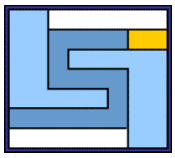
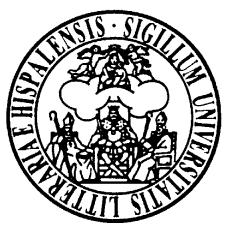
TÉCNICAS DE REDUCCIÓN EN BASES DE DATOS

Francisco J. Ferrer
Jesús Aguilar
Joaquín Peña

Departamento de Lenguajes y Sistemas Informáticos
Facultad de Informática
Universidad de Sevilla

INFORME TÉCNICO: LSI-2000-09
Julio, 2000





INDICE GENERAL

LISTADO DE FIGURAS

1. El vecino más cercano.

El paradigma del vecino más cercano ha sido objeto de numerosos estudios en esta última década. Su criterio de aprendizaje se basa en que los miembros de una población suelen convivir (y al limite, coexistir) rodeados de individuos similares, con propiedades parecidas. Bajo esta hipótesis toda información descriptiva (adicional o desconocida) que quiera extraerse de un individuo puede observarse en otras instancias cercanas, en sus vecinos más cercanos. Consecuencia directa es la aplicación de este método como técnica de clasificación: del valor de la etiqueta de los miembros de una población se induce el valor desconocido de la etiqueta de un nuevo individuo.

Los conceptos básicos que sostienen a los clasificadores por vecindad fueron presentados por [1] en su estudio del problema de la discriminación sin parámetros, donde se describían varios procedimientos cuyos resultados en grandes poblaciones se acercaban al óptimo asintótico. Sin embargo, hasta 1967 ([2], [3], [4]) no se define la regla del vecino más cercano para su aplicación en la clasificación y reconocimiento de patrones.

Conocidos también como algoritmos de aprendizaje basados en instancias (***Instance-Based Learning Algorithms, IBL Algorithms*** o basados en ejemplos [5]), se engloban dentro del ***Lazy-Learning*** [6] al aplazar el proceso de inducción o generalización hasta que se completa la clasificación.

En el ***Lazy-Larning***, rama del Aprendizaje Automático (Inteligencia Artificial), se implementa un algoritmo que se ejercita para terminar aprendiendo a realizar correctamente una determinada tarea, entrenándose previamente con aquellas situaciones o casos posibles con los que puede encontrarse, de los cuales, selecciona los más representativos del concepto sobre el cual se quiere extraer conocimiento (**Aprendizaje Supervisado**).

Los elementos que forman el conjunto de entrada se denominan ejemplos o instancias y se componen de un vector y una o varias etiquetas. Cada dimensión del vector representa una característica o atributo del conjunto de entrada. El dominio de cada uno de los atributos puede ser discreto o continuo. La etiqueta se denomina clase y clasifica categóricamente a cada ejemplo del conjunto de entrada.

Basándose en *similitudes* entre los datos del conjunto de entrada o *entrenamiento* T se almacena únicamente un subconjunto *editado* S con lo que se reduce considerablemente el espacio de búsqueda y se acelera el proceso de aprendizaje [7]. A partir de este nuevo subconjunto se generan hipótesis para describir, clasificar o predecir el comportamiento de nuevos ejemplos cuya clase se desconoce. Estos últimos conforman el *test* del sistema de aprendizaje.

Al contrario que otros los métodos predictivos, el conjunto de entrenamiento no es procesado para crear el modelo ya que éste es el modelo en sí mismo. Cuando un nuevo ejemplo se presenta, el algoritmo encuentra el subconjunto de ejemplos que son más similares a él y los utiliza para predecir su etiqueta.

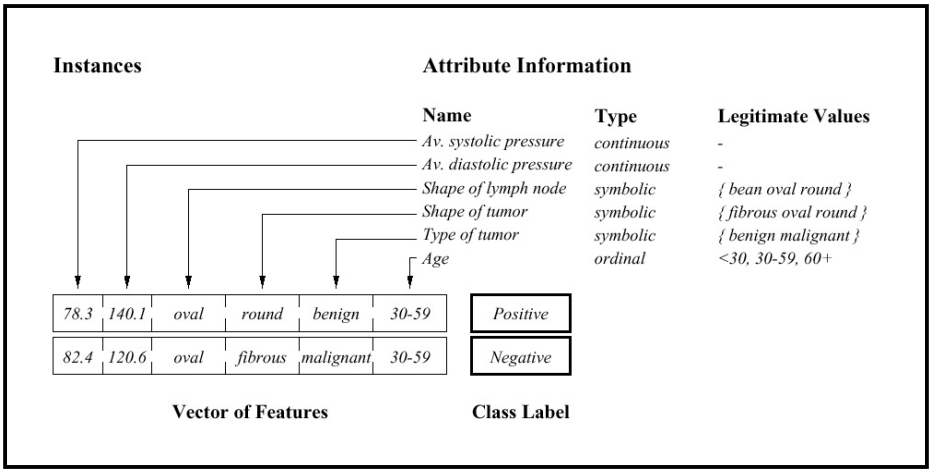


Figura 1.1. Terminología.

El poder del aprendizaje por vecindad ha sido demostrado en numerosos problemas reales, tales como la correcta pronunciación de palabras [8], diagnósticos para enfermos de tiroides [9], reconocimiento de voz [10], predicciones de negocio, simuladores de vuelo y pilotos automáticos, detectores de fraudes bancarios, expertos en juegos [6] o la tan esperada secuenciación del ADN y ARN [11].

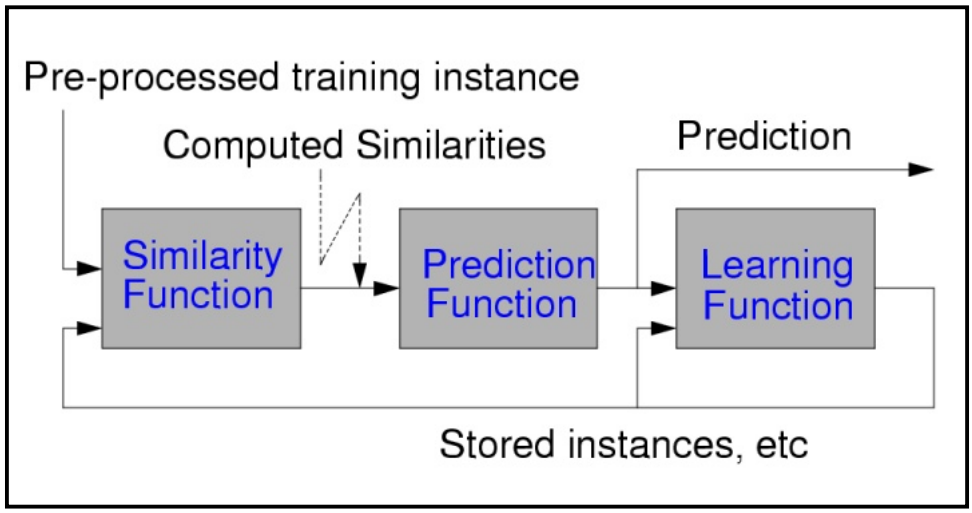


Figura 1.2. Los sistemas de aprendizaje basados en ejemplos.

A pesar de la sencillez conceptual y de la simplicidad de implantación de esta técnica, la probabilidad de error obtenida es siempre cercana a la probabilidad de error óptima de Bayes. Según Cover y Hart se demuestra que cuando el tamaño del conjunto de entrenamiento tiende a infinito, el error asintótico del vecino más cercano no supera al doble del error óptimo de Bayes.

Con todo esto, el ***Lazy-Learning*** sigue siendo muy criticado. En [12] se subrayan cinco inconvenientes fundamentales:

Ausencia de salida descriptiva.
Dependencia de una métrica arbitraria, convirtiéndola en una técnica subjetiva y dependiente del usuario experto que debe conocer el problema.
Escaso rendimiento, pues para cada nueva predicción hay que procesar todo el conjunto de entrenamiento de nuevo.
Aprendizaje complicado y limitado cuando aparecen ejemplos con valores nulos y ruido.

El presente documento pretende dar una aproximación a las principales técnicas de reducción sin analizar cuestiones de rendimiento ni comportamiento para problemas generales o específicos. El lector interesado puede acudir a las referencias dadas al final del mismo.

2. Aprendizaje basado en ejemplos.

La reducción del conjunto inicial de ejemplos forma parte de la fase inicial en los sistemas de adquisición de conocimiento. En este preprocesado de los datos, la reducción puede ser tanto del número de atributos irrelevantes [13] como del número de ejemplos irrelevantes. Diferentes criterios y parámetros clasifican a las técnicas de reducción de ejemplos según el modo en el cual su utilización trata de una forma u otra la información de entrada.

Representación.

Los sistemas basados en la técnica de los vecinos más cercanos representan el conocimiento dividiendo el espacio de búsqueda en forma de teselación de Voronoi obteniendo fronteras entre las clases poligonales [14]. A medida que tenemos más ejemplos, estas fronteras tienden a ser irregulares. El caso más representativo se produce cuando tenemos solo un ejemplo por clase en el que las fronteras son líneas.

