche (C=2/4, P=2/4)

  R7: leche, queso → jamón, pan-de-molde (C=2/4, P=2/3)

R8: leche, jamón → queso, pan-de-molde (C=2/4, P=2/2)

R9: leche, pan-de-molde → queso, jamón (C=2/4, P=2/2)

R10: queso, jamón → leche, pan-de-molde (C=2/4, P=2/2)

R11: queso, pan-de-molde → leche, jamón (C=2/4, P=2/2)

R12: jamón, pan-de-molde → leche, queso (C=2/4, P=2/2)

R13: leche → queso, jamón, pan-de-molde (C=2/4, P=2/4)

R14: queso → leche, jamón, pan-de-molde (C=2/4, P=2/3)

R15: jamón → leche, queso, pan-de-molde (C=2/4, P=2/2)

R16: pan-de-molde → leche, queso, jamón (C=2/4, P=2/2)

8. TEMA 7: APRENDIZAJE POR REFUERZO

Consiste en aprender a decidir, ante una situación determinada, que acción es la más adecuada para lograr un objetivo.

Se utiliza una Política que acción, que dado un estado del agente nos diga cuál es la acción debe ejecutar, $\pi : s \rightarrow a$.

Se construye a partir de experiencias de interacción con el entorno: $< s_i, a_i, s'_i, r_i >$ (estado actual, acción, estado siguiente, refuerzo)

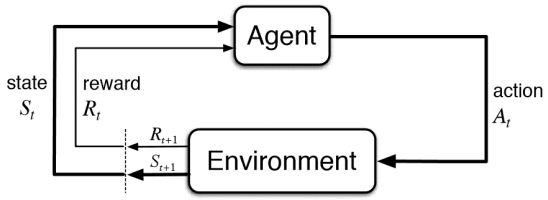


Fig. 8.1: Diagrama Aprendizaje por refuerzo

Mediante un proceso iterativo de prueba y error, en que se van proporcionando señales de refuerzo que premia (positivo) o castigan (negativo) las decisiones en el proceso de maximizar el refuerzo.

Por debajo del aprendizaje por refuerzo hay un Proceso de Decisión de Márkov.

8.1. MDP - Markov Decision Process

Se tiene una serie de estados en los que se pueden realizar unas determinadas acciones, aunque también se hay estados absorbentes en los que no se puede aplicar ninguna acción.

Las acciones **deterministas** siempre produce la misma transición de estado y el mismo refuerzo, sin embargo, en los **no deterministas/estocástico** un mismo estado y una acción pueden producir distintos resultados, son funciones estocásticas.

Se define con la tupla $< S, A, T, R >$:

- S - Conjunto de Estados.
- A - Conjunto de Acciones.
- T - Función de Transición $T(s, a, s')$; dado un estado, una acción y el estado siguiente nos dice la probabilidad de que ocurra esa transición.
- R - Función de Refuerzo $R : S \times A \times S \rightarrow \mathbb{R}$, proporciona el refuerzo recibido al ejecutar la acción a en estado s y llegar al s'.

8.1.1. Propiedad de Márkov

Se asume que se cumple esta propiedad cuando realizamos un MDP.

Propiedad de Márkov: El pasado más allá del estado inmediatamente anterior no son relevantes, la acción a ejecutar solo depende del estado actual. El estado actual y el refuerzo obtenido son condicionalmente independientes de la historia pasada.

$$P(s_{t+1}, r_{t+1}|s_t, a_t)$$

Fig. 8.2: Propiedad de Márkov

8.1.2. Métodos de Resolución

Hay dos posibilidades:

- **Se conoce el modelo**, la Función de Transición de estados y la Función de Refuerzo del MDP.

  En este caso se utiliza **Programación Dinámica**, hay dos opciones.

  - **Iteración de valor.**
  - **Iteración de la política.**

- **No se conoce el modelo**, hay dos opciones:

  - **Aprender el modelo** y usar métodos basados en el modelo, Programación Dinámica.
  - **Aprender las funciones de valor y/o políticas directamente**: métodos libres de modelo o Aprendizaje por Refuerzo.

8.1.3. Políticas y Optimalidad

El **objetivo es encontrar una política**, $\pi : S \rightarrow A$, que nos diga para cada estado posible cuál es la acción que debe ejecutarse para maximizar el refuerzo acumulado en el tiempo.

