Métodos basados en instancias

K-vecinos, variantes

Contenido

1. Caracterización
2. K-vecinos más próximos
3. Mejoras al algoritmo básico
4. Bibliografía

1. Caracterización

- Forma más sencilla de aprendizaje: aprendizaje memorístico.

- Aprendizaje basado en instancias:
  - *Instance Based Learning*, IBL
  - Memoriza los ejemplos de entrenamiento y su clase.
    - No obtienen una descripción estructural del concepto: los propios ejemplos constituyen el conocimiento del sistema.
  - Realiza una clasificación en base a la similitud de la instancia a clasificar con los ejemplos memorizados.

- También denominado basado en casos, basado en ejemplares o perezoso (lazy).

Proceso de aprendizaje en IBL

- Entrenamiento: memorización.
  - En los casos básicos, se limita a almacenar los ejemplos de entrenamiento y su clase.
- El proceso de generalización se realiza en tiempo de clasificación.
  - Denominados perezosos –lazy- porque retrasan el procesamiento hasta que hay que clasificar una nueva instancia.
- Clasificación: dos fases
  - Selección de instancias según medida de similitud.
  - Resultado en función de las instancias seleccionadas.

Características IBL

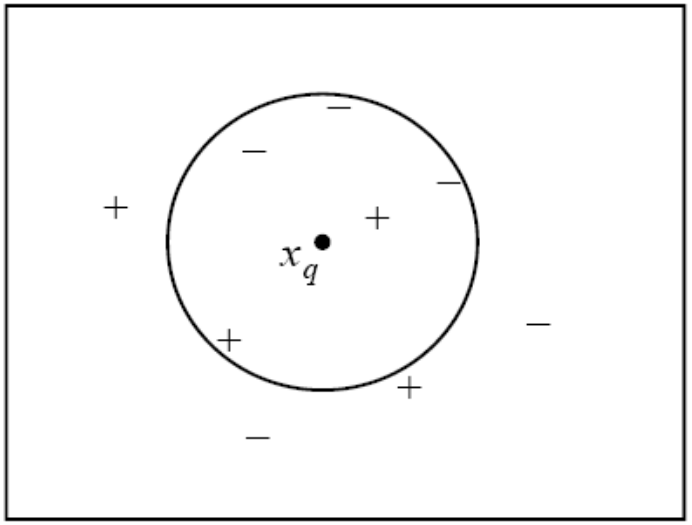
- Carácter incremental: basta con almacenar nuevas instancias.
- Coste entrenamiento: mínimo.
- Clasificación costosa.
  - Determinar ejemplos más cercanos.
- Clasificación y regresión –ajuste de funciones-.
- Aproximaciones locales al concepto objetivo, posiblemente diferentes para cada nueva instancia a clasificar.
- Descripción implícita del concepto objetivo

2. *K*-Vecinos más próximos

- *K*-NN, algoritmo básico IBL
- Efectivo, popular, bien conocido, sencillo, asentado (1967).
- Trata las instancias como puntos en un espacio n-dimensional.
- Almacena todos los ejemplos de entrenamiento.
- Define la distancia entre instancias – euclídea, manhattan-
- Selecciona las *k* instancias más próximas.
- Asigna la clase mayoritaria.

K-NN: ejemplo clasificación

- Ejemplos de entrenamiento: +, -
- Nueva instancia a clasificar: x_q
- 1-NN: asigna +
- 5-NN asigna -



Espacio de hipótesis

- Descripción de instancias: pares atributo=valor, continuos
- Espacio de instancias: R^n
- Hipótesis: Si clase de decisión binaria, función booleana de *n* variables reales
  - Función descrita por instancias memorizadas y procedimiento de clasificación.