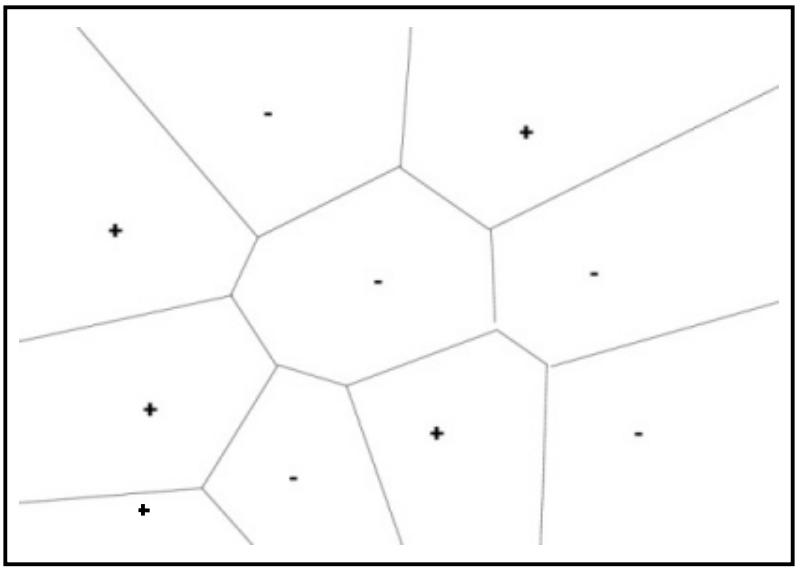


Figura 2.1. Diagrama de Voronoi en un espacio de búsqueda con dos valores posibles para la clase: {+, -}.

Una de las desventajas de utilizar como representación los propios ejemplos es que en ellos puede no existir aquel subconjunto que proporcione la mayor precisión y descripción del conjunto para el cual se quiere generalizar. Una nueva representación supone un nuevo modelo del problema. Entre las alternativas mas utilizadas para representar una colección de datos cabe destacar: *Hiperrectangulos* (en un espacio bidimensional, áreas determinadas por dos puntos del plano), *Prototipos* (un subconjunto de ejemplos se reemplaza por su centro de masa, medias, etc.) y *Reglas.*

Los *Hiperrectangulos* y las *reglas* son construidos para reducir el número de ejemplos necesarios en determinadas áreas del espacio de soluciones y acelerar la búsqueda de similitudes [15]. El coste adicional en construir la nueva representación debe recompensar la búsqueda en un espacio mas reducido.

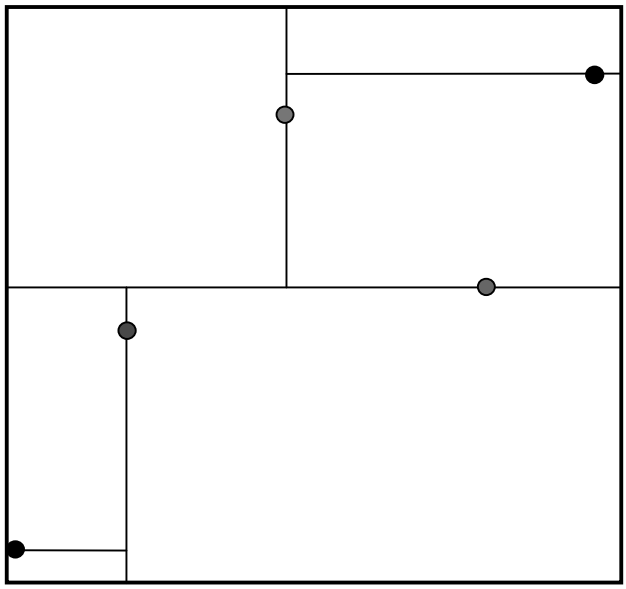


Figura 2.2. kd.tree.

Un modelo de representación muy utilizado es el kd-tree, un árbol binario que divide el espacio de búsqueda de forma recursiva en los dos subespacios (de todos los posibles) con la menor diferencia en tamaño. En la figura 2.2 se observa un ejemplo donde el punto azul es la raíz del árbol cuyos dos hijos son los puntos rojo y verde. Cada uno de ellos tiene un único hijo, hojas del árbol (puntos negros).

Dirección de búsqueda.

La búsqueda de un subconjunto S reducido, obtenido a partir de un conjunto de entrenamiento T, puede realizarse en tres direcciones: incremental, decremental y por lotes.

Incremental: se parte de un conjunto S vacío y a él se añaden los ejemplos que cumplen el criterio de clasificación.

Decremental: el conjunto S es igual que T y de S se eliminan aquellos ejemplos que no cumplen el criterio de clasificación.

Por lotes (batch): En T se marcan todos los ejemplos que serán clasificados y una vez recorrido completamente se añaden a S.

En la búsqueda incremental el orden de aparición de los elementos en el conjunto es critico debido a que la probabilidad de los primeros que se visitan de ser incluidos es mucho mayor a la de los últimos, los cuales al recorrerse pueden ya estar representados. En este sentido la precisión en la generalización puede verse dañada si los últimos ejemplos representan con mayor fidelidad al conjunto que los primeros.

De este modo es necesario que el orden de presentación sea totalmente aleatorio ya que por definición un algoritmo incremental debe poder gestionar nuevas instancias del conjunto sin necesidad de que estas estén todas al principio, lo cual se convierte en una ventaja a su favor.

Otra ventaja es que pueden ser más rápidos y necesitar menos requisitos de almacenamiento que los demás durante la fase de aprendizaje. Esto es debido a que operan sobre las instancias incluidas (O(S)) frente aquellos que necesitan todo el conjunto (O(N), S < N).

Pero además de la sensibilidad al orden de presentación la mayor de las desventajas de estos algoritmos es que sus tempranas decisiones se basan en muy poca información. En la búsqueda decremental el orden de presentación es también importante pero a diferencia de los algoritmos incrementales todos los ejemplos deben estar disponibles en todo momento.

Una desventaja es el tiempo, no es lo mismo incluir un elemento en el conjunto de salida a partir de los existentes en el propio conjunto de salida, inicialmente vacío y por tanto con un recorrido menor que tener en cuenta todos los elementos del conjunto de entrenamiento de entrada. Como ventaja los algoritmos decrementales obtienen una mayor reducción y precisión en la generalización.

Criterios de votación.

Otra decisión que deben tomar muchos algoritmos de reducción es el valor que debe tomar *k*, el número de vecinos a tener en cuenta para clasificar o no un vector de entrada. El valor de *k* suele ser generalmente pequeño e impar y de tal forma que la influencia de los *k* vecinos suele ser la misma. Existen algoritmos que dan una influencia mayor a aquellos vecinos más cercanos. Estos algoritmos se dicen que establecen criterios de votación.

Ejemplos Centrales frente a Ejemplos Extremos.

A la hora de eliminar aquellos ejemplos irrelevantes, el factor principal que distingue a las técnicas de reducción de instancias es si buscan retener a ejemplos extremos o cercanos a las fronteras, ejemplos internos o centrales, o bien a cualquier otro conjunto de ejemplos.

Bajo la intuición de retener ejemplos cercanos a los bordes, está el hecho de que los ejemplos internos no afectan tanto a los limites de decisión, por lo que estos últimos pueden ser descartados sin que ello implique apenas efecto en la posterior clasificación.

Por otro lado eliminar ejemplos extremos obedece al hecho de que los ejemplos extremos pueden suponer ruido (errores de tipeado, etc.) o excepciones que provocan una mala representación del espacio de búsqueda. De esta forma se refinan los limites de decisión.

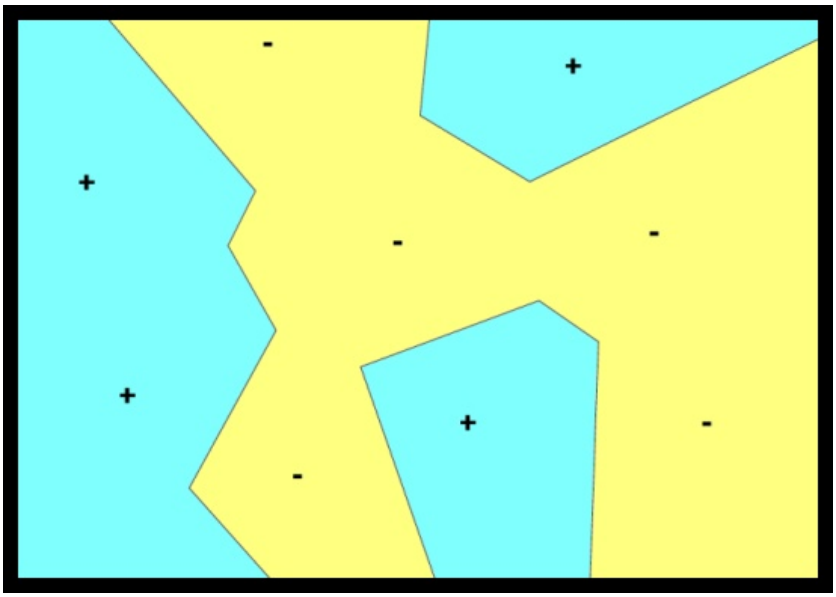


Figura 2.3. Delimitación de las fronteras de decisión en un espacio de búsqueda.

Función Similitud.

La función similitud se utiliza para medir la proximidad entre los individuos de una población, esto es, decidir qué vecinos son más cercanos a un vector de entrada. Dado que es el criterio base en las decisiones tomadas por las técnicas de aprendizaje basadas en ejemplos supone un factor critico en dicho proceso. La elección de una métrica determinada depende de la topología del espacio de búsqueda y debe someterse al usuario experto en el problema.

En [16] se proponen distintas métricas, distinguiendo entre atributos continuos y discretos (véase figura 5).

La mayoría de las métricas empleadas para atributos continuos derivan de la función distancia propuesta por Minkowsky [17], por lo que heredan las propiedades de la definición geométrica:

$D(x, y) = D(y, x)$	Simétrica.
$D(x, y) + D(y, z) \geq D(x, z)$	Triangular.
$D(x, y) \geq 0 \mid D(x, y) = 0 \Leftrightarrow x = y$	Positiva.