Criterio de optimalidad de horizonte infinito descontado: Asumiendo que se suma hasta infinito, se valora más que se obtenga el mayor refuerzo lo antes posible, para esto se utiliza un factor de descuento, $0 \leq \gamma \leq 1$, que reduce la importancia de las ganancias futuras.

$$\sum_{k=0}^{\infty} \gamma^k r_{t+1+k}$$

Función de Valor (V) - valor-estado

Dada una política π y un estado inicial s, $V^{\pi}(s)$ representa el refuerzo descontado en el tiempo si se aplica la política π partiendo de s. En el caso no determinista es un valor esperado.

$$V^{\pi}(s_t) = E\left[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k}\right]$$

Fig. 8.3: Función de Valor

Política óptima

La política óptima π^*, es aquella que maximiza $V^{\pi}(s)$ para todos los estados. $V^*(s)$ es máximo refuerzo descontado acumulado en el tiempo que el agente puede conseguir comenzando en el estado s y siguiendo la política óptima π^*.

$$\pi^* \equiv \arg\max_{\pi} V^{\pi}(s), \forall s \equiv V^*(s)$$

Función de valor-acción (Q)

Dada una política π, un estado inicial s y una acción a, $Q^{\pi}(s,a)$ representa el refuerzo descontado acumulado en el tiempo si se aplica la acción a en el estado s, y a partir de ahí se ejecuta la política π.

$$\begin{aligned} Q^*(s,a) &= \max_{\pi} Q^{\pi}(s,a), \forall s \in S, \forall a \in A \\ V^*(s) &= \max_{a} Q^*(s,a), \forall s \in S \end{aligned}$$

Fig. 8.4: Función de valor-acción

Política óptima en función de Q, maximiza el refuerzo de cada acción que se puede ejecutar en ese estado.

$$\pi^*(s) = \arg\max_{a \in A} Q^*(s,a), \forall s \in S$$

Ecuaciones de Bellman

Ecuaciones recurrentes que se basan en calcular el refuerzo total óptimo maximizando sobre la elección de una primera acción y considerando un futuro óptimo.

$$\begin{aligned} Q^*(s,a) &= \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')] \\ V^*(s) &= \max_a Q^*(s,a) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')] \end{aligned}$$

Fig. 8.5: Ecuaciones de Bellman

8.1.4. Resolver el MDP: Aproximaciones Basadas en el Modelo

La Función de Transición y la Función de Refuerzo son conocidas, se trata de resolver las ecuaciones de Bellman mediante Programación Dinámica y tenemos dos opciones que resuelven el mismo problema:

Algoritmo Iteración de valor - Value Iteration

- Inicializar: $V(s) = 0$ para todo s
- Repetir hasta convergencia
    1. Dados $V(s)$ para todo s, actualizar los valores para la nueva iteración teniendo en cuenta los de la anterior.

$$V(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')], \forall s$$

    2. La política generada en la iteración consiste en quedarnos con la acción que genera el mayor valor para cada estado posible. Aunque solo nos interesará en la última iteración que será la óptima.

$$\pi(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')], \forall s$$

Cuando el algoritmo converge (los valores se estabilizan) la política es óptima.

Algoritmo Iteración de política - Policy Iteration

Es como hacer Iteración valor, pero en la Ecuación de Bellman en vez de evaluar todas las acciones se evalúa solo la acción que proporciona la política que vamos actualizando, en cuanto al cálculo de la política es igual, por lo que debemos tener en cuenta todas las acciones en todos los estados (lo que parecía que nos ahorrábamos calcular, se calcula ahora).

- Elegir una política π arbitraria (aleatoria)
- Repetir hasta convergencia
    1. Evaluación de la política: resolver las ecuaciones $\forall s \in S$

$$V(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^*(s')]$$

    2. Mejora de la política $\forall s \in S$

$$\pi(s) = \arg\max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')], \forall s$$

Cuando el algoritmo converge (la política se estabiliza) la política es óptima.

8.2. Aprendizaje por Refuerzo

8.2.1. Modelo desconocido

Problema de Aprendizaje por Refuerzo (definido como un MDP):

- Conjunto de todos los posibles estados, S.
- Conjunto de todas las posibles acciones, A.
- Función de Transición de estados desconocida $T : S \times A \times S \rightarrow \mathbb{R}$.
- Función de Refuerzo $R : S \times A \times S \rightarrow \mathbb{R}$.

