Reconocimiento de Patrones
Trabajo Final - Reconocimiento de LETRAS

Mario González Olmedo

8 de diciembre de 2014

Abstract

En el presente informe, se plantea el problema propuesto sobre clasificación de caracteres, y la solución implementada en MATLAB utilizando principalmente las técnicas de clasificación de vecinos más cercanos (k-NN) vistas en el curso, elegido a partir de resultados de evaluaciones de clasificación realizadas inicialmente en Weka.

Se estudia además una variante llamada *fuzzy k-NN*, y a partir de esta, se propone una modificación adaptada para el caso en que los caracteres a clasificar provienen del idioma español, incorporando frecuencias de aparición de letras (priors) para mejorar el desempeño del clasificador.

Contents

1 Descripción del proyecto

1.1 Objetivo

El problema consistirá en identificar letras a partir de un gran número de datos basados en imágenes en blanco y negro de 26 letras mayúsculas del alfabeto (ver Figura 1). Se dispondrá de un conjunto de muestras de entrenamiento (con características dadas, previamente extraídas) con el cual se diseñará el clasificador, y luego se evaluará su desempeño con un conjunto de muestras test.

Los objetivos específicos del proyecto son:

1. Se desea lograr un sistema que sea capaz de clasificar correctamente, y en forma automática, a qué letra corresponde cada vector de características. El requerimiento mínimo a alcanzar es el de lograr que la media aritmética de las tasas de acierto por clase esté por encima de 90%. Este requisito se impone para un sorteo de los vectores de test, asumiendo equiprobabilidad entre las clases.

2. Se desea construir un sistema para transcribir un texto en español (tomando la Ñ como una N) a partir de los vector de características de imágenes provenientes de las letras de un texto. Se establece como nuevo objetivo que el sistema maximice su tasa de acierto, conociendo que el conjunto de testeo proviene de un texto en idioma español.

3. Comparar y comentar los resultados de ambos sistemas.

1.2 Enfoque

Se evaluará la utilización de técnicas de selección y extracción de características vistas en el curso para reducir la dimensionalidad del espacio de características, si fuera necesario. Para el primer objetivo, debido a que se dispone de muestras etiquetadas, se estudiará qué herramientas del aprendizaje supervisado son más eficaces para encarar el problema (técnicas de clasificación paramétricas y no paramétricas).

Para el segundo objetivo, se intentará introducir información extra del problema concreto a resolver (información a priori) para mejorar el rendimiento del clasificador. En particular, se tratará de incorporar al clasificador a desarrollar, información sobre las frecuencias (probabilidades) de aparición de las letras en un texto en idioma español.

1.3 Alcance

Las imágenes a ser utilizadas en el proyecto se generaron para 20 tipos de letra diferente y fueron aleatoriamente distorsionadas para generar 20000 imágenes (ver Figura 1). A los efectos del proyecto, no se trabajará con las imágenes sino con un conjunto de 16 características extraídas a partir de esas imágenes.

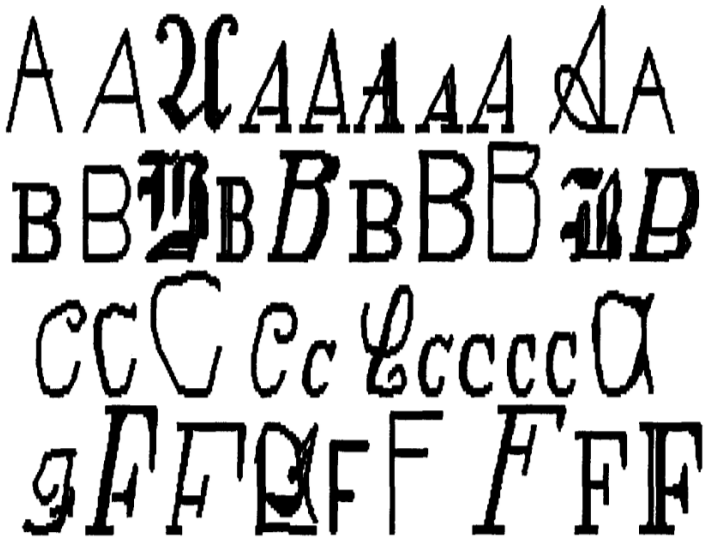

Figure 1: Ejemplos de las muestras del problema

1.4 Datos

Se dispone de un conjunto de muestras de entrenamiento previamente etiquetadas en la forma (vector de características, clase). También se dispondrá de un conjunto de muestras de prueba para la evaluación final del algoritmo, el cual será entregado poco antes de la evaluación final. La validación del desempeño de los modelos será realizada en el trabajo, en base al particionado de los datos de entrenamiento.