Según [18] varios estudios psicológicos han argumentado que la distancia Manhattan es apropiada en dominios con dimensiones separables (ortogonales), aunque los experimentos realizados por Salzberg [19] no corroboraron tal hipótesis.

Minkowsky:
$$D(\boldsymbol{x},\boldsymbol{y}) = \left(\sum_{i=1}^{m} |x_i - y_i|^r\right)^{1/r}$$

Euclidean:
$$D(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

Manhattan / city-block:
$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} |x_i - y_i|$$

Camberra:
$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{|x_i - y_i|}{|x_i + y_i|}$$

Chebychev:
$$D(\boldsymbol{x},\boldsymbol{y}) = \max_{i=1}^{m} |x_i - y_i|$$

Quadratic:
$$D(\boldsymbol{x},\boldsymbol{y}) = (\boldsymbol{x}-\boldsymbol{y})^T Q(\boldsymbol{x}-\boldsymbol{y}) = \sum_{j=1}^{m}\left(\sum_{i=1}^{m}(x_i - y_i)q_{ji}\right)(x_j - y_j)$$
Q is a problem-specific positive definite $m \times m$ weight matrix

Mahalanobis:
$$D(\boldsymbol{x},\boldsymbol{y}) = [\det V]^{1/m}(\boldsymbol{x}-\boldsymbol{y})^{\mathrm{T}} V^{-1}(\boldsymbol{x}-\boldsymbol{y})$$
V is the covariance matrix of $A_1..A_m$, and A_j is the vector of values for attribute j occuring in the training set instances $1..n$.

Correlation:
$$D(\mathbf{x},\mathbf{y}) = \frac{\sum_{i=1}^{m}(x_i - \mu_i)(y_i - \mu_i)}{\sqrt{\sum_{i=1}^{m}(x_i - \mu_i)^2 \sum_{i=1}^{m}(y_i - \mu_i)^2}}$$
μ_i is the average value for attribute i occuring in the training set.

Chi-square:
$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{1}{sum_i}\left(\frac{x_i}{size_{\boldsymbol{x}}} - \frac{y_i}{size_{\boldsymbol{y}}}\right)^2$$
sum_i is the sum of all values for attribute i occuring in the training set, and $size_x$ is the sum of all values in the vector $\boldsymbol{x}$.

Kendall's Rank Correlation:
$$D(\boldsymbol{x},\boldsymbol{y}) = 1 - \frac{2}{n(n-1)}\sum_{i=1}^{m}\sum_{j=1}^{i-1}\mathrm{sign}(x_i - x_j)\mathrm{sign}(y_i - y_j)$$
sign(x)=-1, 0 or 1 if $x < 0$, $x = 0$, or $x > 0$, respectively.

Figura 2.4. Definiciones de las funciones distancias más usuales para atributos continuos. X e Y son ejemplos y M el número de atributos de la base de datos.

El paradigma del vecino más cercano puede verse como una función que mapea un espacio vectorial de M dimensiones en un espacio de salida de 1 dimensión:

> Una instancia y su etiqueta se representan mediante la tupla (x_i, c_i) donde x_i representa la localización de la instancia i en el espacio de instancias y c_i la localización de su etiqueta en el espacio de etiquetas, esto es: *xi pertenece a la clase ci*.

Tras la idea de asociar la distribución de valores del espacio de instancias con la distribución de valores del espacio de etiquetas se establece que la probabilidad condicional de que un ejemplo x_q sea de la clase c_q tiende a ser la misma que la de otro ejemplo x_i respecto a la clase c_i cuando la distancia entre ambos tiende a cero, esto es:

$$\forall (x_q, c_q), (x_i, c_i) \in S \cdot P(c_q \mid x_q) \approx P(c_i \mid x_i) \leftrightarrow D(x_q, x_i) \approx 0;$$

La familia Minkowsky de métricas continuas determinan el valor $D(x_a, y_a)$ para cada atributo continuo a del espacio de ejemplos y calculan luego el valor total, por lo que cuando el rango de valores es distinto en más de un atributo, el total obtenido resulta engañoso.

De esta forma, con una reducción del conjunto *en bruto*, un rango muy amplio tendrá un gran impacto en la distancia total frente a rangos reducidos, cuyo efecto será inapreciable. Un clásico ejemplo es el peso medio (entre 50 y 150 kilogramos) y la altura media (entre 1.5 y 2 metros) de una población de individuos con edades comprendidas entre los 18 y 50 años. Para evitar esta situación los valores de todos los atributos son normalizados en un rango dado por el usuario experto. Los métodos de normalización más usuales son:

Normalización Lineal: los valores se dividen por el rango del atributo al que pertenecen, por lo que quedan comprendidos en el intervalo [0,1]:
Normalización Standard: los valores se dividen por la desviación típica del atributo al que pertenecen.

La distancia usual utilizada con atributos discretos es la métrica *Overlap* (solapar) que es una variante simbólica de la métrica Manhattan, definida como:

$$D(x, y) = \begin{cases} 0 \Leftrightarrow x = y \\ 1 \Leftrightarrow x \neq y \end{cases}$$

Stanfill & Waltz [8] proponen la métrica de la diferencia de valores (VDM, *Value Difference Metric*) que a pesar de no cumplir la definición de distancia (no es simétrica) ha recibido una enorme aceptación por parte de la comunidad científica ([20], [16]).

$$\begin{aligned} vdm(i,j) &= \sum_{a=0}^{A} \delta(i_a, j_a) \cdot \omega(i_a) \\ \delta(i_a, j_a) &= \sum_{c \in C} \left| \mathrm{P}(c|i_a) - \mathrm{P}(c|j_a) \right|^2 \\ \omega(i_a) &= \left[\sum_{c \in C} \mathrm{P}(c|i_a)^2 \right]^{0.5} \end{aligned}$$

Figura 2.5. Distancia VDM.

A diferencia de otras métricas VSM no determina la distancia entre dos valores comparándolos directamente sino a través de la distribución de probabilidad condicional. La ecuación viene dada en la figura 2.5 donde:

i,j son dos instancias del conjunto de datos.
a es uno de los A atributos.
C es el conjunto de todas las etiquetas que aparecen en la clase para los ejemplos del conjunto de datos.
$P(c|i_a)$ es la probabilidad de que el valor i_a aparezca en el conjunto de datos para el atributo *a* en instancias cuya clase tenga la etiqueta *c*, esto es:

$$P(c|i_a) = \frac{N^o \text{ de instancias con el valor ia en el atributo a y con etiqueta c como clase}}{N^o \text{ de instancias con el valor } i_a \text{ en el atributo a}}$$

$w(i_a)$ es el peso asociado e indica como el valor de un atributo influye entre las distintas etiquetas de clasificación. El peso puede variar entre un máximo 1, que representa al discriminante ideal (es decir, el valor de un atributo solo aparece para una clase) y un mínimo, que representa una distribución uniforme de las etiquetas de clasificación en la que un valor de un atributo aparece con igual probabilidad para todas las etiquetas del conjunto. La ecuación del peso mínimo es:

$$w(u) = |C|^{-0.5}$$

El peso controla la influencia de la distancia para un atributo en el cálculo de la distancia total entre dos ejemplos. Un atributo con un rango elevado tendrá mayor efecto que otro con un rango reducido. Usando pesos pequeños este problema desaparece y la distancia resultante, $\delta(i_a, j_a) \times w(i_a)$, también pequeña por lo que el impacto es mucho menor.

Si el rango de valores de $\delta(i_a, j_a)$ es [0,1], el peso puede utilizarse para restringir dicho rango, de forma que el rango de $\delta(i_a, j_a) \times w(i_a)$ será [0, $w(i_a)$].

3.1. CNN: Condensed Nearest Neighbour.

Con el algoritmo del vecino más cercano condensado (también conocido como 1NN, 1 Nearest Neighbour), Hart [3] introduce el concepto de subconjunto consistente mínimo como aquel subconjunto S del conjunto inicial T que clasifica correctamente a todos los ejemplos de T. Sin embargo CNN no encuentra el subconjunto mínimo.

CNN encuentra el subconjunto S donde cada ejemplo de T es más cercano a un ejemplo de S de su misma clase que a un ejemplo de S de distinta clase. CNN termina cuando todos los ejemplos de T que cumplen el criterio de clasificación están en S.

CNN (T, d)

Entrada:
S: $R^M \times N$
d: $(R^M \times R^M) \rightarrow R^+_0$
$T = \{e_1, ..., e_N\} / \forall i \in [1, N] \cdot e_i = (x_i, c_i), x_i \in R^M, c_i \in [1, C]$

Salida: S

Pre: $\forall e_i = (x_i, c_i), e_j = (x_j, c_j) \in T / c_i = c_j \Rightarrow x_i \neq x_j$

Post: $S \subseteq T$

```
INICIO

    S ← ∅
    Añadir a S un ejemplo de T de cada clase seleccionados al azar

    Repetir
        adiciones ← FALSO
        Para cada i ∈ [1, |T|]
            Elegir aleatoriamente e_i = (x_i, c_i) ∈ T no usado
            Marcar e_i como usado
            e_j ← NN(e_i, S, d)
            Si c_j ≠ c_i →
                S ← S ∪ {e_i}
                adiciones ← CIERTO
    Hasta NO adiciones

FIN
```

Figura 3.1.1. Algoritmo CNN.

Para ello comienza seleccionando de forma aleatoria un ejemplo de cada clase y los introduce en S. Los ejemplos de T serán clasificados por aquel ejemplo de S más cercano con distinta clase, esto es, se seleccionan aquellos ejemplos de T que son clasificados erróneamente por ejemplos de S. Es, por tanto, incremental y busca retener a los ejemplos extremos.