- Espacio de hipótesis:
  - Conjunto de todas las funciones booleanas de *n* variables

- Tamaño espacio de hipótesis:
  - Atributos continuos: infinito

Tarea inducción en *k*-vecinos más próximos

- Dados
  - Descripción de Instancias, *X*, (atributos, valores)
  - Descripción de las Hipótesis, *H (*funciones booleanas de **n** variables, descritas por ejemplos memorizados y procedimiento clasificación*)*
  - Concepto objetivo, c : *X* → {0,1}
  - Ejemplos positivos y negativos, *D*, pares (<x, c(x)>)

- Determinar
  - Hipótesis *h* de *H* / *h(x) = c(x)* para todo *x* de *X*

Algoritmo básico

- Entrenamiento
  - Añadir cada ejemplo $(\underline{x}_j, c(\underline{x}_j))$ a la lista Ejemplos_Entrenamiento
- Clasificación
  - Sea $\underline{x}_q$ la nueva instancia a clasificar
  - Obtener las *k* instancias más próximas a $\underline{x}_q$ de Ejemplos_Entrenamiento: $\underline{x}_1, \underline{x}_2 \ldots \underline{x}_k$
  - $h(\underline{x}) = argmax_{v \in V} (\Sigma_{i=1,\ k}\ \delta(v, c(\underline{x}_i)))$

  Con:
  - $\delta(a, b) = 1$ *si* $a=b$; 0 en otro caso
  - $\underline{c}: R^n \rightarrow V$
  - $h(\underline{x}): R^n \rightarrow V$

Distancias

- Atributos continuos: Euclídea
  - Necesidad de **normalización** [0, 1] (escalado)
    - $a_i = (v_i - \min v_i) / (\max v_i - \min v_i)$
- Manhattan (atributos enteros)
  - $D(\underline{x}, \underline{y}) = \sum_{i=1}^{n} | x_i - y_i |$
- Atributos discretos: Hamming
  - $D(\underline{x}, \underline{y}) = \Sigma_{i=1,\,k} d_i$ con $d_i = 0$ si $x_i = y_i$, 1 en otro caso
- Atributos desconocidos
  - Distancia máxima (considerando normalización: 1 si faltan ambos)

Necesidad de normalización

- Suponer instancias descritas por dos atributos continuos (a_1, a_2), sin normalizar
- Dominios:
  - a_1 : [0, 1]
  - a_2 : [0, 1000]

- Máxima distancia eje a_1:
  - $x_1=(0, 0)$, $x_2=(1,0)$, $d(x_1, x_2)=1$
- Máxima distancia eje a_2:
  - $x_1=(0, 0)$, $x_3=(0,1000)$, $d(x_1, x_3)=1000$

- La distancia entre dos puntos está dominada por la diferencia de valores de a_2

Normalizando…

- Normalización (escalado)

  $\mathbf{a_i = (v_i - \min v_i) / (\max v_i - \min v_i)}$

  - a_i: valor atributo i-ésimo después de normalizar
  - v_i: valor atributo i-ésimo sin normalizar
- Dominios:
  - a_1 : [0, 1]
  - a_2 : [0, 1]
- Máxima distancia eje a_1:
  - $x_1=(0, 0)$, $x_2=(1,0)$, $d(x_1, x_2)=1$
- Máxima distancia eje a_2:
  - $x_1=(0, 0)$, $x_3=(0, 1)$, $d(x_1, x_3)=1$
- Tras la normalización, todos los atributos tienen la misma influencia en la distancia

Propiedades K-NN

- Simple y eficiente.
- Entrenamiento muy rápido.
- Aprende conceptos complejos.

Pero:

- Lento en clasificación: necesidad de encontrar vecinos más próximos.
- Sensible al ruido para valores pequeños de *k.*
- Dificultades si Atributos irrelevantes: asume que todos los atributos tienen la misma importancia.

3. Mejoras al algoritmo básico (K-NN)

- Lento en clasificación:
  - **Búsqueda eficiente de vecinos más próximos.**
  - Eliminar ejemplos irrelevantes, almacenar prototipos
- Ruido:
  - **Seleccionar K mediante validación cruzada.**
  - **Eliminar ejemplares ruidosos.**
- Atributos irrelevantes:
  - Selección de atributos.
  - **Distancia Euclídea ponderada.**

Búsqueda eficiente de vecinos más próximos

- Estructuras de datos que:
  - Disminuyan el tiempo de acceso.
  - Permitan actualización incremental.
- Posibilidad:
  - Indexar instancias con KD-Trees (< 10 atributos)
  - Ball Trees en espacios multidimensionales.