El conjunto de datos de entrenamiento consistirá en 16000 elementos. El conjunto de test consistirá en 4000 elementos extraídos de un texto en español (tomando la Ñ como una N). Las características son normalizadas en un rango que varía entre 0 y 15, siendo representadas con números enteros. A efectos del entendimiento general del problema, se describe brevemente en qué consiste cada una de las características calculadas sobre las imágenes de las letras. Por más detalles se puede consultar [3].

1. x-box posición horizontal del recuadro (int)
2. y-box posición vertical del recuadro (int)
3. width ancho del recuadro (int)
4. high altura del recuadro (int)
5. onpix número total de pixels encendidos (int)
6. x-bar media de x de pixels encendidos dentro del recuadro (int)
7. y-bar media de y de pixels encendidos dentro del recuadro (int)
8. x2bar varianza de x (int)
9. y2bar varianza de y (int)
10. xybar correlación entre x e y (int)

11. x2ybr valor medio de x * x * y (int)
12. xy2br valor medio x * y * y (int)
13. x-ege conteo medio de bordes horizontales izq a der (int)
14. xegvy correlación de bordes horizontales con y (int)
15. y-ege conteo medio de bordes verticales abajo a arriba (int)
16. yegvx correlación de bordes verticales con x (int)
17. clase letras mayúsculas (26 de la A a la Z)

1.5 Software

Se trabajará fundamentalmente con MATLAB para la implementación de los algoritmos a ser utilizados por el clasificador. Asimismo, se utilizará el software Weka, disponible en

http://www.cs.waikato.ac.nz/ml/weka

para una familiarización inicial de los datos del problema.

2 Trabajo realizado

2.1 Exploración de los datos

Inicialmente, se exploraron los datos en Weka para alcanzar una familiarización inicial con los mismos. A partir de ello puede observarse a primera vista, según la distribución de las características, que algunas de ellas sirven para discriminar bien ciertas clases de otras, mientras que no distinguen adecuadamente otras clases (ver Figura 2).

En general, se observa que no hay características que logren buena discriminación de todas las clases del problema al mismo tiempo. Esto se debe, quizás, a que las características fueron previamente bien elegidas, lo cual motiva a no poner mucho énfasis en técnicas de selección de características en una primera instancia. Además, como se verá después, el esfuerzo puesto en la optimización de los algoritmos hace que la dimensionalidad del espacio de características no sea demasiado elevado como para que los tiempos de ejecución fueran prohibitivos.

Para una primera aproximación al problema de clasificación, se aplicaron distintos clasificadores del software Weka, utilizando validación cruzada (10 partes) para evaluar desempeño, y los parámetros por defecto de cada clasificador. Se obtuvieron los siguientes porcentajes de aciertos:

Clasificador	% Aciertos
NaiveBayes	64.04
C4.5	86.88
1-NN	95.45
3-NN	95.11
SVM (lineal)	81.69
SVM (RBF)	88.37

A raíz de estos datos, se decide implementar un algoritmo de clasificación basado en **vecinos más cercanos** (k-NN).

2.2 Preprocesamiento de los datos

Luego de una primera aproximación a los datos del problema, se procedió a la implementación en MATLAB. Para ello, debido a que los datos se encuentran en un formato que no es nativo de MATLAB (format .arff), se convirtieron los mismos a formato .csv, que puede utilizarse fácilmente. Se aprovechó este proceso además para convertir la última columna de los datos (etiquetas de los patrones) de letras a números (del 1 al 26), que pueden usarse para trabajar mejor (por ejemplo, al indexar) en las funciones luego desarrolladas.

Por otra parte, como el clasificador implementado es de la familia de los k-NN, se desarrollaron scripts[1] para particionar los datos en subconjuntos más pequeños, y de esta forma, al clasificar un patrón dado, no se tenga que medir las distancias a cada uno de los patrones de entrenamiento, según se describe en [2], 4.5.5. Así, podremos más