CNN es especialmente sensible al ruido, ya que ejemplos ruidosos serán normalmente mal clasificados por sus vecinos y por lo tanto, se retienen. Es evidente que existen problemas, son dos:

Complica la reducción óptima del espacio de almacenamiento ya que a menudo se encuentran cerca ejemplos no ruidosos que necesitaran ser retenidos.

La posterior clasificación será más difícil debido a que el ruido dará lugar a un sobreajuste (overfiting) con lo que los ejemplos a clasificar no se ajustan a la función subyacente. Puesto que algunos vecinos han sido probablemente eliminados un ejemplo ruidoso en S cubrirá mas ejemplos de los que ya hacia en T, provocando mas clasificaciones incorrectas que antes de la reducción.

3.2. IB2: Instaces Based 2.

En [16] se proponen distintas heurísticas derivadas de la técnica CNN como alternativas a los costes en tiempo y memoria. Entre ellas aparece IB2, incremental y análogo a CNN, solo que S inicialmente contiene un único ejemplo elegido al azar y recorre el conjunto de entrada T una sola vez. Esta alternativa intenta ser una mejora en tiempo respecto a 1NN.

IB2 (T, S, d)

Entrada:
S: $R^M \times N$
d: $(R^M \times R^M) \rightarrow R^+_0$
$T = \{e_1, ..., e_N\} / \forall i \in [1, N] \cdot e_i = (x_i, c_i), x_i \in R^M, c_i \in [1, C]$

Salida: S

Pre: $\forall e_i = (x_i, c_i), e_j = (x_j, c_j) \in T / c_i = c_j \Rightarrow x_i \neq x_j$

Post: $S \subseteq T$

INICIO

$S \leftarrow \varnothing$
Para cada $i \in [1, |T|]$
Elegir aleatoriamente $e_i = (x_i, c_i) \in T$ *no usado*
Marcar e_i *como usado*

$e_j \leftarrow NN(e_i, S, d)$
Si $c_j \neq c_i \rightarrow$
$S \leftarrow S \cup \{e_i\}$

FIN

Figura 3.2.1. Algoritmo IB2.

3.3. RNN: Reduced Nearest Neighbour.

El algoritmo del vecino más cercano reducido [21] parte con el subconjunto S=T y elimina aquellos ejemplos de S que no influyen en la clasificación de los restantes, esto es, si su eliminación no causa que cualquier otro ejemplo de T sea clasificado incorrectamente por los ejemplos restantes de S. Es decremental y busca retener a los ejemplos internos.

RNN (T, S, d)

Entrada:
S: $R^M \times N$
d: $(R^M \times R^M) \rightarrow R^+_0$
$T = \{e_1, ..., e_N\} / \forall i \in [1, N] \cdot e_i = (x_i, c_i), x_i \in R^M, c_i \in [1, C]$

Salida: S

Pre: $\forall e_i = (x_i, c_i), e_j = (x_j, c_j) \in T / c_i = c_j \Rightarrow x_i \neq x_j$

Post: $S \subseteq T$

```
INICIO

    S ← T
    Para cada i ∈ [1, |T|]
        Elegir aleatoriamente e_i = (x_i, c_i) ∈ T no usado
        Marcar e_i como usado
        W ← {e_j ∈ T | NN(e_j, S, d) = e_i}
            eliminable ← CIERTO
            Para cada e_j ∈ W
                Si c_j = c_i →
                    c_k ← NN(e_j, {S - {e_i}}, d)
                    Si c_j = c_k →
                        eliminable ← FALSO
                    Si eliminable
                        S ← S - {e_i}

FIN
```

Figura 3.3.1. Algoritmo RNN.

Aunque computacionalmente es más costoso que CNN genera siempre un subconjunto del resultado proporcionado con CNN. El tiempo empleado en la reducción se recupera en tiempo y memoria en la fase de clasificación. Puesto que no se garantiza que los ejemplos que se eliminan se clasifiquen correctamente, RNN es capaz de eliminar ruido y ejemplos internos mientras conserva los casos cercanos a las fronteras.

3.4. SHRINK: Substrative Algorithm.

Derivado directo de RNN, podría decirse que implementa el proceso opuesto. Partiendo con el subconjunto S=T, elimina aquellos ejemplos de S que son clasificados correctamente con el resto de los ejemplos. La diferencia con RNN es que considera aquellos ejemplos que podrían ser clasificados correctamente, mientras que RNN considera si la clasificación de otros ejemplos podría afectar al ejemplo eliminado. Es muy sensible al ruido. Ambos son incremental y buscan retener a los ejemplos internos.

SHRINK (T, S, d)

Entrada:
S: $R^M \times N$
d: $(R^M \times R^M) \rightarrow R^+_0$
$T = \{e_1, ..., e_N\} / \forall i \in [1, N] \cdot e_i = (x_i, c_i), x_i \in R^M, c_i \in [1, C]$

Salida: S

Pre: $\forall e_i = (x_i, c_i), e_j = (x_j, c_j) \in T / c_i = c_j \Rightarrow x_i \neq x_j$

Post: $S \subseteq T$

INICIO

S ← *T*
Para cada $i \in [1, |T|]$
Elegir aleatoriamente $e_i = (x_i, c_i) \in T$ *no usado*
Marcar e_i *como usado*
$e_j \leftarrow NN(e_i, S, d)$
Si $c_j = c_i \rightarrow$
$S \leftarrow S - \{e_i\}$

FIN

Figura 3.4.1. Algoritmo SHRINK.

3.5. ENN: Edited Nearest Neighbour.

El algoritmo del vecino más cercano editado [22] parte con el subconjunto S=T y elimina aquellos ejemplos de S cuya clase no coincide con la clase mayoritaria de sus *k* vecinos más cercanos. El valor de k es fijo y Wilson propone un valor pequeño e impar, por defecto 3.

RENN (Repeated Edited Nearest Neighbour) aplica el algoritmo ENN hasta que todos los ejemplos de S tienen la misma clase que la clase mayoritaria de sus *k* vecinos. La principal característica de ambos es la insensibilidad al ruido y a aquellos ejemplos cercanos a los limites de decisión. Ambos realizan una búsqueda incremental en la que retienen los ejemplos internos.

RENN (T, S, d)

Entrada:
S: $R^M \times N$
d: $(R^M \times R^M) \rightarrow R^+_0$
$T = \{e_1, ..., e_N\} / \forall i \in [1, N] \cdot e_i = (x_i, c_i), x_i \in R^M, c_i \in [1, C]$

Salida: S

Pre: $\forall e_i = (x_i, c_i), e_j = (x_j, c_j) \in T / c_i = c_j \Rightarrow x_i \neq x_j$

Post: $S \subseteq T$

```
INICIO

    S ← T
    Repetir
        eliminado ← FALSO
        Para cada i ∈ [1, |S|]
            Elegir aleatoriamente e_i = (x_i, c_i) ∈ S no usado
            Marcar e_i como usado

            W ← kNN(e_i, S, d, k)
            Si ClaseMayoritaria(W) ≠ c_i →
                S ← S - {e_i}
                eliminado ← CIERTO
      Hasta NO eliminado

FIN
```

Figura 3.5.1. Algoritmo RENN.

3.6. AENN: All k Edited Nearest Neighbour.

Tomek proporciona dos extensiones a su algoritmo ENN. AENN señala como *erróneo* aquellos ejemplos no señalados de S cuya clase no coincide con la clase mayoritaria de sus *k* vecinos más cercanos, para cada valor de *k*.

AENN no es mas que la aplicación de ENN para cada *k*=1, 2,..., *K*. Una vez terminado el bucle elimina aquellos ejemplos *erróneos*. Las pruebas demuestran que All k-NN [23] obtiene mayor precisión que RENN.

La figura 3.6.1. muestra como el valor de k afecta significativamente el rendimiento de un algoritmo para aquellos ejemplos cercanos a los limites de decisión: para k = 9 x_q es + mientras que para k = 3 es –.

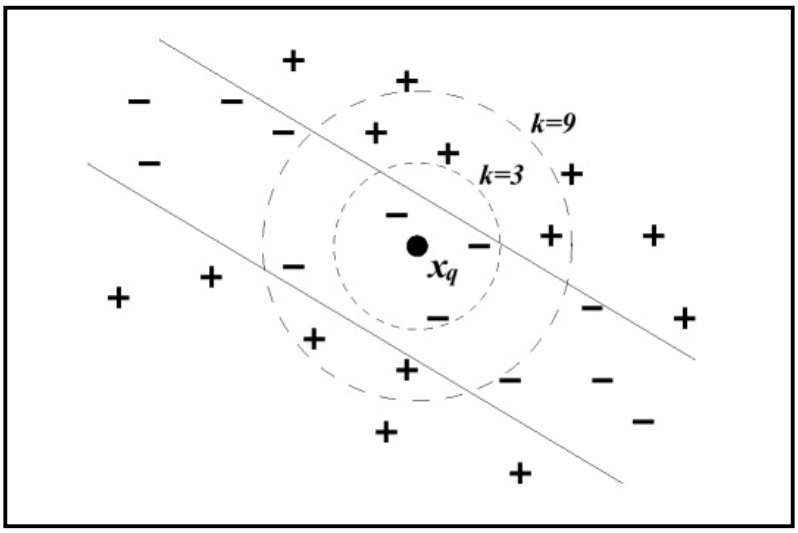


Figura 3.6.1. Algoritmo All k-NN.

Cuando k→n y la distribución no es uniforme en cuanto a las clases, el algoritmo de aprendizaje se comporta como un aproximador global, es decir, la precisión obtenida depende exclusivamente de la distribución.

Según los experimentos de Wettschereck, kNN es mucho más robusto que 1NN en presencia de ruido para valores pequeños de k y grandes conjuntos de datos. Sin embargo para conjuntos reducidos (con menos de 100 ejemplos) y con mucha dispersión, 1NN obtenía mejores resultados.