Objetivo: aprender la política de acción,$\pi : S \rightarrow A$, que maximice el refuerzo esperado acumulado en el tiempo mediante exploración del entorno.

8.2.2. Aproximaciones Basadas en el Modelo

1. Se aprende el modelo (T y R) y se resuelven las ecuaciones por Programación Dinámica (Iteración de valor o Política).
    - Técnicas costosas computacionalmente.
    - No útil para respuestas en tiempo real
    - Se debe asegurar que se aprende el entorno completamente.
    - No es sensible a cambios en el entorno
2. Se aprende el modelo a la vez que la función Q (algoritmo Dyna-Q).

8.2.3. Aproximaciones libres de Modelo

Aprende directamente del entorno mediante prueba y error.

- Actualización directa de la función de valor Q a partir de interacciones con el entorno.
- Basados en procesos de prueba y error.
- NO aprenden el MDP subyacente.

Métodos Monte Carlo y métodos de Diferencia Temporal (Q-learning, SARSA)

8.2.4. Métodos Monte Carlo (MC)

El objetivo es estimar Q^*, el refuerzo que voy a obtener a futuro si aplico la opción óptima.

Basado en:

- Alternar la evaluación de política y su mejora, en sucesivas ejecuciones de episodios de aprendizaje.
- Actualización de Q basada en la media de los refuerzos en los episodios.
- Actualiza Q al final de cada episodio.

Monte Carlo con Arranque Exploratorio

- Inicializar, $\forall s \in S, \forall a \in A$:
    - $Q(s, a)$ valor arbitrario.
    - $\pi(s)$ valor arbitrario.

- $ganancias(s, a)$ lista vacía.

- Repetir para siempre:

  1. Generar un episodio usando arranque exploratorio (comienza en un estado aleatorio) y la política π.
  2. Para cada par (s, a) que aparece en el episodio:
     - $R(s, a) = r + \gamma r' + \gamma r'' + \gamma r''' + \ldots$ refuerzo descontado acumulado tras (empezando a descontando con el refuerzo siguiente) la PRIMERA ocurrencia del par (s, a).
     - Añadir $R(s, a)$ a $ganancias(s, a)$, cada par tiene su lista de refuerzos almacenados.
     - $G = promedio(ganancias(s, a))$
     - $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha G$ (Caso determinista $\alpha = 1$, $Q(s, a) \leftarrow G$)
  3. Para cada s en el episodio: $\pi(s) \leftarrow \arg\max_a Q(s, a)$

8.2.5. Métodos de Diferencia Temporal (TD)

El objetivo es estimar Q^*.

Combinación de las ideas de Programación Dinámica y los métodos de Monte Carlo:

- Aprendizaje por prueba y error. Estimaciones sobre estimaciones.
- Basado en el cálculo de las funciones de valor-acción Q.
- Se actualiza Q en el paso del episodio.

Algoritmos:

- Q-Learning: off-policy (no lleva una política que vaya actualizando)
- SARSA (State–action–reward–state–action): on-policy

Q-Learning

Se basa en estimar de forma iterativa la función de valor-acción óptima, $Q^*(s, a)$.

La función Q la podemos representar en forma de tabla con |S| filas y |A| columnas, en cada casilla se pondrá el refuerzo acumulado en el tiempo cuando en ese estado se ejecuta esa acción asumiendo refuerzo futuro óptimo.

Esta tabla que se genera nos facilita obtener la política óptima, de tal manera que la mejor acción de cada estado es el máximo de su fila.

$$\pi(s) = \arg\max_a Q^*(s, a)$$

Consiste en que después de cada transición se recalcula la tabla Q entera aproximándose poco a poco a la óptima, y se van generando episodios de aprendizaje, una serie de transiciones consecutivas hasta que termina o se para. $\underbrace{sas'}\ \underbrace{a'r's''}\ \underbrace{a''r''s'''}\ ...$

Funciones de Actualización de Q

- Función de Actualización **determinista**

$$Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s',a')$$

Fig. 8.6: Función de Actualización de Q Determinista

- Función de actualización **no determinista** (en general)

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a')]$$

Fig. 8.7: Función de Actualización de Q No Determinista

$\alpha \in [0,1]$ representa la tasa de aprendizaje, la agresividad con la que se actualiza la tabla.

γ es el factor de descuento, como van perdiendo importancia las decisiones más alejadas del estado actual del agente.