[1] `particion_optima.m, particionar.m, patrones_part.m`

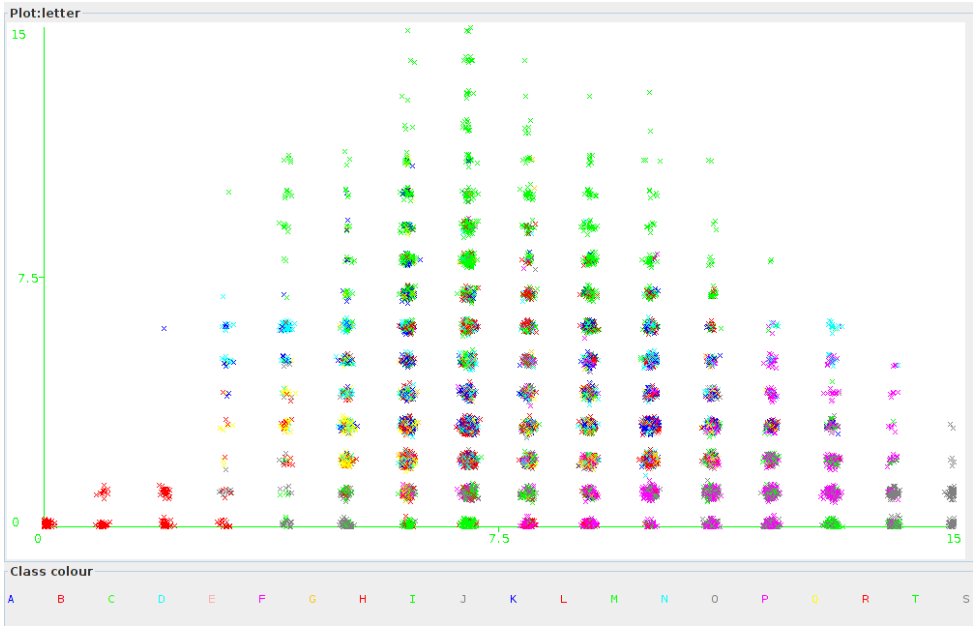

Figure 2: Visualización de las características

adelante optimizar los tiempos de ejecución del clasificador cuando es sometido a grandes cantidades de datos. Lo que logran dichos scripts es particionar el conjunto de patrones dados en 4 subconjuntos, eligiendo 2 características y 2 umbrales respectivos, y tratar de que la distribución sea lo más uniforme posible (ver Figura 3).

2.3 Primeros resultados con k-NN

A raíz de los resultados obtenidos en Weka, se implementó 1-NN en MATLAB, inicialmente con cálculo total de las distancias[2]. Se consideraron varias normas para la medición de distancias (norma 1, norma 2, norma infinito), y la que mejor desempeño mostró fue la norma 1 (a partir de aquí se utilizará dicha norma).

Debido a que la clasificación por k-NN puede ser muy lenta si la cantidad de datos es grande, se puso empeño en lograr algoritmos rápidos de clasificación, sin sacrificar significativamente el desempeño de los mismos. En este sentido, primero se modificó la función anteriormente mencionada para lograr otra[3] que utilice cálculos parciales de distancias, según se detalla en [2], 4.5.5. Y además de esto, acorde a lo descrito en la sección anterior, se implementó una tercera función[4] en la cual además se clasifican los patrones test usando una partición generada sobre los datos de entrenamiento.

Se implementó también una función que arma la matriz de confusión[5], para poder visualizar los errores que comete el clasificador con mayor detalle.

[2] `k1_vecino.m`
[3] `k1_vecino_opt.m`
[4] `k1_vecino_opt2.m`
[5] `matriz_confusion.m`

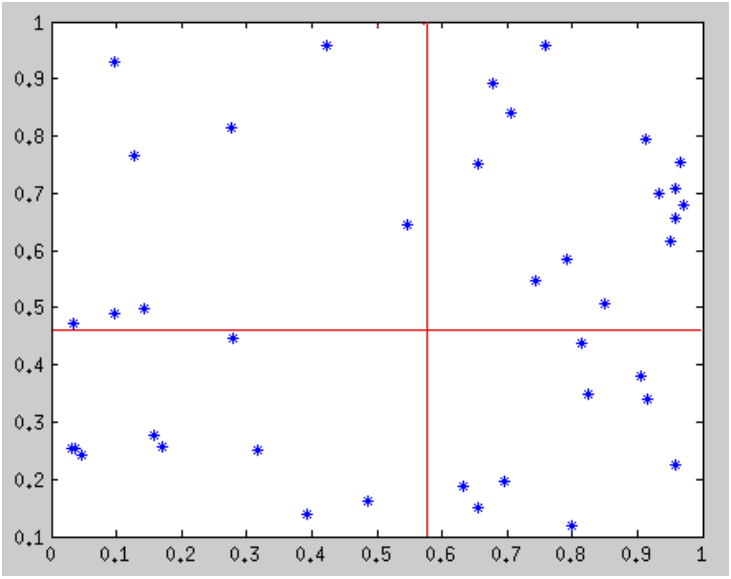

Figure 3: Ejemplo de partición en dos variables

Para comparar estas tres funciones, se evaluó[6] el desempeño tomando, de las 16000 muestras disponibles, un 80% (12800 muestras) para entrenamiento y 20% (3200 muestras) para test. Los resultados[7] que se obtuvieron son los siguientes[8]:

Función	% Aciertos	Tiempo (seg)
`k1_vecino.m`	94.00	81
`k1_vecino_opt.m`	94.09	16
`k1_vecino_opt2.m`	93.44	6

Vemos que ya utilizando cálculo parcial de distancias se logra reducir el tiempo de ejecución a la quinta parte. Si además se utilizan las particiones, se logra reducir aún más el tiempo de ejecución, aunque en este caso con un pequeño sacrificio de rendimiento.

Esta optimización será valiosa más adelante cuando le requeriremos un poco más de cálculo a nuestro clasificador, para que el tiempo de ejecución no se haga excesivo. Comprobamos de esta manera además, que el rendimiento de nuestro algoritmo es similar al reportado por Weka (ambos por encima al 93%).

Más adelante, se extendió el clasificador para considerar k vecinos: una versión sin utilizar particiones[9] y otra versión usando particiones[10] (ambas usan cálculo parcial de distancias). La evaluación[11] de estas nuevas funciones, para varios valores de k, arroja los siguientes resultados:

[6] `evaluacion1.m`
[7] `resultados/resultados_evaluacion1.m`
[8] En todas las ejecuciones, la máquina utilizada fue un Core i3, 4 GB de RAM
[9] `kk_vecino.m`
[10] `kk_vecino2.m`
[11] `evaluacion2.m`

kk_vecino.m	% Aciertos	Tiempo (seg)
$k = 2$	92.59	175
$k = 3$	94.25	185
$k = 4$	93.86	193
$k = 5$	94.09	196

kk_vecino2.m	% Aciertos	Tiempo (seg)
$k = 2$	91.38	53
$k = 3$	93.00	55
$k = 4$	92.03	57
$k = 5$	92.18	59

2.4 Fuzzy k-NN

Luego se estudió una variante de k-NN llamado *fuzzy k-NN*, siguiendo [4], la cual se explica a continuación.

La idea principal en la clasificación clásica de k-NN (la considerada anteriormente) es la siguiente: dado un patrón a clasificar, al considerar los k vecinos más cercanos de entre los patrones de entrenamiento disponibles, cada uno "vota" a su clase para, al final de la votación, clasificar a dicho patrón como perteneciente a la clase más votada. En este caso, todos los votos de los k vecinos tienen el mismo peso.

En la variante que consideraremos a partir de ahora, ponderamos el voto de cada uno de los k vecinos más cercanos según a qué distancia se encuentren del patrón a clasificar. Concretamente, dado un patrón x a clasificar, y suponiendo que $x_1, \ldots, x_k$ son los k vecinos más cercanos a x, entonces cada vecino x_j aporta un voto equivalente a

$$u_j(x) = \frac{1/\|x - x_j\|^2}{\sum_{i=1}^{k} 1/\|x - x_i\|^2} \tag{1}$$

De esta forma, los vecinos más cercanos al patrón x (más "parecidos", según la métrica utilizada) tienen un voto mayor que los vecinos más lejanos (menos "parecidos"). El denominador se incorpora para que la totalidad de votos de los k vecinos sea 1. Notar que en este caso, fuzzy 1-NN coincide con 1-NN clásico, así que recién para $k \geq 2$ esta variante tiene sentido.

Luego de realizada la votación de los k vecinos, podemos seleccionar la clase más votada según estos pesos. Además, contamos con una medida de seguridad sobre la clasificación de dicho patrón (pertenencia difusa a la clase arrojada por el clasificador), ya que a mayor voto de la clase seleccionada (es decir, suma de votos cercano a 1) habrá mayor seguridad de clasificar correctamente, y si tenemos por ejemplo dos clases con votaciones 0.45 y 0.55, tendremos más incertidumbre y podremos entonces incorporar alguna otra información para ayudar a la clasificación (por ejemplo, información a priori, clasificación de otros clasificadores, etc).

Se incluye la implementación de la función[12] correspondiente a esta variante, ya in-

[12] fuzzy_kk.m

cluyendo todas las optimizaciones antes mencionadas para mejorar tiempos de ejecución (notar que esta nueva función requiere más computaciones que las anteriores).