All k-NN retiene ejemplos internos tras haberlos seleccionado una vez recorrida la base de datos k veces (*batch*).

AENN (T, S, d)

Entrada:
S: $R^M \times N$
d: $(R^M \times R^M) \rightarrow R^+_0$
$T = \{e_1, ..., e_N\} / \forall i \in [1, N] \cdot e_i = (x_i, c_i), x_i \in R^M, c_i \in [1, C]$

Salida: S

Pre: $\forall e_i = (x_i, c_i), e_j = (x_j, c_j) \in T / c_i = c_j \Rightarrow x_i \neq x_j$

Post: $S \subseteq T$

```
INICIO

    Para cada i ∈ [1, |T|]
        Flags[i] ← FALSO

    Para cada i ∈ [1, |T|]
        Elegir aleatoriamente e_i = (x_i, c_i) ∈ S no usado
        Marcar e_i como usado

        k ← 1
        Mientras (k ∈ [1, K]) Λ (NO Flags[i])
            W ← kNN(e_i, S, d, k)
            Si ClaseMayoritaria(W) ≠ c_i →
                Flags[i] ← CIERTO
            Sino
                k ← k+1

    Para cada i ∈ [1, |T|]
        Si Flags[i] →
            S ← S – {e_i}

FIN
```

Figura 3.6.2. Algoritmo All k-NN.

La segunda variante es el algoritmo de los vecinos más cercanos editados ilimitado (Unlimited Edited Nearest Neighbour, UENN) y aplica iterativamente ENN realizando recorridos adicionales sobre el conjunto editado.

3.7. VSM: Variable Simility Metric.

El algoritmo de la métrica de similitud variable genera un nivel de confianza de sus clasificaciones. Con objeto de cubrir ambos objetivos, reducción y ruido, VSM [24] parte con el subconjunto S=T y elimina aquellos ejemplos de S que tienen todos sus *k* vecinos más cercanos con la misma clase, igual o distinta a la suya, usando un k atípicamente alto, normalmente 10, y un sistema de votación que más favorece el valor de *k* cuan mayor sea éste. VSM es decremental y busca retener ejemplos cercanos a los bordes.

VSM (T, S, d)

Entrada:
S: $R^M \times N$
d: $(R^M \times R^M) \rightarrow R^+_0$
$T = \{e_1, ..., e_N\} / \forall i \in [1, N] \cdot e_i = (x_i, c_i), x_i \in R^M, c_i \in [1, C]$

Salida: S

Pre: $\forall e_i = (x_i, c_i), e_j = (x_j, c_j) \in T / c_i = c_j \Rightarrow x_i \neq x_j$

Post: $S \subseteq T$

```
INICIO

    S ← T
    Para cada i ∈ [1, |S|]
        eliminado ← CIERTO
        Elegir aleatoriamente e_i = (x_i, c_i) ∈ S no usado
        Marcar e_i como usado

        S' ← vsmNN(e_i, S, d, k)

        i ← 1
        Mientras (i ∈ [1, k-1]) Λ eliminado
            Si clase(s'_i) ≠ clase(s'_{i+1}) →
                eliminado ← FALSO

        Si eliminado →
            S ← S – {e_i}

FIN
```

Figura 3.7.1. Algoritmo VSM.

3.8. ME: MultiEdit Algortihm.

El algoritmo de multieditado es un proceso iterativo *top-down* que descarta a los ejemplos mal clasificados del subconjunto editado, pero invoca a una forma de validación cruzada que realiza el editado. Los datos son aleatoriamente separados en una lista ordenada en P subconjuntos. Cada uno de subconjuntos es clasificado utilizado los ejemplos del siguiente subconjunto partición mediante la técnica 1NN. Los ejemplos que son mal clasificados son descartados. Los restantes son almacenados para volver a realizar el proceso. Si tras un determinado número de iteraciones no se ha conseguido el editado, el bucle se detiene y el algoritmo devuelve los ejemplos que queden.

ME (T, S, d, p)

Entrada:
S: $R^M \times N$; $p \in N$
d: $(R^M \times R^M) \rightarrow R^+_0$
$T = \{e_1, ..., e_N\} / \forall i \in [1, N] \cdot e_i = (x_i, c_i), x_i \in R^M, c_i \in [1, C]$

Salida: S

Pre: $\forall e_i = (x_i, c_i), e_j = (x_j, c_j) \in T / c_i = c_j \Rightarrow x_i \neq x_j$

Post: $S \subseteq T$

```
INICIO

    Dividir aleatoriamente T en p subconjuntos Ti disjuntos tal que T = ∪ Ti

    Repetir
        eliminado ← FALSO
        Para cada i ∈ [1, p]
            Para cada j ∈ [1, Ti]
                Elegir aleatoriamente ej = (xj, cj) ∈ S no usado
                Marcar ej como usado

                ek ← NN(ej, T(i%p)+1, d)
                Si ck ≠ cj →
                    Ti ← Ti – {ej}
                    eliminado ← CIERTO
    Hasta NO eliminado

FIN
```

Figura 3.8.1. Algoritmo ME.

3.9. IB3: Instaces Based 3.

Presentado por [4], es otro método incremental que mejora el rendimiento de IB2 respecto al ruido, reteniendo únicamente *los ejemplos clasificados incorrectamente que sean aceptables*:

IB3 (T, S, d)

Entrada:
$S = \varnothing$
$d: (R^M \times R^M) \rightarrow R^+_0$
$T = \{e_1, ..., e_N\} / \forall i \in [1, N] \cdot e_i = (x_i, c_i), x_i \in R^M, c_i \in [1, C]$

Salida: S

Pre: $\forall e_i = (x_i, c_i), e_j = (x_j, c_j) \in T / c_i = c_j \Rightarrow x_i \neq x_j$

Post: $S \subseteq T$

```
INICIO

    S ← ∅
    Para cada i ∈ [1, |T|]
        Elegir aleatoriamente e_j = (x_j, c_j) ∈ S no usado
        Marcar e_j como usado

        e_j ← ANN(e_i, S, d)
        Si c_i ≠ c_j →
            S ← S ∪ {e_i}
            Para cada e_k ∈ S
            Si d(e_k, e_i) ≤ d(e_j, e_i) →
                Actualizar(e_k)
                Si SigPobre(e_k) →
                    S ← S – {e_k}

    Para cada e_i ∈ S
        Si NoAceptable(e_i) →
            S ← S - {e_i}

FIN
```

Figura 3.9.1. Algoritmo IB3.

Definición.

Un ejemplo es aceptable si el limite inferior de su precisión es significativamente mayor, desde el punto de vista estadístico, al limite superior de la frecuencia de su clase, tomando un nivel de confianza del 90%.

Análogamente un ejemplo será eliminado de S si el limite superior de su precisión es significativamente inferior, estadísticamente, al limite inferior de la frecuencia de su clase, a un nivel de confianza del 70%.

Los ejemplos se mantienen en S durante el aprendizaje y se eliminan al final si no demuestran ser aceptables. La figura 3.9.2 muestra la ecuación para el test de significancia, donde para la precisión de un ejemplo en S, *n* es el número de intentos de clasificación desde la introducción del ejemplo en S.

Es decir, el número de veces que fue al menos tan cercano a t como lo fue a; *p* es la precisión de tales intentos, es decir, el número de veces que la clase satisface la clase de t, dividido por *n*; *z* es la confianza (0.9 para la aceptación y 0.7 para la eliminación).

$$\frac{p + z^2/2n \pm z\sqrt{\frac{p(1-p)}{n} + \frac{z^2}{4n^2}}}{1 + z^2/n}$$

Figura 3.9.2. Ecuación de IB3.

Para la frecuencia de la clase, *p* es la proporción de ejemplos que son de esa clase; *n* es el número de ejemplos previamente procesados y *z* es la confianza (análogamente, 0.9 para la aceptación y 0.7 para la eliminación).

IB3 reduce un mayor número de ejemplos que IB2 debido a su escasa sensibilidad al ruido. Ib4 y Ib5 son ampliaciones a Ib3. La primera incorpora la gestión de atributos irrelevantes mediante una asignación de pesos para cada clase. La segunda permite incorporar nuevos atributos al problema después de que el aprendizaje haya sido comenzado.

3.10. SNN: Selective Nearest Neighbour.

El algoritmo del vecino más cercano selectivo [25] extiende heurísticamente la definición de Hart de subconjunto consistente mínimo, SNN encuentra el subconjunto S a partir del conjunto de entrenamiento T donde cada ejemplo de T es más cercano a un ejemplo de S de su misma clase que a un ejemplo de T de distinta clase. SNN garantiza que el conjunto encontrado es el menor conjunto que satisface tales condiciones.

Este requisito adicional da lugar a una relación que asocia a cada ejemplo los ejemplos que son de la misma clase y que están mas cerca que cualquier ejemplo de otra clase.

Mucho más complejo que los algoritmos anteriores (el tiempo de aprendizaje es significativamente mayor, con orden $O(mn^2+n)$) SNN construye una matriz binaria A de dimensión $N \times N$ donde A_{ij} es 1 si el ejemplo j es el vecino más cercano a i y además es de su misma clase ($A_{ii} = 1 \ \forall \ i \in [1,N]$). Una vez construida la matriz, el algoritmo sigue:

1. Para aquella columna i con un único bit 1 en la fila j, borrar las columnas con un 1 en la fila j, borrar ficha fila j y añadir el ejemplo j al conjunto S, $\forall i \in [1..N]$

2. Borrar aquella fila j si existe otra fila k que tiene un 1 donde lo tiene la fila j, $\forall j \in [1..N]$

3. Borrar aquella columna i si existe otra columna k que tiene un 0 donde lo tiene la columna i, $\forall i \in [1..N]$

4. Repetir los pasos 1-3 hasta que no se consiga ninguna modificación. Si se eliminan todas las columnas de la matriz ($A = \varnothing$), el subconjunto S está completo y el algoritmo finaliza.
En otro caso ir a 5.

