Un estudio experimental del algoritmo LVQ

An experimental study of LVQ algorithm

Daimerys Ceballo Gastell [1*], Alain Guerrero Enamorado [2], Yanet De Diego Ceruto [3]

[1] Universidad de las Ciencias Informáticas. dceballo@uci.cu
[2] Universidad de las Ciencias Informáticas. alaing@uci.cu
[3] Universidad de las Ciencias Informáticas. ycdediego@uci.cu
[*] Autor para correspondencia: dceballo@uci.cu

Resumen
Este trabajo se realizó con el objetivo de obtener información empírica sobre el comportamiento del algoritmo LVQ (Aprendizaje por Cuantificación Vectorial) en la tarea de clasificación de información en disímiles dominios de aplicación. Para acometer esta tarea, se realizó una evaluación extensiva sobre 21 colecciones de entrenamiento del mundo real. Como resultado de este proceso de evaluación, se determinó, que la versión LVQ2.1 se desempeñó mejor en la métrica de exactitud predictiva que las versiones LVQ1 y LVQ3 frente a cuatro de los algoritmos de clasificación reconocidos mundialmente entre los primeros. Este trabajo sirve como punto de partida para profundizar en futuras evaluaciones del LVQ en colecciones de datos desbalanceadas, ruidosas o con datos omitidos.
Palabras clave: aprendizaje por cuantificación vectorial, árboles de decisión, bayesiano ingenuo, clasificación de información, k-vecino más cercanos, máquinas de soporte vectorial.

Abstract
The aim of this work was obtain empirical information on the behavior of the LVQ (Learning Vector Quantization) algorithm in the task of information classification and in dissimilar application domains. To accomplish this task, an extensive evaluation in 21 data sets of real world was conducted. As a result of this evaluation process, was determined that the LVQ2.1 version achieved better predictive accuracy than LVQ1 and LVQ3 against four of the recognized top-ten classification algorithms. This work serves as a starting point for further evaluations of the LVQ in data sets unbalanced, noisy or missing data.
Keywords: *classification of information, decision trees, k-nearest neighbors, learning vector quantification, naive bayes, and support vector machines.*

1 Introducción

Cuando de clasificar información se trata un método simple para hacerlo pudiera ser estimar una función lineal que devuelva un valor que represente la membresía a una clase determinada, utilizando para ello como variables de entrada los atributos predictores del ejemplo que se quiere clasificar. Sin embargo en los problemas del mundo real no existen funciones lineales capaces de hacer esto, nos encontramos entonces frente a problemas no-lineales.

En este trabajo se realiza un estudio experimental del algoritmo de clasificación Aprendizaje por Cuantificación Vectorial (LVQ) propuesto por Tuevo Kohonen en el trabajo [Kohonen 1988]. La idea fundamental que se pretende

con este artículo es realizar una comparación extensiva de varias de las versiones del LVQ frente a cuatro de los algoritmos de clasificación más trascendentales de los últimos años, al mismo tiempo que se realiza dicha evaluación en un amplio dominio de aplicaciones (21 colecciones de datos extraídas de problemas del mundo real) que van desde campos tan variados como la medicina hasta la agricultura. En la sección 2 se detalla el algoritmo pasando por sus principales versiones, posteriormente se exponen las principales características del resto de los algoritmos utilizados para la experimentación. El algoritmo LVQ también puede entrenarse sin clases de salida utilizando un aprendizaje no-supervisado para aplicarlo a la tarea de agrupamiento en lugar de la tarea de clasificación. En los artículos [Bezdek and Pal 1995; Karayiannis and Pai 1996; Karayiannis 1997; Karayiannis 1999; Pal et al. 1993] pueden encontrarse este tipo de aplicaciones. El centro de este trabajo es sin embargo su aplicación en la tarea de clasificación.

Recientemente algunas de las extensiones que se le han realizado al LVQ toman como punto de partida la sustitución de le medida de distancia con métricas más generales tales como: medida euclidiana pesada [Hammer and Villmann 2002], matriz de relevancia adaptativa [Schneider et al. 2009], métrica pseudo-euclidiana [Hammer et al. 2011] y métricas en el espacio del núcleo característico [Qin and Suganthan 2004]. A lo largo de los años muchas han sido las aplicaciones del LVQ, algunas de ellas son las aplicaciones en el campo del procesamiento de señales e imágenes [Bashyal and Venayagamoorthy 2008; Blume and Ballard 2007], en la industria [Ahn and Nguyen 2007; Bassiuny et al. 2007] y en la medicina [Anagnostopoulos et al. 2001; Chen 2012; Dieterle et al. 2003; Dutta et al. 2001]. Una lista más exhaustiva puede encontrarse en línea en la base de datos [Neural Networks Research Centre 2007].

En la sección 3 se mencionan las herramientas y colecciones de entrenamiento utilizadas, se evalúan los resultados obtenidos y se aplican las pruebas estadísticas para analizar los resultados. Finalmente se exponen los resultados obtenidos en la sección 4 al mismo tiempo que se dan algunas recomendaciones para la continuidad de este trabajo.