Algoritmo de Q-Learning(γ, α) En el que se realizan varios escenarios para ir mejorando la tabla Q, en cada uno de los escenarios se va evaluando la mejor acción para generar un estado completo (s, a, r, s') y se actualiza la tabla antes de transitar a él.

- Inicializar $Q(s,a), \forall s \in S, a \in A$ (lo habitual es inicializar a 0)
- Repetir (para cada episodio)
  - Inicializa el estado inicial, s, aleatoriamente.
  - Repetir (para cada paso del episodio)
    - Selecciona una acción a y la ejecuta.
    - Recibe el estado actual s', y el refuerzo, r.
    - $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a')]$
    - Asigna $s \leftarrow s'$
- Devuelve $Q(s,a)$.

8.2.6. Exploración vs. Explotación

Exploración: Acciones aleatorias.

Explotación: Utilizar acciones de la política actual $\pi(s) = \arg\max_a Q(s, a)$

Estrategias principales de selección de acciones, para balancear la exploración y la explotación sesgando la selección de acciones con conocimiento del dominio:

- $\epsilon-greedy$**:** Ejecuta una acción aleatoria con probabilidad ϵ y la acción $\arg\max_a Q(s, a)$ con probabilidad $1 - \epsilon$
- **Softmax:** A cada acción que se puede ejecutar en un estado se le asigna una probabilidad.

$$P(a_i|s) = \frac{e^{Q(s,a_i)/\tau}}{\sum_{a_j \in A} e^{Q(s,a_j)/\tau}}$$

  Se utiliza un parámetro llamado Temperatura τ, si es alta la acción es equiprobable (exploración pura), si es media es más probabilidad la que tiene mayor $Q(s, a)$ y si es baja se sigue la política (exploración pura).

  Lo ideal es que de más probabilidad a aquellas con mayor $Q(s, a)$, de tal manera que es aleatorio, pero la mejor acción es más probable.

8.2.7. Aplicaciones

Tiene muchas aplicaciones entre las que destaca la planificación en tiempo real y para entornos estocásticos.

- Control de dispositivos de salud.
- Control de robots.
- Control de procesos de fabricación.
- Control de temperatura.
- Predicción de series.
- Juegos: AlphaGo, TD-Gammon.
- Logística.
- Problemas canónicos: navegación en un grid, balance de un sistema cart-pole.

8.3. Representación de la función Q

8.3.1. Pac-Man

Observaciones: coordenadas discretas x e y de Pac-Man y fantasmas.

Acciones: movimientos de tamaño 1 en las 4 direcciones.

Objetivo: comer fantasmas.

8.3.2. Representación Tabular de la Función Q

Problema: espacio de estados continuo o de gran tamaño.

Solución: métodos de generalización, abstracción:

- **Aproximaciones ad-hoc** basadas en conocimiento del dominio.
- **Discretización** del espacio de estados, pasar de continuo a discreto.
- **Aproximación de funciones**, en vez de una tabla discreta que sea una función.

8.3.3. Mountain-Cart

Observaciones: valores continuos de posición x y velocidad v del coche.

Acciones: aplicación de fuerza hacia la derecha, izquierda, o nula.

Objetivo: alcanzar la bandera con velocidad 0.

8.4. Generalización en Aprendizaje por Refuerzo

8.4.1. Discretización del Espacio de Estados

Los problemas que surgen al discretizar son:

- Las discretizaciones erróneas pueden romper fácilmente la propiedad de Márkov.

  Al discretizar puede dejar de ser determinista, ya que puede ocurrir que al hacer una acción no se salga siempre del estado (como ocurriría en Pac-man si en vez de ir casilla a casilla son de 2x2).
- El número de regiones necesarias para discretizar el espacio de estados.

8.4.2. Discretización uniforme del espacio de estados

Discretizar cada atributo en un número dado de niveles de discretización o regiones, todas las regiones del mismo tamaño.

Esto supone un problema porque no se tiene en cuenta el número de ejemplos por región, por lo que puede haber regiones que no tengan ninguno y otras que estén superpobladas.

Se debería especializar más en las que tienen más ejemplos.

8.4.3. Discretización de Resolución Variable: Árboles KD

La discretización más fina en las regiones que más ejemplos hay.

Los nodos y hojas del árbol representan regiones del espacio de estados.

En cada nodo del árbol, una región se divide en otras dos, esta división se realizará según un criterio.

