
Práctica 5

Handwritten Digit Recognition

SMO

Componentes del grupo:

- Edgar Andres
- Paula de Jaime
- Maitane Ruiz

Índice

Introducción	3
Marco teórico	5
Documentación: SMO	5
Support Vector Machines	6
Sequential Minimal Optimization	10
Desglose del algoritmo	10
Módulo heurístico	10
Resolución de multiplicadores de Lagrange (técnicas analíticas)	11
Caso de dos multiplicadores	12
Cálculo de la constante b (umbral de error y caché)	13
Diseño	15
Marco experimental	16
Datos	16
Análisis train-digits.arff	16
Análisis test-digits.arff	18
Pre-proceso elegido	19
Filtros probados en train-digits.arff	19
Resultados experimentales	21
Sin aplicar filtros	21
Cuadrícula números 15x15	21
Filtro Resample	21
Ejemplo de ejecución del ejecutable (descritos en la sección 5.1).	22
Conclusiones y trabajo futuro	24
Bibliografía	25
Introducción	25
Marco teórico. SMO	25
Valoración subjetiva	26

Introducción

El Handwritten Digit Recognition (HDR) es un campo importante del Optical Character Recognition (OCR) que consiste en recibir e interpretar dígitos manuscritos de forma automatizada.

En las últimas décadas las investigaciones sobre Handwritten Digit Recognition (HDR) han adquirido gran relevancia en el campo de la inteligencia artificial.

Debido al avance tecnológico, es cada vez menos frecuente encontrar textos manuscritos. Sin embargo, hay ámbitos como los servicios postales, bancos o documentos históricos para los cuales el desarrollo de aplicaciones de procesamiento automática de texto manuscrito es todavía de gran relevancia.

En el HDR, la variabilidad de la letra de cada persona hace que implementar un detector automático que se adapte a cualquier escenario sea un gran reto. De hecho, la letra de una misma persona también varía, y el sistema debe ser suficientemente robusto para clasificar cualquier tipo de texto manuscrito.

En esta práctica se propone una tarea de HDR que utiliza el benchmark digits [\[Seewald, 2005\]](#).

Los autores proporcionaron a un conjunto de estudiantes un documento plantilla en el cual cada estudiante tenía que escribir 10 veces los dígitos 0-9. A continuación, escanearon y procesaron los datos dando lugar, finalmente, a los ficheros ARFF. [\[Enunciado Practica 5\]](#)

En los archivos ARFF anteriormente nombrados, hemos podido comprobar que cada atributo hace referencia a un pixel de los escaneos realizados. En ellos existe un valor numérico que refleja el grado de tinta que se encontraba en esa zona de la escritura, en el caso de haberla.

Mediante un estudio detallado de la referencia bibliográfica que se proporciona en el fichero '*paper.pdf*' que se nos ha facilitado, y después de barajar varias hipótesis entre el grupo de trabajo, determinamos llevar la práctica a cabo de la siguiente forma:

Práctica 5 - Handwritten Digit Recognition

A través de los valores numéricos que contienen los atributos (píxeles) de cada instancia (escaneo), entrenar el modelo para que sea capaz de predecir con un máximo de aciertos posibles la clase de los datos a clasificar.

Gracias al proceso de esta práctica hemos sido capaces de conocer distintos algoritmos en busca de la mayor precisión de previsión posible, donde nuestro mayor porcentaje ha sido de 94,15% utilizando el *SMO*.

Los resultados que vienen documentados y explicados en este informe, han sido realizados mediante el algoritmo anteriormente mencionado, el cual, también viene explicado.

Marco teórico

Documentación: SMO

En los últimos años se ha producido un aumento en el interés de “Support Vector Machines” (SVM). Éstos han demostrado empíricamente un buen desempeño en problemas de reconocimiento de caracteres, de caras, etc. [\[Sequential Minimal Optimization\]](#)

De todas maneras el uso de los SVM está todavía limitado a un grupo pequeño de investigadores debido a su lentitud en problemas grandes. Otra posible razón de su uso limitado puede ser su complejidad, ya que un ingeniero tendría dificultad en implementar estos algoritmos. [\[Sequential Minimal Optimization\]](#)

SMO o bien “Sequential Minimal Optimization”, es un nuevo algoritmo de aprendizaje de SVM creado por John Platt [\[SMO\]](#). Conceptualmente es simple, fácil de implementar, y generalmente más rápido. [\[Sequential Minimal Optimization\]](#)

Práctica 5 - Handwritten Digit Recognition

Support Vector Machines¹

En su más simple forma lineal, SVM es un hiperplano que separa instancias de dos clases distintas maximizando el margen (Fig. 1).