2 Materiales y métodos

En esta sección se detallan los algoritmos que se involucraron en los diferentes experimentos realizados, partiendo de la explicación detallada del algoritmo LVQ en sus variantes fundamentales y exponiendo las principales características del resto de los algoritmos utilizados para la experimentación.

2.1 Aprendizaje por cuantificación vectorial (LVQ)

Tuevo Kohonen presentó un modelo de red neuronal [Kohonen 1988] con capacidad para formar mapas de características de manera similar a como ocurre en el cerebro. Este modelo parte de dos variantes: el aprendizaje por cuantificación vectorial (LVQ, *Learning Vector Quantization*) y los mapas auto-organizados (SOM, *Self Organizing Map*). En ambos casos se construyen mapas topológicos donde son agrupadas de cierta manera las entidades con características similares. Ambos algoritmos utilizan un aprendizaje competitivo reforzado, distinguiéndose una etapa

de entrenamiento y otra de explotación [Kohonen 1990; Kohonen 2001]. Una de las principales diferencias que existen entre el algoritmo LVQ y su precursor SOM radica en que este último agrupa las instancias de acuerdo a su similaridad, mientras que en el primero una vez que fueron agrupadas se les asigna además una clase. Esta última característica del LVQ es la que permite que sea utilizado para la tarea de clasificación.

El modelo LVQ utiliza una red de dos capas como el mostrado en la Figura 1, sus conexiones se realizan hacia delante. Los pesos de conexión se representan mediante una matriz de pesos W según cada capa de la red neuronal.

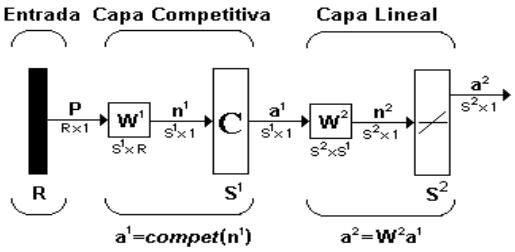

Figura 1. Diagrama en bloques de una red LVQ.

El proceso de entrenamiento empieza con la inicialización de los pesos sinápticos. Dicha inicialización se puede realizar de varias formas, entre ellas están: pesos nulos, pesos aleatorios de pequeños valor absoluto, o pesos con un valor de partida predefinido. Luego se escoge una instancia del conjunto de entrenamiento y se presenta a la entrada de la red. Cada vector prototipo (vector fila de la matriz de pesos W^1, o vector formado con el conjuntos de todos los pesos que unen cada atributo de entrada con una neurona en particular de esta capa competitiva), de la capa competitiva calcula su distancia con respecto al vector de entrada en base a alguna función de similaridad, como por ejemplo: producto interno, función coseno, distancia *hamming*, distancia euclidiana, etc. El criterio de medida utilizado en este trabajo fue la distancia euclidiana definida por la siguiente ecuación:

$$dist_{eucl}(P, W_i^1) = \sqrt{\sum_{j=1}^{R}(r_j - w_{ij}^1)^2} \quad (1)$$

Después se establece una competición entre los vectores prototipos y el vector de entrada para determinar cuál es el más cercano al vector de entrenamiento el cual será proclamado ganador de dicha competición, es decir, se determina la neurona ganadora, cuya distancia sea la menor de todas.

La salida de la primera capa de la red LVQ sería:

$$a^1 = Compet(n^1) \quad (2)$$

Donde a^1 es la salida y n^1 es la entrada neta a la función de transferencia, esta entrada neta sería:

$$n^1 = \begin{bmatrix} dist_{eucl}(P, W_1^1) \\ dist_{eucl}(P, W_2^1) \\ \vdots \\ dist_{eucl}(P, W_{S^1}^1) \end{bmatrix} \quad (3)$$

La función $Compet$ es una función de transferencia que encuentra el índice i-ésimo de la neurona con la entrada neta con mayor similaridad con la entrada actual y fija su salida en uno, el resto de las neuronas devuelven como salida cero [McClelland et al. 1987]. Dicho de otra forma, la neurona cuyo vector de pesos esté más cercano al vector de entrada tendrá salida 1 y el resto de las neuronas tendrán salida 0; la salida no cero representa una sub-clase, muchas neuronas (subclases), conforman una clase. La segunda capa de la red LVQ es usada para combinar subclases dentro de una sola clase, esto es realizado por la matriz de pesos W^2. Las columnas de W^2 representan las subclases y las filas representan las clases, W^2 tiene un solo 1 en cada columna, todos los demás elementos son cero, la fila de la columna en la cual se presenta el 1 índica cuál es la clase a la que la subclase pertenece. Dicho de otra forma si $W_{ki}^2 = 1$; entonces la subclase i pertenece a la clase k. Una propiedad importante de esta red, es que el proceso de combinar subclases para formar clases, permite la creación de clases más complejas. Una sola capa competitiva estándar tiene la limitación de que puede crear solo regiones de decisión convexas; la red LVQ soluciona esta limitación [Kohonen 2001].