Con esta técnica una región se irá partiendo en dos hasta que cumpla las condiciones establecidas, y se crearan regiones más grandes para las menos pobladas y más pequeñas para las superpobladas.

8.4.4. Clustering (Agrupamiento) para Q-Learning

1. Discretización no uniforme del espacio de estados (k-means).
2. Los clústeres obtenidos se consideran la nueva representación del espacio de estados.

   $sars' \rightarrow C_5 arC_7$
3. Se aplica Q-Learning sobre la nueva representación.

9. TEMA 8: PROGRAMACIÓN LÓGICA INDUCTIVA - ILP

Aprendizaje para cuando tenemos los datos expresados de manera relacional.

Se hace cuando el conocimiento no se puede representar como atributo-valor o tengan relación entre los datos difíciles de expresar.

Método que emplearemos: FOIL - First Order Inductive Learner (Quinlan 1990).

Es aprendizaje supervisado, inductivo, generaliza con los ejemplos.

9.1. Aprendizaje Relacional

TILDE - Árboles de decisión.

WARMR - Reglas de Asociación, conjuntos frecuentes.

9.2. Proposicionalización

Lo que se puede hacer con la información relacional es pasarla a proposicional, hace mediante:

- **Agregación**: Sumar los valores de distintos atributos para juntarlos en un atributo, el problema es que se pierde información.
- **Introducir atributos nuevos** en la tabla original, tantos como registros haya. El problema es que muchos atributos tendrán valores desconocidos.

9.3. Elementos de representación

- **Términos**: Cualquier secuencia que empieza en minúsculas es una constante y en mayúsculas es una variable. También funciones que tiene como parámetros constantes y variables. a, X y $fun(X, Y)$
- **Fórmulas atómicas**: Predicados definidos sobre términos. $tipo - cliente(X, bueno)$
- **Literal**: Fórmula atómica o su negación ($\neg$).
- **Clausula**: Disyunción de literales, oes de literales. $cliente(X, Y) \vee cliente(X, Z)$
- **Fórmulas bien formadas**: Fórmulas atómicas unidas por conectivas ($\wedge, \vee, \rightarrow, \neg$) y cuantificadas ($\forall, \exists$)

- **Clausula de Horn:** Clausula, disyunción de literales, que como mucho tiene un literal positivo. $\neg p \vee \neg q \vee \neg r \vee s$ que es $p \wedge q \wedge r \rightarrow s$ ($cuerpo \rightarrow cabeza$ que en prolog es $consecuente :- antecedente$).
- **Regla de Horn:** Conjunto de cláusulas de Horn. Si se da una de las cláusulas se cumple la regla.
- **Definición implícita:** Definido como una regla de Horn, de manera genérica.
- **Definición explicita:** Definido como todos los argumentos válidos que hace que se cumplan.

9.4. Generalización simple. Subsunción

Dada una sustitución $\theta = \{v_1/t_1, ..., v_n/t_n\}$ y una fórmula F, $F\theta$ es el resultado de aplicar la sustitución sobre F.

Que una sentencia sea más general que otra quiere decir que conociendo la general se puede deducir el otro.

Átomo A subsume al átomo B, $A \geq B$, si al aplicar la sustitución sobre A es igual a B.

Cláusula C subsume a la cláusula D, $C \geq D$, si al aplicar la sustitución sobre C está contenida en D.

9.5. Generalización como consecuencia lógica

Consecuencia lógica: La consecuencia lógica se indica con $C \vDash D$, indica que conocido el antecedente se puede deducir el consecuente.

Consecuencia lógica relativa: La consecuencia lógica se indica con $B, C \vDash D$, indica que conocido el antecedente y un conocimiento base se puede deducir el consecuente.

9.6. ILP - Programación Inductiva Lógica

Lo que buscamos es generar una hipótesis de manera que junto al conocimiento base, los ejemplos sean una consecuencia lógica relativa.

$$B, H \vDash E$$

- **B**: Conocimiento base.
- **E**: Conjunto de Ejemplos.
- **H**: Hipótesis.

Se trata de una búsqueda en el espacio de cláusulas que se llama **Refinamiento de Cláusulas**, los algoritmos que se emplean son:

- Más específico a más general, como GOLEM.
- Más general a más específico, como FOIL o PROLOG.

9.7. FOIL - First Order Inductive Learner

Aprendizaje inductivo de reglas de Horn, de más general a más específico, mediante una búsqueda en escalada con heurística en el espacio de cláusulas.