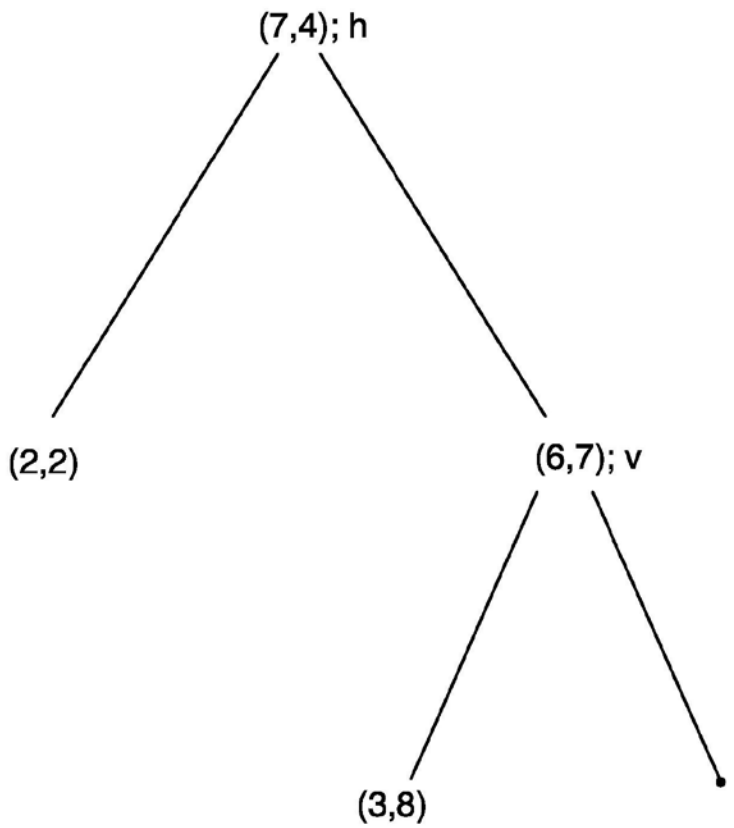
kD-Trees

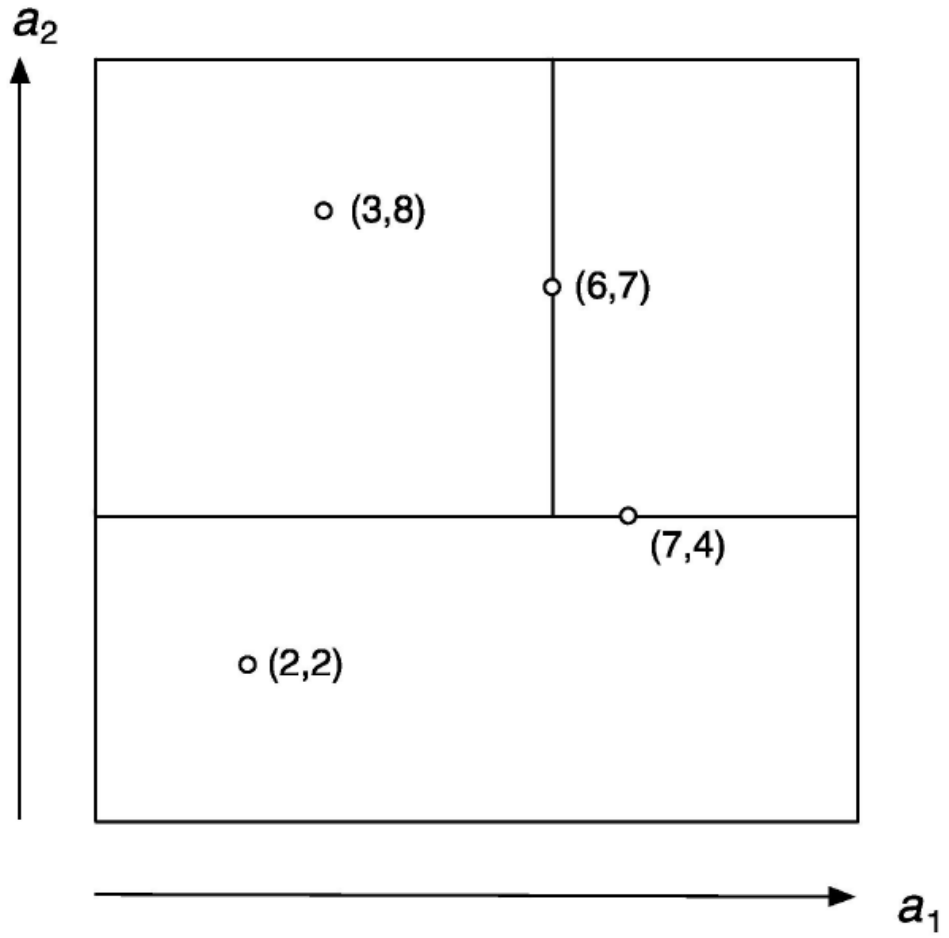
- kD-Trees: k dimensional trees (k: número de atributos)