Para generar la matriz W^2, antes de que se inicie el aprendizaje cada neurona en la segunda capa es asignada a una neurona de salida; por lo general igual número de neuronas ocultas son conectadas a cada neurona de salida, para que cada clase pueda ser conformada por el mismo número de regiones convexas. Una vez que ha sido definida nunca será alterada durante el proceso de entrenamiento que solo modificará los pesos de W^1. Para obtener la salida final se calcula $a^2 = W^2 * a^1$ que indicará a quien está siendo asignado el vector de entrada P presentado inicialmente a la red. Se procede entonces a la actualización de los pesos sinápticos de la neurona ganadora por medio de la regla de Kohonen [Kohonen 1990] que es empleada para mejorar la capa oculta de la red LVQ.

Puesto que los vectores prototipos y los patrones de entradas están etiquetados con las clases a las que pertenecen, las correcciones son del tipo premio o castigo. Si se acertó en la clasificación (la clase del vector prototipo y la clase del vector de entrada coinciden) se premia al vector prototipo acercándolo al vector de entrada. Por el contrario, si la clasificación no se ha realizado correctamente (la clase seleccionada por el vector prototipo es diferente a la clase del vector de entrada), se castiga al vector prototipo alejándolo del vector de entrada. La modificación de los pesos en el instante t se realiza según la siguiente ecuación:

$$W_i^1(t+1) = \begin{cases} W_i^1(t) + \alpha(t) * dist_{eucl}\left(P(t), W_i^1(t)\right); si\ W_i^1(t)\ y\ P(t)\ pertenecen\ a\ la\ misma\ clase \\ W_i^1(t) - \alpha(t) * dist_{eucl}\left(P(t), W_i^1(t)\right);\ si\ W_i^1(t)\ y\ P(t)\ pertenecen\ a\ clases\ diferentes \\ W_i^1(t);\ si\ i \neq k \end{cases} \quad (4)$$

Donde $\alpha(t)$ es la tasa de aprendizaje, este es un parámetro que utilizan muchas redes neuronales en su fase de entrenamiento para controlar la velocidad de convergencia de los pesos sinápticos. Resulta de gran importancia la elección de un valor adecuado para la tasa de aprendizaje que permita hallar un punto de equilibrio entre la velocidad de convergencia y la estabilidad final de los vectores prototipo. Una $\alpha(t)$ cercana a 0, hace que el aprendizaje sea lento pero asegura que cuando un vector prototipo haya alcanzado una clase se mantenga allí indefinidamente; sin embargo valores altos de $\alpha(t)$ (cercanos a 1) hacen que el acercamiento del vector prototipo al vector de entrada sea muy rápido, pero tiene como desventaja que los pesos tendrán grandes oscilaciones provocando muchas veces inestabilidad en el proceso de convergencia. Otra forma es utilizar un $\alpha(t)$ adaptativo, se inicializa con algún valor no muy alto, por ejemplo 0,3; y se decrementa a medida que avanza el entrenamiento según alguna función, de manera que al final del proceso su valor sea muy cercano a 0. Normalmente, por motivos computacionales, se suele elegir una función lineal, monótona decreciente y que tome valores en el intervalo (0,1).

2.2 Versiones del algoritmo LVQ

Kohonen y sus colaboradores han desarrollado varias versiones del algoritmo inicial LVQ1, a las cuales se les conoce por LVQ2.1, LVQ3 y OLVQ1.

LVQ2.1:

El cambio fundamental respecto al algoritmo original radica en la forma de actualizar los pesos. En este en lugar de seleccionar un único vector prototipo ganador son seleccionados los dos (W_i y W_j) más próximos al vector de entrada (X), pero con la condición que uno de ellos pertenezca a la clase del vector de entrada y el otro no. Además el vector de entrada debe quedar situado cercano al punto medio de la ventana que se forma entre estos dos vectores prototipo. Definiendo s como el ancho de la ventana (valores recomendados entre 0.2 y 0.3), d_i y d_j como las distancias euclídeas desde el vector de entrada hasta cada uno de los vectores prototipo ganadores se dice que este vector se encuentra dentro de la ventana si satisface la siguiente ecuación:

$$mínimo\left(\frac{d_j}{d_i}, \frac{d_i}{d_j}\right) > \frac{1-s}{1+s} \quad (5)$$

En esta versión las ecuaciones de actualización de los pesos quedan como sigue:

$$W_i^1(t+1) = W_i^1(t) + \alpha(t) * dist_{eucl}\left(X(t), W_i^1(t)\right) \quad (6)$$

$$W_j^1(t+1) = W_j^1(t) - \alpha(t) * dist_{eucl}\left(X(t), W_j^1(t)\right) \quad (7)$$

Con esta variante del algoritmo en cada iteración se premia al mejor vector prototipo que pertenece a la misma clase que el vector de entrada al mismo tiempo que se castiga al vector prototipo más cercano que pertenece a una clase distinta a la del vector de entrada.