Se clasificaron[13] los patrones test antes considerados, ahora con este nuevo clasificador (con votos ponderados) para varios valores de k (cantidad de vecinos), y se obtuvieron los siguientes resultados (sobre las mismas muestras de entrenamiento y test anteriores):

`fuzzy_kk`	% Aciertos	Tiempo (seg)
$k = 2$	93.47	52
$k = 3$	93.84	55
$k = 4$	93.84	58
$k = 5$	93.59	59

Puede verse que este nuevo clasificador obtiene mejores resultados al considerar más vecinos que en el caso k-NN clásico, que a su vez constituyen una mejora con respecto a los resultados obtenidos anteriormente.

Además de los resultados de clasificación, esta función fue diseñada para que devuelva una matriz con las distancias totales por clase en la votación interna, para cada patrón. Con esta información, podemos revisar qué tipo de errores se cometió: cuántos de ellos fueron en aquellos patrones en donde la seguridad del clasificador es alta (suma de votos cercano a uno) o en los demás, así como cuántos patrones se clasificaron correctamente, pero con poca seguridad.

El análisis de esta matriz nos proporciona la siguiente tabla (3200 patrones clasificados usando 12800 como patrones de entrenamiento):

`fuzzy_kk`	$k = 2$	$k = 3$	$k = 4$	$k = 5$
Errores con seguridad > 0.75	57	19	34	23
Aciertos con seguridad < 0.75	191	333	282	362

Concluímos esta sección comentando que consideraremos de ahora en más al clasificador fuzzy 3-NN (3 vecinos) como nuestro clasificador fuzzy predilecto, tanto por su desempeño en la clasificación, como por poseer la mínima cantidad de patrones errores en aquellos patrones en los que estaba considerablemente seguro (en nuestro análisis, por encima del 0.75% de seguridad).

En la siguiente sección trataremos de ayudar al clasificador fuzzy en aquellos casos donde la seguridad es baja, incorporando información a priori de clases.

2.5 Incorporando priors

Otra ventaja de las pertenencias difusas de la variante fuzzy k-NN, es que pueden incorporarse fácilmente algunos tipos de información a priori, como por ejemplo la frecuencia de aparición de las letras, es decir, la probabilidad a priori de cada una de las clases del problema considerado.

[13] `evaluacion3.m`

Proponemos a su vez una variante del clasificador anterior, en la cual ponderamos los votos antes considerados y sustituímos el voto de la ecuación (1) por:

$$u_j(x) = \frac{\alpha_j / \|x - x_j\|^2}{\sum_{i=1}^{k} \alpha_i / \|x - x_i\|^2} \tag{2}$$

donde $\alpha_j = \alpha + (1-\alpha)P(\omega_j)$ con $\alpha \in [0,1]$, y $P(\omega_j)$ es la probabilidad a priori de la clase de x_j.

La motivación para considerar esta variante es la siguiente: usando la ecuación (1), para cada vecino se utilizaban votos que dependían de la distancia al patrón a clasificar, sin considerar la probabilidad de la clase de dicho vecino. En la nueva variante, dicho voto se pondera por el factor $\alpha_j = \alpha + (1-\alpha)P(\omega_j)$ que consiste en un valor base α más otro que depende de la probabilidad de la clase del vecino de donde proviene el voto.

Así, un vecino de una clase muy probable ($P(\omega_j) \simeq 1$) aportará el voto completo (según su distancia al patrón), un vecino de una clase poco probable ($P(\omega_j) \simeq 0$) aportará sólo el voto base, y los votos de vecinos de clases con probabilidades intermedias se ponderan proporcionalmente.

El parámetro α modera la importancia que le damos a las priors. Por ejemplo, el valor $\alpha = 1$ nos lleva al caso anterior (ecuación (1)), o sea, no le damos importancia a las priors. El valor $\alpha = 0$ pondera completamente el voto por las priors, y valores intermedios ponderan proporcionalmente (dan una base α al voto, y el resto según priors).

El efecto que producirá esta ponderación es la de desequilibrar la balanza para las clases más probables cuando la seguridad del clasificador es baja (varias clases con pertenencias difusas similares), y para patrones donde el clasificador ya tiene una seguridad casi completa, esta ponderación (para ciertos valores del parámetro α a determinar) no influye demasiado en el resultado de la clasificación.