- Estructura de datos que representa una partición de un espacio k-dimensional.
  - Útil para organizar puntos en el espacio.
  - Acelerar el acceso a vecinos más próximos.

- Árbol binario.
- Cada nodo interno divide el espacio de entrada con un hiperplano.
  - Paralelo a un eje.
- Un nodo por punto (quizás más).

kD-Trees





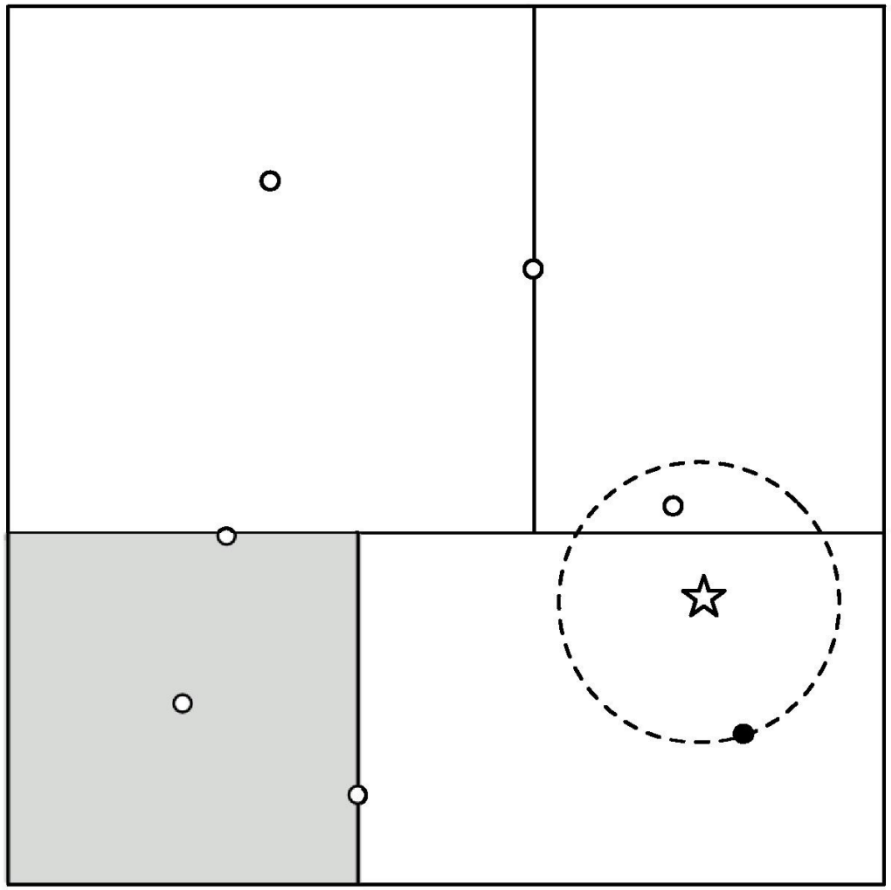
Construcción kD- Trees

- Recursivamente, seleccionando dirección división y punto.
  - Dirección división: eje con mayor varianza.
  - Punto división: mediana en esa dirección.
    - Heurísticas adicionales si ejes paralelos (rectángulos delgados)
- Mantener carácter incremental:
  - Actualizar árbol con nuevos puntos
    - Determinar su hoja; si vacía añadir ejemplo, sino nueva división.
- Mantener el árbol equilibrado:
  - Heurística: reconstruir el árbol de vez en cuando.