LVQ3:

En esta nueva versión se realiza una combinación entre las dos variantes anteriores LVQ1 y LVQ2.1. Al igual que en la versión anterior son seleccionados los dos (W_i y W_j) más próximos al vector de entrada (X). Ahora si W_i pertenece a la clase del vector de entrada y W_j no pertenece a la misma clase que X, y además ambos quedan definidos en la ventana como en LVQ2.1 las modificaciones a los pesos sinápticos se realizan exactamente igual como en las ecuaciones 6 y 7. Pero si por el contrario ambos vectores prototipo pertenecen a la misma clase que X entonces ambos son premiados según las siguientes ecuaciones:

$$W_i^1(t+1) = W_i^1(t) + \varepsilon * \alpha(t) * dist_{eucl}\left(X(t), W_i^1(t)\right) \quad (8)$$

$$W_j^1(t+1) = W_j^1(t) - \varepsilon * \alpha(t) * dist_{eucl}\left(X(t), W_j^1(t)\right) \quad (9)$$

donde ε toma valores en el intervalo [0.1; 0.5].

OLVQ1:

En la variante optimizada del algoritmo LVQ1 cada vector prototipo tiene su propia taza de aprendizaje $\alpha_i(t)$. Por tanto se utilizan las siguientes ecuaciones para la actualización de los pesos:

$$W_i^1(t+1) = \begin{cases} W_i^1(t) + \alpha_i(t) * dist_{eucl}\left(P(t), W_i^1(t)\right); si\ W_i^1(t)\ y\ P(t)\ pertenecen\ a\ la\ misma\ clase \\ W_i^1(t) - \alpha_i(t) * dist_{eucl}\left(P(t), W_i^1(t)\right);\ si\ W_i^1(t)\ y\ P(t)\ pertenecen\ a\ clases\ diferentes \\ W_i^1(t);\ si\ i \neq k \end{cases} \quad (10)$$

Aunque en este caso hay que actualizar también el valor de $\alpha_i(t)$ según las siguientes ecuaciones:

$$\alpha_i(t+1) = \begin{cases} \frac{\alpha_i(t)}{1+\alpha_i(t)};\ si\ W_i^1(t)\ y\ P(t)\ pertenecen\ a\ la\ misma\ clase \\ \frac{\alpha_i(t)}{1-\alpha_i(t)};\ si\ W_i^1(t)\ y\ P(t)\ pertenecen\ a\ clases\ diferentes \\ \alpha_i(t);\ si\ i \neq k \end{cases} \quad (11)$$

En la bibliografía no existen muchas referencias sobre cual algoritmo escoger para un problema en cuestión, el mismo Kohonen sostiene que las cuatro variantes dan resultados muy similares [Kohonen 2001]. Por esta razón se utilizó la versión inicial LVQ1 en los experimentos del presente trabajo puesto que utiliza menos parámetros de ajuste que el resto de las versiones.

2.3 Trabajos Relacionados

En este trabajo se hace una comparación del algoritmo LVQ frente a cuatro técnicas reconocidas por la comunidad científica [Wu et al. 2007]. En particular se realiza la comparación contra los algoritmos: árboles de decisión C4.5 [Quinlan 1993], Bayesiano Ingenuo o NB [Domingos and Pazzani 1997; John and Langley 1995; Maron 1961], K-vecinos más cercanos o KNN [Fix and Hodges 1951; Cover and Hart 1967; Wang et al. 2007] y las Máquinas de Soporte Vectorial o SVM [Cortes and Vapnik 1995]. En todos los casos se utilizaron las implementaciones de la herramienta KEEL[1][Alcalá-Fdez et al. 2008].

2.3.1 Árboles de decisión (C4.5)

Este algoritmo genera reglas de clasificación en forma de árboles de decisión partiendo de un conjunto de entrenamiento, dicho árbol se construye de arriba hacia abajo y en cada paso de ejecución del algoritmo se realiza una prueba en cada nodo (partiendo del nodo raíz) para determinar cuál separa mejor los ejemplos de la colección de acuerdo a la clase a la que pertenecen. Este algoritmo surgió inicialmente como una mejora al ID3 [Quinlan 1986] para permitir información incompleta o valores omitidos en los datos, además maneja atributos continuos utiliza una métrica para la selección de los atributos que maximiza la ganancia de información, finalmente poda el resultado partiendo del principio de mínima longitud de descripción o principio MDL [Rissanen 1978].

Este modelo se puede configurar en KEEL con los siguientes parámetros:

- *Prune*: este parámetro permite activar o desactivar el mecanismo de poda de los árboles.
- *Confidence*: define el nivel mínimo de confianza que debe tener una hoja para mantenerse dentro del árbol.
- *minItemsets*: define el número mínimo de instancias por hojas.

2.3.2 Bayesiano Ingenuo (NB)

Este algoritmo se basa en la aplicación del teorema de probabilidad condicional de Bayes [Bayes 1763], al mismo tiempo que se asume independencia entre los atributos predictores para poderse aplicar dicho teorema. El algoritmo calcula la probabilidad condicional de cada ejemplo pertenecer a cada clase, entonces se asigna como clase de un nuevo ejemplo la de mayor probabilidad condicional calculada anteriormente. Este tipo de clasificador a pesar de su sencillez ha demostrado comportarse muy bien en muchos problemas incluso aun cuando no se cumple la asunción de independencia de los atributos de entrada.