Se incluye la implementación de la función que clasifica según esta nueva variante[14]. En la siguiente sección, describiremos cómo se generaron patrones provenientes de un texto en español para poner a prueba este último clasificador.

2.6 Resultados sobre texto en español

Como probabilidades a priori de las clases (frecuencias de letras en el idioma español), se tomaron los datos que se encuentran en Wikipedia:

http://es.wikipedia.org/wiki/Frecuencia_de_aparici%C3%B3n_de_letras

Luego se tomó un texto en español al cual se le quitaron los signos de puntuación y espacios (y se cambiaron las Ñ por N), y a partir de eso se tomaron, de entre las muestras dadas, vectores de características aleatorios, uno correspondiente a cada letra del texto mencionado, obteniendo así un total de 1573 vectores de características test a clasificar.

[14] `fuzzy_kk_priors.m`

Luego se utilizaron otros 12800 patrones (diferentes a los utilizados para generar las muestras del texto en español) como muestras de entrenamiento.

Para estimar el valor óptimo del parámetro α de la ecuación (2), conjuntamente con la cantidad de vecinos k a considerar, se utilizó una heurística *grid-search*[15] sobre los datos antes mencionados. Se obtuvieron los siguientes resultados de clasificación:

	$\alpha = 0.60$	$\alpha = 0.65$	$\alpha = 0.70$	$\alpha = 0.75$
$k = 2$	92,75	92,75	92,75	92,75
$k = 3$	93,51	93,58	**93,64**	**93,70**
$k = 4$	93,64	93,64	93,45	92,88
$k = 5$	93,45	93,58	93,20	92,69

Marcados están los dos mejores resultados, que se dan para $k = 3$ vecinos, y parámetro $\alpha = 0.70$ y $\alpha = 0.75$.

Para comparar el desempeño del nuevo clasificador utilizando estos parámetros, se clasificaron los mismos patrones con el clasificador anterior fuzzy k-NN (ecuación (1). Los resultados obtenidos son los siguientes (se marcan en negro los mejores resultados por clasificador):

k	fuzzy k-NN	c/priors, $\alpha = 0.70$	c/priors, $\alpha = 0.75$
2	**89.83**	—	—
3	88.74	**93,64**	**93,70**
4	88.37	—	—

Se observa que la incorporación de las priors constituye una mejora significativa del desempeño del clasificador. Por lo tanto, nuestra solución para el problema de maximizar la tasa de aciertos sabiendo que el texto proviene del idioma español, es el modelo antes descrito (fuzzy k-NN con priors), utilizando los parámetros $k = 3$ y $\alpha = 0.75$.

[15] `grid_search.m`

3 Resultados finales y conclusiones

Para finalizar, evaluaremos el desempeño de los tres clasificadores anteriormente descritos (k-NN clásico, fuzzy k-NN y fuzzy k-NN con priors) sobre los dos juegos de datos proporcionados para test[16]. Los resultados de dicha evaluación son los siguientes:

- **Primer juego de datos: Letras equiprobables**
  (16000 entrenamiento, 2600 test)

Clasificador	% Aciertos	Tiempo (seg)
1-NN	98.5	4.5
fuzzy 3-NN	98.19	53.7

- **Segundo juego de datos: Texto en español**
  (16000 entrenamiento, 5000 test)

Clasificador	% Aciertos	Tiempo (seg)
1-NN	96.46	9.6
fuzzy 3-NN	96.2	107
fuzzy 3-NN ($\alpha = 0.75$)	97.94	102

Es importante notar que en este caso, el clasificador fuzzy 3-NN no logró un desempeño mayor que el logrado en instancias anteriores. Pero aún así, la variante fuzzy 3-NN con priors logró una mejora significativa dentro de los márgenes posibles de mejora (casi 1.5% mejor), considerando que el rendimiento de los clasificadores estudiados ya de por sí son buenos.

[16] `evaluacion_final.m`

References

[1] Bishop, C. M; Pattern Recognition and Machine Learning, Springer (ISBN-13-9780387310732)-2006

[2] Duda, Hart and Stork; Pattern Classification, John Wiley & Sons (ISBN-10-0471056693)-2001

[3] Frey, Peter W y David J Slate (1991). "Letter Recognition Using Holland-Style Adaptive Classifiers". En: Machine Learning 6, pág. 161.

[4] Keller, J.M.; Gray, M.R.; Givens, J.A., "A fuzzy K-nearest neighbor algorithm," Systems, Man and Cybernetics, IEEE Transactions on , vol.SMC-15, no.4, pp.580,585, July-Aug. 1985