5. Encontrar la fila j que una vez incluida en S requiera el menor número de filas que deban también incluirse en S. Esto se consigue mediante:

> *(a). Para cada fila j restante se asume que el ejemplo j será añadido en S y que la fila j y cualquier columna con un bit en la fila j será eliminada. Con este supuesto, se buscan el menor nº de filas adicionales que tomarían por lo menos tantos unos como hay en las restantes columnas (no se borran ni la fila j ni las columnas todavía). A partir de los mínimos encontrados para cada fila j, mantener el mínimo absoluto encontrado para cada fila j.*
>
> *(b). Para cada fila j del paso anterior cuyo resultado es el número absoluto de filas adicionales que podrían necesitarse, eliminar j y las columnas con bits a 1 en la fila j, llamando al algoritmo recursivamente empezando por el punto 1. Si el mínimo número de filas fuese realmente utilizado, S se ha completado y el algoritmo termina.*
> *En otro caso restaurar la fila j y las columnas eliminadas e intentar la siguiente fila j posible.*

(c). Si ninguna fila j es satisfactoriamente tratada con el número mínimo, incrementar el mínimo absoluto e intentar el paso (b) otra vez hasta que sea satisfactoria.

Nótese que los únicos pasos en los que se eligen ejemplos para incluirlos en S son el primero y el quinto. SNN es decremental y busca retener ejemplos internos, por lo que es sensible al ruido.

3.11. Editado de VORONOI.

Definición.

Sea U una base de datos de ejemplos y $d:(R^M \times R^M) \rightarrow R^+_0$ una función distancia:

Para cualquier subconjunto $A \in U$ de tamaño $m = |A|$, $1 \leq m < N$, se define la *celda de Voronoi* de orden m de A como aquel conjunto de ejemplos que mejor representa al subconjunto A, esto es:

$$CeldaVoronoi(A) = \{ x \in R^M \mid \forall p_i \in A, p_j \in \{U \backslash A\} \cdot d(x, p_i) \leq d(x, p_j) \}$$

A partir de la definición anterior se define el *diagrama de Voronoi* de orden m de un conjunto U como el conjunto de todas las celdas de Voronoi de orden m, esto es:

$$DiagramaVoronoi_m(U) = \{ CeldaVoronoi(A) \mid A \subset U \ \Lambda \ |A| = m \}$$

Para cualquier ejemplo $p \in U$ se define la *NN-Celda* como:

$$NN\text{-}Celda(p) = \{ x \in R^M \mid \forall p' \in U - \{p\} \cdot d(x, p) \leq d(x, p') \}$$

A partir de la definición anterior se define el *NN-Diagrama* de una base de datos U como:

$$NN\text{-}Diagrama(U) = \{ NN\text{-}Celda(p) \mid p \in U \}$$

De acuerdo con la definición anterior, la suma de los volúmenes de todas las NN-celdas es el volumen del espacio de datos. Si fuésemos capaces de determinar eficientemente las NN-celdas, almacenarlas y encontrar directamente la NN-celda que contiene a un ejemplo cualquiera, entonces la costosa consulta del vecino más cercano sería ejecutada por un único acceso a la NN-celda correspondiente.

En general, sin embargo, calcular las NN-celdas consume bastante tiempo y requiere, al menos, $\Omega(N \log N)$ para $d \in \{1,2\}$ y $\Omega(N^{\lceil d/2 \rceil})$ para $d \geq 3$; en el peor caso para una función de distancia euclídea [23].

Además, el espacio requerido para almacenar las NN-celdas (número de k caras de los NN-diagramas) es $O(N^{\min(d+1-k, \lceil d/2 \rceil)})$ para $k \in [0, d-1]$ en el peor caso [16], haciendo prácticamente imposible almacenarlos explícitamente.

Una solución más abordable consiste en utilizar aproximaciones de las NN-celdas, lo cual ha conseguido satisfactorios resultados en el contexto de las bases de datos geográficas [6].

En principio, cualquier aproximación tal como hiperrectangulos, hiperrectangulos rotados, esferas, elipsoides, etc. pueden emplearse. Así por ejemplo, la aproximación MBR [3] de una NN-celda (NNC) es el menor hiperrectangulo que la contiene:

Sea MBR = (l1, h1; ...; ld,hd) de NNC, es decir, para $i \in [1,d]$, entonces:
$L_i = \min \{ p_i \mid p \in NNC\}$ y además, $h_i = \max\{p_i \mid p \in NNC\}$

Dado un conjunto de ejemplos, un diagrama de Voronoi es una partición del espacio en regiones dentro de las cuales todos los posibles ejemplos están mas cerca de uno particular (denominado nodo) que a cualquier otro. Si dos regiones de Voronoi son fronterizas, los nodos pertenecientes a estas regiones están conectados con una arista y tales nodos son llamados vecinos de Voronoi o vecinos de Delaunay. Considerando la técnica de los vecinos más cercanos se observa como calcular el diagrama de Voronoi utilizando los ejemplos del conjunto inicial como nodos, cuando algún ejemplo desconocido cae una región particular de Voronoi, esta mas cerca del nodo de esa región que de cualquier otro, ya que este nodo debe ser el vecino más cercano a cualquier ejemplo dentro de la región. Las fronteras de decisión de una regla de decisión separan todos los ejemplos del espacio en regiones de la misma clase.

Obviamente las fronteras de decisión generadas por la técnica de los vecinos más cercanos pertenecen al diagrama de Voronoi, En otras palabras, las fronteras de los diagramas de Voronoi que separan las regiones cuyos nodos son de diferente clase contribuyen a una parte de las fronteras de decisión. Aunque las fronteras de decisión raramente son calculadas explícitamente, es claro que las fronteras de decisión son suficientes para clasificar cualquier nuevo ejemplo. De hecho, los nodos de las regiones de Voronoi cuyas fronteras no contribuyen las fronteras de decisión son redundantes y pueden ser eliminados del conjunto de entrenamiento. Además, calcular los diagramas de Voronoi cuando la dimensionalidad supone un coste muy elevado. Cualquier algoritmo llevara por lo menos $O(N^{d/2})$, donde N es el número de ejemplos y d el número de dimensiones. El algoritmo de editado de Voronoi puede resumirse en tres grandes bloques:

Calcular la triangulación de Delaunay para el conjunto de entrenamiento.
Visitar cada nodo, marcándolo si todos sus vecinos de Delaunay son de la misma clase que el nodo actual.
Borrar todos los nodos marcados, devolviendo los restantes como subconjunto editado.

VORONOI (T, S, d)

Entrada:
S: $R^M \times N$
d: $(R^M \times R^M) \rightarrow R^+_0$
$T = \{e_1, ..., e_N\} / \forall i \in [1, N] \cdot e_i = (x_i, c_i), x_i \in R^M, c_i \in [1, G]$

Salida: S

Pre: $\forall e_i = (x_i, c_i), e_j = (x_j, c_j) \in T \cdot c_i = c_j \leftrightarrow x_i \neq x_j$

Post: $S \subseteq T$

```
INICIO

    Calcular la triangulación de Delaunay S a partir de T

    Para cada i ∈ [1, |S|]
        Flags[i] ← CIERTO

    Para cada nodo s_i ∈ S
        V ← Vecninos(s_i)

        j ← 1
        Mientras (j ∈ [1, |V|]) Λ Flags[i]
            Si clase(s_i) ≠ clase(v_j) →
                Flags[i] ← FALSO

    Para cada i ∈ [1, |S|]
        Si Flags[i] →
            S ← S - {s_i}

FIN
```

Figura 3.11.1. Editado de Voronoi.

Las principales propiedades del algoritmo de editado basado en Voronoi son dos:

Se preservan las fronteras de decisión.
Se genera el conjunto editado más pequeño que las preserva.

Una desventaja del editado de Voronoi es que aunque preserve fronteras de decisión, dos ejemplos de clases distintas podrían ser suficientes para establecer una frontera. Es decir, deja demasiados ejemplos en el conjunto editado que son fácilmente separables de otras clases para preservar esas fronteras, aunque podrían ser eliminados intuitivamente. No es difícil apreciar que si hay degeneraciones, concretamente más de dos ejemplos coplanares, entonces el conjunto editado resultante no será mínimo.

3.12. Editado de GABRIEL.

Dos ejemplos x e y son *vecinos de Gabriel* si su esfera diametral (aquella esfera que tienen el segmento xy como diámetro) no contiene a cualquier otro ejemplo. Un grafo donde todos los pares de vecinos de Gabriel están conectados con una arista se denomina *grafo de Gabriel*.

El grafo de Gabriel puede calcularse descartando aristas a partir de la triangulación de Delaunay. Sin embargo, la complejidad del cálculo puede ser en el peor caso $O(N^{d/2})$, por lo que esta propuesta no resulta atractiva cuando el número de dimensiones es elevado.

Claramente el grafo de Gabriel se puede obtener mediante fuerza bruta si para cada potencial par de vecinos x e y verificamos si cualquier otro ejemplo z esta contenido en la esfera diametral:

$$dist^2(z, x) + dist^2(z, y) < dist^2(x, y)$$

Puesto que hay n2 vecinos potenciales y necesitamos verificar cada par comprobando los n-1 restantes, lo que requiere O(d), la complejidad resultante es $O(dn^3)$.