[1] Knowledge Extraction based on Evolutionary Learning. http://www.keel.es/

Este modelo se puede configurar en KEEL sin ningún parámetro, aunque la implementación que se utilizó era solo para atributos nominales, por lo cual se utilizó previamente en las colecciones numéricas una discretización con frecuencia uniforme [Liu et al. 2002].

2.3.3 K-vecinos más cercanos (KNN)

Método de clasificación supervisada que permite estimar una función de clasificación partiendo de los atributos predictores. La información para estimar la función se obtiene del conjunto de los k vecinos más cercanos. Cada ejemplo de la colección es un punto en un espacio multidimensional de los atributos descriptores. Utilizando una medida de distancia se estima cuan cerca se encuentra el ejemplo actual del resto, tomando entonces las clases a las que pertenecen los k más cercanos se tiene un voto para asignar la clase del ejemplo en cuestión. Se utiliza generalmente la distancia euclidiana.

Este modelo se puede configurar en KEEL con los siguientes parámetros:

- *k*: número de vecinos a encuestar.
- *Distance function*: se implementan 3 posibilidades en KEEL:
    - *Euclidean*: con atributos normalizados.
    - *Heterogeneous Value Difference Metric o HVDM* [Wilson and Martinez 2000]: esta utiliza la distancia euclidiana para atributos numéricos y la VDM [Stanfill and Waltz 1986] para atributos nominales.
    - *Manhattan:* la distancia entre dos puntos es el valor absoluto de la suma de las diferencias de las coordenadas de estos.

2.3.4 Máquinas de Soporte Vectorial (SVM)

Realiza la tarea de clasificación por medio de encontrar un hiperplano que maximiza el margen entre dos clases. Los vectores (ejemplos de la colección) que definen el hiperplano son los vectores de soporte. El algoritmo inicia con definir un hiperplano óptimo que maximice el margen entonces los datos son mapeados a un espacio de mayor dimensión, por medio de una función de *kernel*, donde es más fácil encontrar superficies de decisión lineales. En el caso ideal este algoritmo logra generar un hiperplano que separa completamente las clases sin solapamiento. No obstante el caso ideal casi nunca se logra quedando un ligero solapamiento entre las clases que se traduce en errores de clasificación. Por lo anterior el algoritmo lo que busca es encontrar el hiperplano que maximiza el margen y minimiza los errores de clasificación. Para el caso multi-clase SVM utiliza un esquema de votación.

Este modelo se puede configurar en KEEL con los siguientes parámetros:

- *KernellType*: que función de *kernel* utilizar para transformar los datos.
- *C*: parámetro que define el costo del error de clasificación.

- *degree*: grados de libertad de la función de *kernel*.
- *gamma*: parámetro *gamma* de la función de *kernel*.
- *coef0*: parámetro *coef0* en la función de *kernel*.
- *shrinking*: reduce el tamaño del problema de optimización dejando de considerar algunas de las variables.

3 Resultados y discusión

En esta sección se presentan los resultados empíricos del proceso de evaluación del algoritmo LVQ en las tres variantes que se encuentran implementadas en KEEL: LVQ1, LVQ2.1 y LVQ3. Fue utilizada la exactitud predictiva como métrica para la evaluación del desempeño de todos los algoritmos involucrados. Los algoritmos fueron evaluados en las 21 colecciones cuyas características se presentan en el acápite 3.1. Se siguieron las recomendaciones de Demšar [Demšar 2006] para realizar el proceso de evaluación estadística.

3.1 Colecciones de entrenamiento utilizadas

Para acometer la experimentación y evaluar el desempeño del algoritmo LVQ se escogieron 21 colecciones de entrenamiento del repositorio de KEEL [Alcalá-Fdez et al. 2011]. Las colecciones utilizadas fueron clasificadas por los desarrolladores de KEEL en problemas de: regresión, agrupamiento, multi-instancia, clasificación con desbalance, clasificación multi-etiqueta, clasificación clásica y otras subcategorías. Los experimentos se llevaron a cabo con las siguientes colecciones: *appendicitis*, *australian*, *automobile*, *balance*, *banana*, *bands*, *breast*, *bupa*, *ecoli*, *glass*, *heart*, *hepatitis*, *ionosphere*, *iris*, *lymphography*, *pima*, *sonar*, *wdbc*, *wine*, *wisconsin* y *zoo*. Todas ellas son colecciones creadas partiendo de datos del mundo real de distintos dominios. La medicina y la agricultura son solo dos de los dominios representados en estas colecciones de datos. En la (Tabla 1) se puede encontrar un resumen de las características de cada una de ellas.

Colección	Ejemplos	Atributos	Clases
appendicitis	106	9	2
australian	690	14	2
automobile	159	25	6
balance	625	4	3
banana	5300	2	2
bands	365	19	2
breast	277	9	2
bupa	345	6	2
ecoli	336	7	8
glass	214	9	7

heart	270	13	2
hepatitis	80	19	2
ionosphere	351	33	2
iris	150	4	3
lymphography	148	18	4
pima	768	8	2
sonar	208	60	2
wdbc	569	30	2
wine	178	13	3
wisconsin	683	9	2
zoo	101	17	7

Tabla 1. Características de las colecciones de entrenamiento.