9.7.1. Algoritmo de FOIL

Función FOIL (E+, E-, P, B): REGLA

E^+ y E^- ejemplos positivos y negativos del concepto meta resp., P predicado a aprender, B conocimiento de base.

- Hasta que $E^+ = \emptyset$, es decir que están cubiertos por alguna cláusula:
  - $N = E^-$
  - Clausula limpia, $CUERPO = \emptyset$
  - Hasta que $N = \emptyset$, que no deje pasar ejemplos negativos
    - $L = literal - maxima - ganancia(E^+, E^-, P, B, N, CUERPO)$
    - Se añade un literal al cuerpo, $CUERPO = CUERPO, L$
    - Se quitan de N los ejemplos negativos que no deja pasar el literal, $N = N - \{e \in N | L \nvdash e\}$
  - Se añade la cláusula a las reglas, $REGLA = REGLA \cup [P :- CUERPO]$
  - Se quitan los ejemplos positivos cubiertos por la regla, $E^+ = E^+ - \{e \in E^+ | CUERPO \vDash e\}$
- Devolver *REGLA*

9.7.2. Heurística de FOIL

$$G(L) = k \times [I(n^+, n^-) - I(n_L^+, n_L^-)]$$

- $G(L)$: Ganancia que se obtiene al añadir el literal L a la cláusula.
- k: Número de ejemplos positivos que se cumplen con el conocimiento base B con L.
- n^+: Número de tuplas de la cláusula que hacen ciertos ejemplos positivos.
- n^-: Número de tuplas de la cláusula que hacen ciertos ejemplos negativos.
- n_L^+: Número de tuplas de la cláusula con el L que hacen ciertos ejemplos positivos.
- n_L^-: Número de tuplas de la cláusula con el L que hacen ciertos ejemplos negativos.
- $I(x, y) = -\log_2\left(\frac{x}{x-y}\right)$ la x son las positivas y la y la negativa.

9.7.3. Tipos de literales

- Literales: $Q(V_1, V_2, ..., V_n)$, $\neg Q(V_1, V_2, ..., V_n)$.
- Desigualdades: $X_i < c, X_i < X_j, X_i > c, X_i > X_j, X_i <= c, X_i <= X_j, X_i >= c, X_i >= X_j$.
- Igualdades: $X_i = c, X_i = X_j, X_i \neq c, X_i \neq X_j$.

Los literales deben proceder de B.

En los literales se pueden utilizar variables nuevas.

En las comparaciones o desigualdades se usan variables que aparecen previamente.

9.7.4. Otras restricciones

- Q debe tener al menos una variable ya existente
- No permite funciones ni constantes en las reglas aprendidas.
- El conocimiento debe estar definido extensionalmente, que es costoso.
- Se puede hacer poda cuando la ganancia este por debajo de un umbral.
- FOIL permite obtener definiciones recursivas.

9.7.5. Sesgo o bias

El espacio de búsqueda es mucho mayor que con representación atributo-valor, por lo que es necesario restringir:

- El lenguaje en las cláusulas, no permitiendo funciones, ni contantes o solo atributos nuevos en los literales (al menos uno conocido).
- Profundidad de recursión o número de variables.
- Predicados procedentes del conocimiento base.
- Heurísticas: definida por la función de ganancia.
- Modos de los predicados.

9.7.6. Otros algoritmos relacionales

- Reglas de decisión relacionales: PROGOL, GOLEM
- Árboles de decisión relacionales: TILDE
- Árboles de regresión relacionales: S-CART, TILDE-RT
- Basado en instancias relacionales: RIBL
- Aprendizaje modelos probabilísticos relacionales
- Por refuerzo relacional: RRL
- Clustering relacional: FORC=K-MEDIAS relacional
- Reglas de asociación o conjuntos frecuentes relacionales: APRIORI, WARMR
- Aprendizaje de atributos relevantes con ILP: LINUS
- Híbridos: árboles de decisión con IBL en los nodos hoja

10. RESUMEN

10.1. Selección

Clase etiqueta: Árboles y reglas de decisión, KNN, Ensembles, Naïve Bayes, FOIL.

Probabilidad de las clases o text mining: Naïve Bayes.

Muchos datos y difícil atributo-valor: Relacional.

Clase numérica: M5, CART, Ensembles o KNN.

Agrupar elementos: k-medias.