El algoritmo de fuerza bruta puede ser fácilmente mejorado con la siguiente heurística, reduciendo la complejidad media a $O(dn^2)$:

> *Para cada ejemplo x se crea una lista de ejemplos que sean su vecino de Gabriel potencial.*
> *Cuando comprobamos si un ejemplo y es vecino de Gabriel, se realiza el test para cada ejemplo z si está contenido en la esfera diametral. De esta forma puede comprobarse si esta en la mitad derecha con respecto al plano p que es ortogonal al diámetro xy.*
> *Si lo es, el ejemplo z no puede ser vecino potencial de Gabriel de x ya que el ejemplo y estará contenido en la esfera de diámetro xz.*
> *Así lo descartamos de la lista de vecinos potenciales y prescindimos de un costoso cálculo que habría sido necesario de otra manera.*

El algoritmo de Gabriel se puede formular de forma análoga al algoritmo de editado de Voronoi:

> *Calcular el grafo de Gabriel para el conjunto de entrenamiento.*
> *Visitar cada nodo, marcándolo si todos sus vecinos de Gabriel son de la misma clase que el nodo actual.*
> *Borrar todos los nodos marcados, devolviendo los restantes como subconjunto editado.*

El conjunto editado de Gabriel es siempre un subconjunto del editado de Voronoi debido al hecho de que un grafo de Gabriel de un conjunto de ejemplos es un subgrafo de la triangulación de Delaunay para dicho conjunto, por lo que reduce el tamaño del conjunto de entrenamiento aun más que el editado de Voronoi. Aunque el conjunto editado resultante no preserve las fronteras de decisión originales, los cambios ocurren principalmente fuera de las zonas de interés.

3.13. Editado con grafos de Vecinos Relativos.

Utilizando la misma metodología que con los grafos de Gabriel, puede ser ventajoso restringir aún mas la definición de vecino con objeto de obtener conjuntos editados mas pequeños. El grafo de vecinos relativos (estudiado en [28]) proporciona la siguiente propiedad:

> Dos ejemplos x e y son llamados vecinos relativos si para todo ejemplo z se cumple que:
>
> $$D(x, y) < \max\{D(x, z), D(y,z)\}$$

Geométricamente, la expresión implica que ningún ejemplo z está contenido en la luna construida sobre los ejemplos x e y.

Un grafo de vecinos relativos es un grafo donde todos los pares de vecinos relativos están conectados por una arista. El algoritmo para calcular el grafo de vecinos relativos es análogo al utilizado para calcular el grafo de Gabriel.

El algoritmo puede expresarse de forma similar al algoritmo de editado de Voronoi y el editado de Gabriel, sustituyendo el grafo de vecinos relativo:

> *Calcula el grafo de vecinos relativos para el conjunto de entrenamiento.*
> *Visita cada nodo, marcándolo si todos sus vecinos relativos de la misma clase que el nodo actual.*
> *Borra todos los nodos marcados, devolviendo los restantes como subconjunto editado.*

Al ser el grafo de vecinos relativos un subconjunto del grafo de Gabriel, se reduce el tamaño del conjunto editado, si embargo los cambios de las fronteras pueden llegar a ser severos tanto en zonas importantes como en aquellas irrelevantes.

3.14. EPO: Editado por Proyeccion Ordenada.

Según [26], tras observar conjuntos de entrenamiento, se puede colegir que muchos de los ejemplos de determinadas bases de datos son interiores a regiones ortoédricas, lo cual puede interpretarse como que esos ejemplos no son importantes a la hora de establecer la frontera de esa región. Esa es la idea central del algoritmo: eliminar los ejemplos que no sirvan para obtener la frontera de una posible región ortoédrica.

Para ello, no se calcula la envolvente conexa, sino que se van a "intuir" heurísticamente las posibles regiones, a pesar de que algunas no lo sean, ya que el tratamiento que el algoritmo realiza a un ejemplo no lo hace en toda su dimensionalidad, sino que operará con la proyección del ejemplo en cada dimensión. Como resultado se obtiene un algoritmo cuyas características más importantes son las siguientes:

- Reducción importante de ejemplos de la base de datos.
- Coste computacional inferior a los algoritmos convencionales O(M·N·log N).
- No necesita cálculo de distancias.
- Conservación de las fronteras de decisión, especialmente desde el punto de vista de la aplicación de clasificadores que generen reglas de decisión paralelas a los ejes.
- Reducción del tamaño del árbol de decisión o el número de reglas de decisión.

Para mostrar gráficamente (figura 1) la idea del algoritmo utilizamos una base de datos bidimensional, la cual contiene doce ejemplos con etiquetas I (impares) y P (pares).

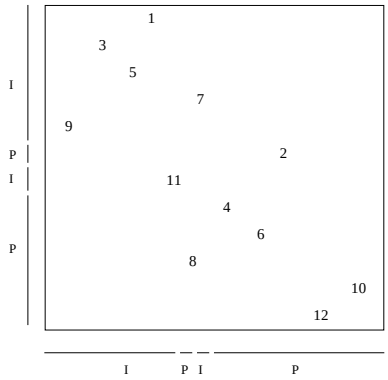


Fig.3.14.1. Base de datos de ejemplo.

Analizando la proyección sobre el eje de abscisas, observamos que existen cuatro secuencias ordenadas {I, P, I, P} con los ejemplos {[9, 3, 5, 1, 11], [8], [7], [4, 6, 2, 12, 10]}. Idénticamente, con la proyección sobre el eje de ordenadas, calculamos las secuencias ordenadas {P, I, P, I}, con los siguientes ejemplos {[12, 10, 8, 6, 4], [11], [2], [9, 7, 5, 3, 1]}.

Existen dos grandes regiones para ambas proyecciones, en las cuales no hay ejemplos con la otra etiqueta. En este caso, aquí actuará el algoritmo eliminando aquellos ejemplos que no sean importantes para conservar la frontera del ortoedro mínimo que los envuelve. Tales regiones ortoédricas se muestran en la figura 3.14.2 (izq.). Sin embargo, estas regiones ortoédricas tienen intersección distinta de vacío, por lo que, desde la perspectiva del algoritmo, que basa su proceso en la proyección de los ejemplos, no es útil. Por lo tanto, las regiones ortoédricas que vamos a tratar son aquellas que son disjuntas. Estas regiones ortoédricas se muestran en la figura 3.14.2 (dcha.).

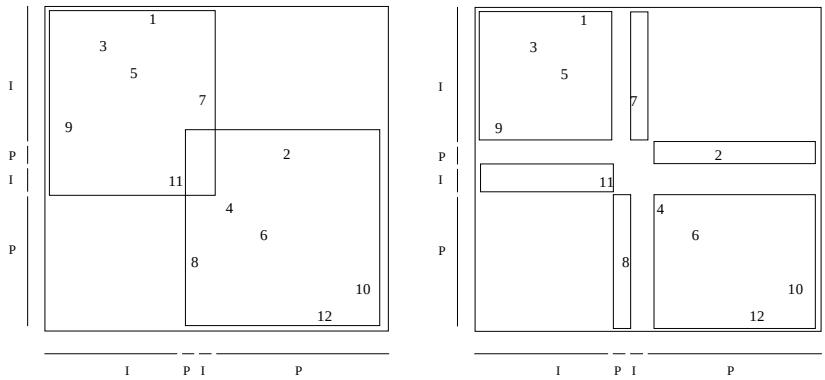


Fig.3.14.2. Regiones ortoédricas (izq); regiones ortoédricas con intersección nula en la proyección (dcha).

El algoritmo mantendrá los valores mínimo y máximo de cada secuencia ordenada para cada proyección, por lo tanto, mantendrá cada región ortoédrica. El resultado que ofrece la ejecución del algoritmo a este ejemplo será el siguiente:

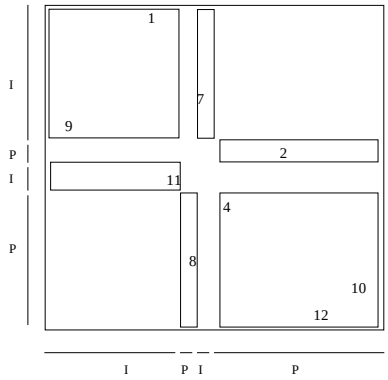


Fig. 3.14.3. Resultado de ejecución de EPO.

Se han eliminado los ejemplos 3, 5 y 6. Obsérvese que se han mantenido los ejemplos que permiten a un clasificador ortoédrico (tal como C4.5) encontrar una de las cuatro posibles siguientes regiones (figura 3.14.4).

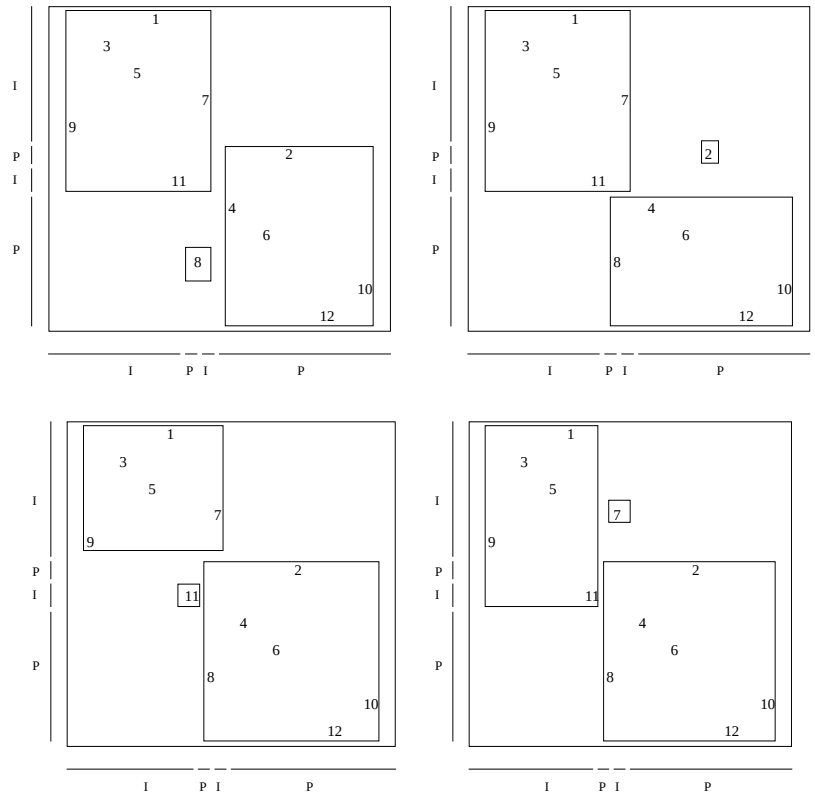


Fig. 3.14.4. Eliminando los ejemplos 3, 5 y 6, se conservan los ejemplos que definen las fronteras de las regiones ortoédricas, para cualquiera de las mejores soluciones.