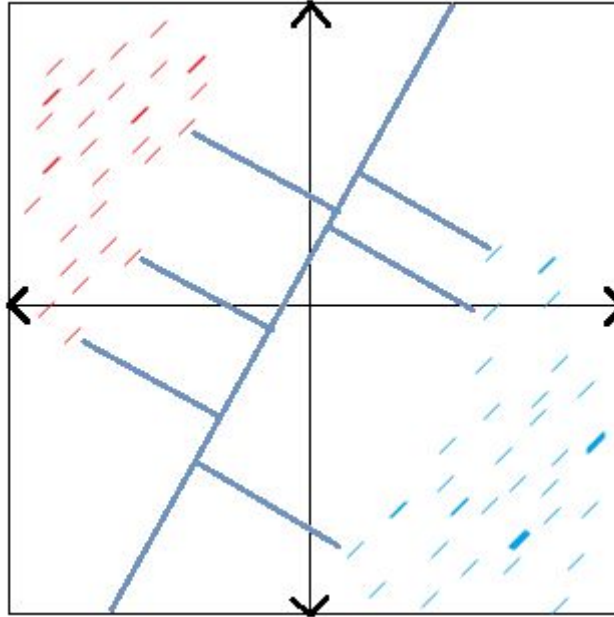


Fig. 1 - Plano bidimensional con instancias de dos clases (rojas y azules).

En el caso lineal, el margen se define como la distancia del hiperplano a la instancia más cercana de cada clase. La fórmula para la salida del SVM lineal es la siguiente:

$$u = w * x - b$$

La w es el vector normal al hiperplano y la x es el vector de entrada (las instancias). El hiperplano constituye $u = 0$, por lo que los puntos más cercanos serán $u = \pm 1$ con respecto al hiperplano.

El margen m entonces será: $m = \frac{1}{\|w\|_2}$

¹[\[Sequential Minimal Optimization\]](#)

Práctica 5 - Handwritten Digit Recognition

Y la maximización de este margen se expresa mediante el siguiente problema de optimización:

$$\min_{\vec{w}, b} \frac{1}{2} \|\vec{w}\|^2 \text{ subject to } y_i (\vec{w} \cdot \vec{x}_i - b) \geq 1, \forall i$$

x_i es una instancia del conjunto de entrenamiento, e y_i es su clase real.

Este problema de optimización puede ser resuelto convirtiéndolo a una forma dual (problema QP = problema cuadrático) usando una función "*Lagrangian*". La función objetivo Ψ es solamente dependiente a una serie de multiplicadores (α_i) Lagrange.

$$\min_{\vec{\alpha}} \Psi(\vec{\alpha}) = \min_{\vec{\alpha}} \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N y_i y_j (\vec{x}_i \cdot \vec{x}_j) \alpha_i \alpha_j - \sum_{i=1}^N \alpha_i$$

$$\alpha_i \geq 0, \forall i, \text{ y } \sum_{i=1}^N y_i \alpha_i = 0.$$

N es el número de instancias de entrenamiento

Hay una relación entre cada multiplicador *Lagrange* y cada instancia de los datos de entrenamiento. Una vez que se determinan éstos multiplicadores, el vector w y el umbral b se pueden calcular de la siguiente manera:

$$\vec{w} = \sum_{i=1}^N y_i \alpha_i \vec{x}_i, \quad b = \vec{w} \cdot \vec{x}_k - y_k \text{ for some } \alpha_k > 0.$$

No todos los conjuntos de datos pueden ser linealmente separables (Fig. 2). Puede que no exista un hiperplano que pueda separar las instancias de dos clases diferentes.

Práctica 5 - Handwritten Digit Recognition

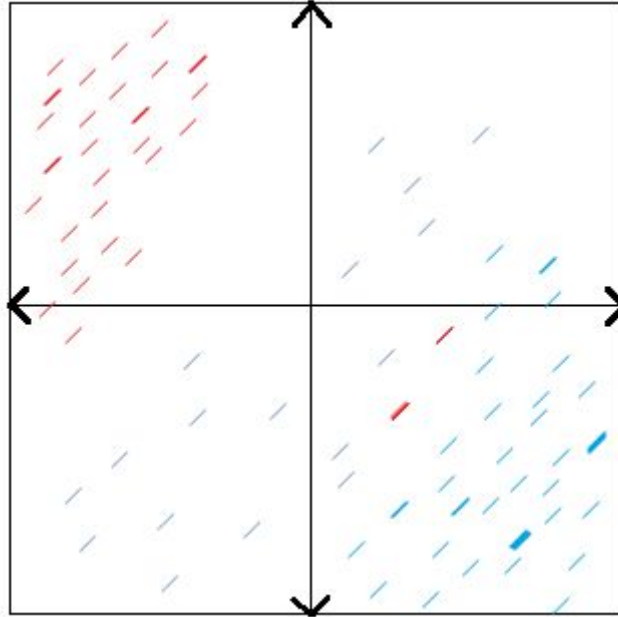


Fig. 2 - Plano bidimensional linealmente no separable.