Búsqueda vecinos en kD-Trees

- Recorrer árbol y localizar la región que contiene la instancia a clasificar
- Primera aproximación: punto región localizada
  - No necesariamente el más próximo, pero el vecino más próximo ha de estar más cerca de la instancia a clasificar
- Comprobar si punto más cercano en región vecina (nodo padre y hermano)
- Comprobar hermanos del nodo padre y sucesores

Búsqueda vecinos kD-Trees



- En un árbol equilibrado:
  - hoja $\log_2 n$
  - vecinos $\log_2 n$ (regiones rectangulares, no muchos atributos)

Limitaciones kD-trees

- Menor eficiencia al aumentar la dimensión del espacio
  - No se recomiendan si k>10
- Problemas con las esquinas
  - Obligan a buscar en regiones potencialmente lejanas

- Alternativas: Ball Trees

Ruido

- *1*-NN muy sensible al ruido.
- *k*-NN con voto por mayoría muy robusto para valores grandes de *k.*
  - Pero *k* elevado disminuye precisión.
  - Selección de *k*: validación cruzada.
- Otras posibilidades:
  - Mayoría ponderada con menor peso a los vecinos más distantes.
  - Podar ejemplos ruidosos.

IB3: poda de ejemplos ruidosos

- Monitoriza el comportamiento de las instancias de entrenamiento.
- Calcular intervalos de confianza para:
  - Tasa de acierto de cada ejemplo (cuando es el más próximo).
  - Tasa de acierto, por defecto, de cada clase.
    - Por defecto: sin conocer los atributos del ejemplo, calculada a partir de la distribución de clases en conjunto entrenamiento
- Aceptar/rechazar instancias:
  - Aceptar si límite inferior de 1 > límite superior de 2
  - Rechazar si límite superior de 1 < límite inferior de 2
  - En otro caso: solo monitorizar el comportamiento del ejemplo, pero no utilizar para clasificación.

Atributos irrelevantes
Maldición de la dimensionalidad

- Suponer:
  - Las instancias se describen con 20 atributos.
  - El concepto objetivo depende de 2 atributos.
  - Ejemplos con dos valores idénticos de los atributos relevantes pueden estar muy distantes.
- Soluciones:
  - Selección de atributos; usar solo atributos seleccionados previamente con otras técnicas.
  - Ponderar la importancia de cada atributo en la medida de distancia.

Distancia euclídea ponderada

$$[w_1^2(x_1-y_1)^2 + w_2^2(x_2-y_2)^2 + \ldots + w_n^2(x_n-y_n)^2]^{1/2}$$

- Ajuste de pesos: heurística.
  - Actualizar pesos tras clasificar cada instancia de entrenamiento.
  - $\underline{x}$, ejemplo a clasificar, $\underline{y}$ vecino más próximo.
  - Clasificación correcta: aumentar pesos.
  - Clasificación incorrecta: disminuir pesos.
  - Cuantía del cambio w_i: inversamente proporcional a $|x_i - y_i|$, considerando clasificación.
  - Renormalizar pesos tras actualización.

Discusión K-NN

- Simple y efectivo.
  - Entrenamiento muy rápido.
  - Aprende conceptos complejos.
- Lento en clasificación:
  - Estructuras de datos mejoran acceso.
  - Solución: no almacenar todos los ejemplos, poda, prototipos.
- Sensible al ruido:
  - Solución: K>1, validación cruzada; poda
- Problemas con dimensionalidad:
  - Solución: selección de atributos, distancia ponderada.

- Con distancia ponderada y poda de ejemplos, tasas de error comparables a otros métodos.

Implementaciones en Weka

- Agrupados bajo pestaña *lazy*

- weka.classifiers.lazy.IB1 : 1-NN con distancia euclídea normalizada.
- weka.classifiers.lazy.IBk: K-NN con opciones para la distancia, ponderación, búsqueda y selección de K.
  - No confundir con algoritmos IB2 e IB3

Bibliografía

- Ian H. Witten, Eibe Frank and Mark A. Hall. Data Mining: practical machine learning tools and techniques (third Edition). Morgan Kaufmann, 2011.