Analizando las cuatro soluciones mostradas en la figura 4, por ejemplo, para la figura inferior izquierda se observa que las regiones están delimitadas por ejemplos que son frontera [9, 1, 7], [11] y [8, 2, 12, 10], entre los cuales no se encuentran el 3, 5 y 6. Para las cuatro soluciones ocurre lo mismo, por lo tanto, se puede concluir afirmando que los ejemplos eliminados no influyen en la obtención de regiones. Y, en definitiva, el método EPO, reduce la base de datos manteniendo los ejemplos fundamentales para la obtención de tales regiones ortoédricas.

La consecuencia inmediata de la heurística aplicada es que el conjunto reducido de ejemplos no necesariamente será el menor; de hecho, sólo sería el menor cuando ninguna de las proyecciones para cada dimensión tuviese una intersección distinta de vacia para cualesquiera dos regiones ortoédricas de ejemplos con la misma etiqueta.

Definición.

Se denomina *secuencia proyección ordenada* a la secuencia formada por la proyección sobre el atributo i de los ejemplos del universo. Dicha secuencia está ordenada de menor a mayor valor del atributo y contiene a los números de los ejemplos del universo.

Por ejemplo, en la figura 1, para el atributo 1 tendríamos {9, 3, 5, 1, 11, 8, 7, 4, 6, 2, 12, 10} y para el atributo 2 tendríamos {12, 10, 8, 6, 4, 11, 2, 9, 7, 5, 3, 1}.

Se denominan *particiones de la secuencia ordenada en subsecuencias* constantes a subsecuencias construidas a partir de la secuencia proyección ordenada de un atributo i que mantienen el orden de la proyección, que son disjuntas y que agrupan a ejemplos con la misma etiqueta.

En la figura 1, tendríamos para el atributo 1 {[9, 3, 5, 1, 11], [8], [7], [4, 6, 2, 12, 10]}; y para el atributo 2 tendríamos {[12, 10, 8, 6, 4], [11], [2], [9, 7, 5, 3, 1]}.

Se denomina *fortaleza de un ejemplo* al número de ocasiones en que el ejemplo es interior a una partición para las particiones obtenidas a partir de las secuencias proyección ordenadas de todos los atributos.

En el ejemplo anterior, dadas las particiones {[9, 3, 5, 1, 11], [8], [7], [4, 6, 2, 12, 10]} y {[12, 10, 8, 6, 4], [11], [2], [9, 7, 5, 3, 1]} se tendrían las siguientes fortalezas para los ejemplos:

fortaleza=0 $\Rightarrow$ {9, 11, 4}
fortaleza=1 $\Rightarrow$ {1, 8, 7, 2, 12, 10}
fortaleza=2 $\Rightarrow$ {3, 5, 6}

Se denominan *ejemplos prescindibles* a aquellos cuya fortaleza es igual al número de atributos de la base de datos (dimensionalidad del universo).

De lo anterior, prescindibles(U)={3, 5, 6}.

EPO supone, por sus características como reductor del número de ejemplos de la base de datos y del número de reglas de decisión, una interesante aplicación como técnica de preprocesado para clasificadores que producen reglas de decisión paralelas a los ejes.

Incrementa muy ligeramente la tasa de error (nótese que en algunos casos el número de ejemplos resultante de la aplicación del algoritmo EPO al entrenamiento es inferior al del test).

No necesita parametrización, como ocurre en los métodos basados en k-NN.

Si añadimos que el coste computacional es inferior a las técnicas enumeradas y que prescinde totalmente de distancias, lo convierten en un método determinista, robusto, eficiente y eficaz.

EPO (T, S)

Entrada:

S: $R^M \times N$

$T = \{e_1, ..., e_N\} / \forall i \in [1, N] \cdot e_i = (x_i, c_i), x_i \in R^M, c_i \in [1, G]$

Salida: S

Pre: $\forall e_i = (x_i, c_i), e_j = (x_j, c_j) \in T \cdot c_i = c_j \leftrightarrow x_i \neq x_j$

Post: $S \subseteq T$

```
INICIO

    S ← T
    Para cada s_i ∈ S
        fortaleza[s_i] ← 0

    Para cada atributo a_i ∈ S
        S_i ← Ordenar S por el atributo a_i
            Para cada s_j ∈ S_i
                Si Interior(s_j) →
                    fortaleza(e_j) ← fortaleza (e_j) + 1

      Para cada s_i ∈ S
            Si fortaleza(s_i) = M →
                S ← S - {s_i}

FIN
```

Figura 3.15.6. Algoritmo EPO.

4. Referencias.

[1] Fix, E. Hodges Jr., J. (1951). *Discriminatory Analysis: nonparametric discrimination. Consistency properties*. In project 21-49-004, Report no 4, pp. 261-279. USAF School of Aviation Medicine, Randolph Field, Tex.

[2] T.M. Cover and P.E. Hart, *Nearest neighbor pattern classification*, IEEE Transactions on Information Theory IT-13 (1967), no. 21-27.

[3] P. Hart, *The condensed nearest neighbor rule*, IEEE Transactions on Information Theory 14 (1968), no. 3, 515-516.

[4] T.M. Cover, *Estimation by nearest neighbor rule*, IEEE Transactions on Information Theory IT-14 (1968), 50-55.

[5] D.W. Aha, D. Kibler, and M.K. Albert, *Instance-based learning algorithms*, Machine Learning 6 (1991), 37-66

[6] T. Mitchell (1997). *Machine Learning*. New York: The McGraw-Hill Companies, Inc.

[7] D.W. Aha, D. Kibler, and M.K. Albert, *Instance-based learning algorithms*, Machine Learning 6 (1991), 37-66

[8] C. Stanfill and Waltz, *Toward memory-based reasoning*, Communications of the ACM 29 (1986), 1213-1228.

[9] D. Kibler and D.W.Aha, *Learning representative exemplars of concepts: An initial case study*, Proceedings of Fourth Intrenational Workshop on Machine Learning (Irvine, CA), Morgan Kaufmann, 1987, pp. 24-30.

[10] Bradshaw G. (1997). *Learning about the speech sound. The NEXUS project*. Proceedings of the 4th International Workshop on Machine Learning, pp 1-11.

[11] S. Cost and S. Salzberg (1993). *A weighted nearest neighbor algorithm for learning with symbolic features*, Machine Learning 10, 57-78.

[12] Breidman L., Freidman J., Olshen R. and Stone G. (1984). *Classification and Regression Trees*. Belmont, CA, Wadsworth International Goup.

[13] Terry R. Payne, *Dimensionality Reduction and Representation for Nearest Neighbor Learning. Doctoral Dissertation.* University of Aberdeen 1999.

[14] F. P. Preparata and M. I. Shamos (1985). *Computational geometry*, Springer.

[15] S. Salzberg (1991a). *A nearest hyperrectangule learning method*, Machine Learning 6, 227-309.

[16] D.R. Wilson and T.R. Martinez, *Improved heterogeneous distance functions*, Journal of Artificiall Intelligence Research 6 (1997), no. 1, 1-34.

[17] H. Minkowsky, *Geometric der zahlen*, Teubner, Leizpig, 1896.

[18] Nosofsky R. (1984). *Choice, Similarity and the Contest Theory of Classification*. Journal of Experimental Psychology: Learning, Memory and Cognition 10(1), 104-114.

[19] S. Salzberg (1991b). *Distance metrics for Instance-Based Learning*, In ISMIS'91, 6th International Symposium, Methodologies for Intelligent Systems, pp 399-408.

[20] Aha, D. W. (1997). *Beyond classification of feature vectors. MLnet ECML'97 workshop: Case-based learning: MLnet News, 5:1, 8--11.*

[21] G. W. Gates, *The reduced nearest neighbor rule*, IEEE Transactions on Information Theory 18 (1972), 431-433.

[22] D. Wilson, *Asyntotic propierties of nearest neighbor rules using edited data*, IEEE Transactiones on Systems, Man and Cybernetics 2 (1972), no. 3, 408-421.

[23] I. Tomek, *An experiment with the edited nearest-neighbor rule*, IEEE Transactions on System, Man and Cybernetics 6 (1976), no. 6, 448-452.

[24] D. G. Lowe, *Similarity metric learning for a variable-kernel classifier*, Neural Computation 7 (1995), no. 1. 72-85.

[25] G. Ritter, H. Woodruff, S. Lowry, and T. Isenhour, *An algorithm for a selective nearest neighbor decisión rule*, IEEE Transactions on Information Theory 21 (1975), no. 6, 665-669.

[26] J. Aguilar, J. Riquelme y M. Toro. Data Set Editing by Ordered Projection. European Conference on Artificial Intelligence, ECAI'2000, Berlin (2000).