3.2 Herramientas utilizadas para los experimentos

Se ejecutaron todas las simulaciones con la herramienta KEEL [Alcalá-Fdez et al. 2011], los experimentos se ejecutaron en un procesador Dual Core AMD-300 a 1.3GHz, 4GB de memoria RAM y Sistema Operativo Windows 8. En general se utilizaron las configuraciones por defecto de KEEL excepto en el caso del algoritmo KNN en el cual se utilizó $k = 5$ y en los algoritmos LVQ donde se utilizaron una cantidad de iteraciones de 500.

3.3 Resultados experimentales y análisis estadístico

Para la experimentación se utilizó el procedimiento de validación cruzada con 10 particiones y además 10 ejecuciones con semillas aleatorias distintas cada vez, para un total de 100 experimentos (en cada par algoritmo-colección), los resultados de la métrica exactitud predictiva (cantidad de ejemplos clasificados correctamente sobre el total de ejemplos) fueron promediados para medir la competitividad de cada algoritmo en cada colección de datos. Los resultados obtenidos se muestran en la (Tabla 2).

Colección	*LVQ1*	*LVQ2.1*	*LVQ3*	*NB*	*C45*	*KNN*	*SVM*
bupa	0.5851	0.5930	0.5587	0.6116	0.6700	0.6131	0.7014
ecoli	0.7832	0.7801	0.7794	0.8156	0.7947	0.8127	0.7592
glass	0.6521	0.6532	0.6447	0.6928	0.6744	0.6685	0.6259
iris	0.9467	0.9533	0.9533	0.9467	0.9600	0.9600	0.9667
pima	0.7047	0.7228	0.6721	0.7488	0.7423	0.7306	0.7710
wine	0.9663	0.9546	0.9663	0.9605	0.9490	0.9605	0.9438
appendicitis	0.8518	0.8609	0.8136	0.8427	0.8327	0.8591	0.8791
australian	0.8217	0.8261	0.8246	0.8667	0.8522	0.8478	0.8580
automobile	0.6193	0.6466	0.5805	0.6795	0.8093	0.5654	0.6197

balance	0.8110	0.7954	0.7952	0.9120	0.7680	0.8624	0.9168
bands	0.6632	0.6907	0.7026	0.7050	0.6499	0.6846	0.6946
breast	0.6382	0.7230	0.6467	0.7440	0.7692	0.7228	0.7075
heart	0.7778	0.7667	0.7296	0.8222	0.7815	0.8074	0.8444
hepatitis	0.8217	0.8617	0.8059	0.8483	0.8400	0.8627	0.8356
ionosphere	0.8603	0.8717	0.8347	0.8890	0.9090	0.8517	0.8803
lymphography	0.7574	0.8195	0.7525	0.8576	0.7430	0.7944	0.8398
sonar	0.8114	0.8171	0.8024	0.7738	0.7007	0.8310	0.7726
wdbc	0.9507	0.9507	0.9472	0.9438	0.9455	0.9683	0.9437
zoo	0.9347	0.9197	0.9206	0.9447	0.9281	0.9364	0.9650
wisconsin	0.9520	0.9666	0.9548	0.9767	0.9563	0.9695	0.9651
banana	0.8677	0.8675	0.8700	0.7117	0.8908	0.8911	0.5517
Ranking Promedio	4.9286	4.0476	5.4761	2.8095	3.9762	3.1429	3.6190

Tabla 2. Exactitud predictiva de cada algoritmo por cada colección.

Para calcular los *rankings* de cada algoritmo se utilizaron nuevamente las herramientas que provee KEEL, los resultados obtenidos pueden verse en la última fila de la (Tabla 2). Siguiendo las recomendaciones de Demšar [Demšar 2006], se utilizó la prueba de Friedman [Friedman 1937; Friedman 1940] con la cual se rechaza la hipótesis nula de que todos los algoritmos se comportan de manera similar con un p-valor = $5.1441*10^{-4}$. Entonces se procedió a realizar la prueba *post-hoc* de Nemenyi [Nemenyi 1963] con las correcciones de Bonferroni-Dunn [Dunn 1961] para determinar la diferencia crítica y establecer diferencias entre los algoritmos estudiados. Usando un valor de $\alpha = 0.5$ se obtiene un $q_\alpha = 2.638$. La diferencia crítica se calcula utilizando la siguiente ecuación:

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}} \quad (12)$$

donde k es el número de algoritmos bajo comparación y N es el número de colecciones de entrenamiento se obtiene una diferencia crítica de CD = 1.7587.