Agrupar jerárquicamente o grupos: Aglomerativos.

Asociaciones o elementos por su frecuencia: APRIORI y Reglas de asociación.

Tomar decisiones, transiciones y recompensa: Q-learning, Monte Carlo o Dyna-Q

Tomar decisiones, pero (S, A, T, R) conocido: Programación dinámica.

10.2. Aprendizaje supervisado

Si va a haber atributos con valores vacíos usar representación relacional.

- **Árboles de reglas de decisión:** Definir el nivel de poda, limpieza de los atributos y la selección de los atributos
  - **ID3:** No con atributos numéricos o discretos con muchos valores.
  - **C4.5 y J48:** Con atributos numéricos o cuando se tengan muchos.
- **Aprendizaje basado en instancias (IBK):** KNN.
  - Definir el valor de k y la medida de similitud.
  - Limpiar los atributos con wrapper, normalizar o ponderar (si no se normaliza)
  - Limpiar el conjunto de ejemplos, borrando o con centroides.
- **Naïve Bayes (text mining):** Clasificación probabilística.
  - No hay parámetros que configurar, es calcular las tablas.
  - Permite trabajar con valores continuos (no hace falta discretizar)
- **Deep Learning (imágenes y sonido):** Trabaja con los datos en bruto.
- **Relacional: FOIL**, es aprendizaje relacional y hay que definir Conocimiento base, el predicado a aprender y los ejemplos que se usaran para entrenar.

Tipos:

- **Clasificación**
  - Árboles y reglas de Decisión: ID3, C4.5, J48
  - Aprendizaje basado en instancias: KNN
  - Conjuntos de clasificadores
  - Naïve Bayes y Clasificador Bayesiano
  - Relacional: FOIL
- **Regresión**
  - Aprendizaje basado en instancias: KNN
  - CART y M5
  - Conjuntos de clasificadores
  - Naïve Bayes, Redes neuronales, Deep Learning
- **En relacional mirar la lista**, tiene posibles para todos.

10.3. Aprendizaje no supervisado

- **Clustering:** k-medias
  - Definir el valor de k, medida de distancia, semillas y normalización/ponderar.
- **Aglomerativos:** Clustering jerárquico, que genera un dendrograma.
  - Definir medida de distancia, como tratar jerarquía y normalización/ponderar.
- **Reglas de asociación y Conjuntos frecuentes:** APRIORI, aprendizaje asociativo
  - Definir la cobertura.
  - Si hay valores numéricos continuos discretizar.
  - Explicar: primero conjuntos frecuentes y genera las reglas.
  - Definir como será las reglas objetiva.
  - Si se quiere tener en cuenta el orden, se busca no solo que aparezcan sino también el orden. Cuando aumenta el tamaño del conjunto salen más combinaciones.

10.4. Aprendizaje por refuerzo

Definir Estados, Acciones, función de Transición (si es conocida) y función de Refuerzo (retardada casi siempre, los inmediatos llevan a comportamientos cortoplacistas).

- **Q-learning**
  - Indicar el factor de reducción γ y de aprendizaje α si no es determinista.
  - Determinar que exploración/explotación usaremos: ϵ-greedy o Softmax
  - Si la situación para la que se crea es crítica, entrenarlo en un simulador realista.
- **Basada en el modelo: Dyna-Q**, no es buena para tiempo real.

Factores que apuntan a este aprendizaje:

- Tarea de toma de decisiones secuenciales.
- Se puede entender un MDP subyacente.
- No se conoce a priori cuál es la mejor acción en cada situación.
- Que sea de prueba y error.
- El beneficio/impacto solo se conoce al final del episodio.

10.5. Normalizar valores numéricos

$$z_i = \frac{x_i - \text{mín}\, x}{\text{máx}\, x - \text{mín}\, x}$$

10.6. Tratamiento de valores nominales en distancias

- **Binarios:** Asignar a cada opción uno de los valores y hacer Hamming, 1 o 0.
- **Ordinarios:** Primaria, ESO o Bachillerato. Asignar números según el orden y usarlos normal; 0, 1 y 3.
- **Categóricos:** Rojo, Verde o Azul. Asignar números arbitrarios y usarlos normal; 1, 2 y 3.

Distancia Hamming: En vez de poner $(x_i - x_j)^2$ en la distancia euclídea se pone 0 o 1.

- 0 si $x_i = x_j$
- 1 si $x_i \neq x_j$