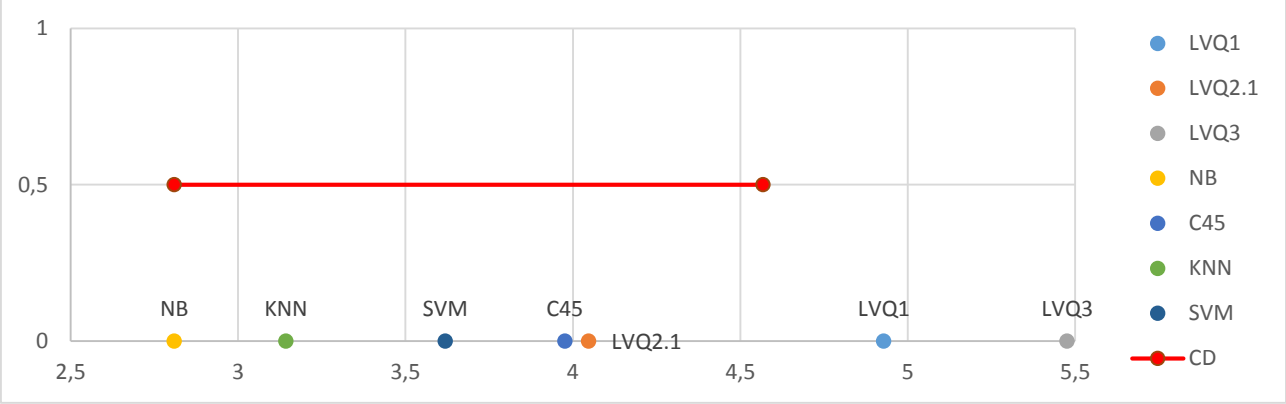

Figura 2. Comparación del clasificador NB contra el resto por medio de la diferencia crítica, prueba de Bonferroni-Dunn.

Como puede observarse en la (Figura 2) los algoritmos LVQ1 y LVQ3 se comportan peor que el mejor de los algoritmos el NB, sin embargo no existe diferencia significativa entre NB y la versión LVQ2.1.

4 Conclusiones y trabajos futuros

En este trabajo se evaluó la competitividad del algoritmo LVQ en las tres versiones que se encuentran implementadas en la herramienta KEEL, fue utilizada la exactitud predictiva como métrica de evaluación obteniéndose como resultado, tras una experimentación sobre 21 colecciones de datos creadas de problemas del mundo real, un desempeño competitivo en el caso de la versión LVQ2.1. Dicha evaluación se realizó frente a cuatro de los algoritmos que se encuentran en el Top-ten [Wu et al. 2007] mundial para clasificación de información. Como parte de este trabajo se vislumbra que puede seguir profundizándose en la evaluación de las distintas versiones del LVQ, en particular puede evaluarse el desempeño de la versión OLVQ1 que actualmente no se encuentra implementada en KEEL, por otro lado puede comprobarse el desempeño de estos en colecciones de datos ruidosas o con valores omitidos, las cuales están disponibles en la colección que brinda la excelente herramienta KEEL.

Referencias

AHN, K.K. AND NGUYEN, H.T.C., 2007. Intelligent switching control of a pneumatic muscle robot arm using learning vector quantization neural network. *Mechatronics*, 17(4), pp.255–262.

ALCALÁ-FDEZ, J. ET AL., 2008. KEEL : a software tool to assess evolutionary algorithms for data mining problems. *Soft Computing*, 13(3), pp.307–318.

ALCALÁ-FDEZ, J. ET AL., 2011. KEEL Data-Mining Software Tool: Data Set Repository and Integration of Algorithms and Experimental Analysis Framework. *Journal of Multiple-Valued Logic and Soft Computing*, 17(2-3), pp.255–287.

ANAGNOSTOPOULOS, C., ANAGNOSTOPOULOS, J., VERGADOS, D., KAYAFAS, E., LOUMOS, V. AND THEODOROPOULOS, G., 2001. Training a learning vector quantization network for biomedical classification. In *Proceedings of the international joint conference on neural networks*. National Technical University of Athens (NTUA), pp. 2506–2511.

BASHYAL, S. AND VENAYAGAMOORTHY, G.K., 2008. Recognition of facial expressions using gabor wavelets and learning vector quantization. *Engineering Applications of Artificial Intelligence*, 21(7), pp.1056–1064.

BASSIUNY, A., LI, X. AND DU, R., 2007. Fault diagnosis of stamping process based on empirical mode decomposition and learning vector quantization. *Int J Mach Tools Manuf*, 47(15), pp.2298–2306.

BAYES, T., 1763. An Essay towards solving a Problem in the Doctrine of Chances. *Philosophical Transactions of the Royal Society of London*, 53, pp.370–418.

BEZDEK, J.C. AND PAL, N.R., 1995. Two soft relatives of learning vector. *Neural Networks*, 8(5), pp.729–743.

BLUME, M. AND BALLARD, D.R., 2007. Image annotation based on learning vector quantization and localized Haar wavelet transform features. In S. K. Rogers, ed. *Society of photo-optical instrumentation engineers*. pp. 181–190.

CHEN, C.Y., 2012. Accelerometer-based hand gesture recognition using fuzzy learning vector quantization. *Adv Sci Lett*, 9(1), pp.38–44.

CORTES, C. AND VAPNIK, V., 1995. Support-Vector Networks. *Machine Learning*, 20, pp.273–297.

COVER, T.M. AND HART, P.E., 1967. Nearest Neighbor. *IEEE Transactions on Information Theory*, IT-13(1), pp.21–27.

DEMŠAR, J., 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. *Machine Learning Research*, 7, pp.1–30.

DIETERLE, F., MULLER-HAGEDORN, S., LIEBICH, H.M. AND GAUGLITZ, G., 2003. Urinary nucleosides as potential tumor markers evaluated by learning vector quantization. *Artificial Intelligence in Medicine*, 28(3), pp.265–280.

DOMINGOS, P. AND PAZZANI, M., 1997. On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning*, 1997(29), pp.103–130.

DUNN, O.J., 1961. Multiple comparisons among means. *Journal of the American Statistical Association*, (56), pp.52–64.

DUTTA, S., CHATTERJEE, A. AND MUNSHI, S., 2001. Identification of ecg beats from cross-spectrum information aided learning vector quantization. *Measurement*, 44(10), pp.2020–2027.

FIX, E. AND HODGES, J.L., 1951. An Important Contribution to Nonparametric Discriminant Analysis and Density Estimation. *International Statistical Review*, 3(57), pp.233–238.

FRIEDMAN, M., 1940. A comparison of alternative tests of significance for the problem of m rankings. *Annals of Mathematical Statistics*, (11), pp.86–92.

FRIEDMAN, M., 1937. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, (32), pp.675–701.

HAMMER, B., MOKBEL, B., SCHLEIF, F.M. AND ZHU, X., 2011. Prototypebased classification of dissimilarity data. *Lecture Notes in Computer Science*, 7014, pp.185–197.

HAMMER, B. AND VILLMANN, T., 2002. Generalized relevance learning vector quantization. *Neural Networks*, 15(8-9), pp.1059–1068.

JOHN, G. AND LANGLEY, P., 1995. Estimating Continuous Distributions in Bayesian Classifiers. In *In Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, pp. 338–345.

KARAYIANNIS, N.B., 1997. A methodology for constructing fuzzy algorithms for learning vector quantization. *IEEE Transactions on Neural Networks*, 8(3), pp.505–518.

KARAYIANNIS, N.B., 1999. An axiomatic approach to soft learning vector quantization and clustering. *IEEE Transactions on Neural Networks*, 10(5), pp.1153–1165.

KARAYIANNIS, N.B. AND PAI, P.I., 1996. Fuzzy algorithms for learning vector quantization. *IEEE Transactions on Neural Networks*, 7(5), pp.1196–1211.

KOHONEN, T., 1988. An introduction to neural computing. *Neural networks : the official journal of the International Neural Network Society*, (1), pp.3–16.

KOHONEN, T., 2001. *Self-Organizing Maps* Third Exte. Springer-Verlag, ed., Berlin, Heidelberg.

KOHONEN, T., 1990. The self organizing map. *Proceeding of the IEEE*, 78(9), pp.1464–1480.

LIU, H., HUSSAIN, F., LIM, C. AND DASH, M., 2002. Discretization: An Enabling Technique. *Data Mining and Knowledge Discovery*, 6(4), pp.393–423.

MARON, M.E., 1961. Automatic Indexing : An Experimental Inquiry. *Journal of the ACM (JACM)*, 8:3(January), pp.404–417.

MCCLELLAND, J.L., RUMELHART, D.E. AND HINTON, G.E., 1987. The Appeal of Parallel Distributed Processing. In D. E. Rumelhart, J. L. McClelland, & others, eds. *Parallel Distributed Processing: Volume 1: Foundations*. Cambridge: MIT Press, pp. 3–44.

NEMENYI, P.B., 1963. *Distribution-free multiple comparisons*. Princeton University.

NEURAL NETWORKS RESEARCH CENTRE, 2007. Bibliography on the self-organizing map (som) and learning vector quantization (lvq).

PAL, N.R., BEZDEK, J.C. AND TSAO, E.K., 1993. Generalized clustering networks and kohonen's self-organizing scheme. *IEEE Transactions on Neural Networks*, 4(4), pp.549–557.

QIN, A.K. AND SUGANTHAN, P., 2004. A novel kernel prototype-based learning algorithm. *Pattern Recognition*, 4, pp.621–624.

QUINLAN, J.R., 1993. *C4.5: Programs for Machine Learning*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

QUINLAN, J.R., 1986. Induction of Decision Trees. *Machine Learning*, 1, pp.81–106.

RISSANEN, J., 1978. Modeling by shortest data description. *Automatica*, 14(5), pp.465–471.

SCHNEIDER, P., BIEHL, M. AND HAMMER, B., 2009. Distance learning in discriminative vector quantization. *Neural Computation*, 21(10), pp.2942–2969.

STANFILL, C. AND WALTZ, D., 1986. Instance-based Learning Algorithms. *Communications of the ACM*, 12, pp.1213–1228.

WANG, J., NESKOVIC, P. AND COOPER, L.N., 2007. Improving nearest neighbor rule with a simple adaptive distance measure. *Pattern Recognition Letters*, 28, pp.207–213.

WILSON, D.R. AND MARTINEZ, T.R., 2000. Reduction Tecniques For Instance-Based Learning Algorithms. *Machine Learning*, 38:3, pp.257–286.

WU, X. ET AL., 2007. Top 10 algorithms in data mining. *Knowledge Information Systems*, (2008)(14), pp.1–37.