Si este es el caso, se produce una modificación en el planteamiento del problema original de optimización. Esta modificación permite el fallo de que una instancia esté en el “lado” adecuado.

$$\min_{\bar{w}, b, \xi} \frac{1}{2} \|\bar{w}\|^2 + C \sum_{i=1}^N \xi_i \quad \text{subject to } y_i(\bar{w} \cdot \bar{x}_i - b) \geq 1 - \xi_i, \forall i,$$

$$0 \leq \alpha_i \leq C, \forall i.$$

ξ_i son variables que permiten el fallo y el parámetro C sacrifica anchura de los márgenes por menores cantidades de fallos con penalización.

Los algoritmos *SVM* pueden ser generalizados a clasificadores linealmente no separables. El hiperplano de estos clasificadores puede ser calculado mediante multiplicadores de *Lagrange*:

$$u = \sum_{j=1}^N y_j \alpha_j K(\bar{x}_j, \bar{x}) - b$$

Práctica 5 - Handwritten Digit Recognition

K hace referencia a una función Kernel, la cual mide la similitud o distancia entre el vector de entrada x y el vector de entrenamiento almacenado x_j .

El problema se resuelve mediante una función objetivo Ψ en forma dual (cuadrático).

$$\begin{aligned}\min_{\vec{\alpha}} \Psi(\vec{\alpha}) = \min_{\vec{\alpha}} \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N y_i y_j K(\vec{x}_i, \vec{x}_j) \alpha_i \alpha_j - \sum_{i=1}^N \alpha_i, \\ 0 \leq \alpha_i \leq C, \forall i, \\ \sum_{i=1}^N y_i \alpha_i = 0.\end{aligned}$$

El problema cuadrático generado no puede ser fácilmente resuelto mediante técnicas estándar. Se generaría una matriz cuyo número de elementos equivaldría al cuadrado del número de instancias del entrenamiento. Ésta sería tan grande que sobrepasaría los 128MB si hubiera más de 4000 instancias.

El algoritmo SMO será el encargado de resolver esta función.

Práctica 5 - Handwritten Digit Recognition

Sequential Minimal Optimization

El algoritmo *SMO* es simple y puede resolver rápidamente problemas *SVM* cuadráticos sin necesidad de memoria extra ni computación recursiva. Este algoritmo busca resolver el subproblema de optimización minimal de cada problema *SVM QP*.

[\[Sequential Minimal Optimization\]](#)

El subproblema cuenta con dos multiplicadores de *Lagrange* que son necesarios de optimizar. Una vez lograda esta tarea se modifica el problema para reflejar los nuevos óptimos. [\[SMO\]](#)

Desglose del algoritmo²

Para la resolución de los multiplicadores se aplican técnicas analíticas, evitando así el despilfarro de memoria, ya que únicamente almacena matrices (2x2) y evita el uso de librerías iterativas para la resolución de problemas *SVM QP*.

Por otro lado, existe un módulo heurístico para seleccionar los multiplicadores a optimizar.

Por último, un módulo para computar la '*constante b*'.

Módulo heurístico³

Primeramente debe comprobarse que ambos multiplicadores de *Lagrange* conjugan todas las restricciones del problema *SVM QP* completo, es decir, se busca aproximar el problema inicial con varias variables a un problema de optimización lineal de únicamente dos variables.

Para garantizar que el problema se resuelve, se aplican las condiciones KKT [1], y aquellos multiplicadores de *Lagrange* que no las cumplan serán los seleccionados para la aplicación de técnicas analíticas.

² [\[Microsoft Research\]](#)

³ [\[Microsoft Research\]](#)

Práctica 5 - Handwritten Digit Recognition

El problema se soluciona cuando todas las variables del problema SVM QP cumplen las siguientes condiciones:

$$[1] \quad a_i=0 \quad \Rightarrow \quad y_i * \gamma \geq 1$$

$$0 < a_i < C \quad \Rightarrow \quad y_i * \gamma = 1$$

$$a_i=C \quad \Rightarrow \quad y_i * \gamma \leq 1$$

Nota: a_i será cada variable involucrada en el problema SVM QP

El algoritmo evitará tener que recorrer el conjunto de instancias 'train' y utilizará la CPU para discriminar aquellas variables sobre las que no se hayan aplicado técnicas analíticas y por tanto no se encuentren dentro de las restricciones generales del problema.

Una vez seleccionado a_1 , no se premia el encontrar la variable más significativa para la resolución del problema lo cual a veces genera que no se progrese de una iteración a otra. En estos casos el algoritmo recurre al cálculo de errores E_i [2] para seleccionar variables más representativas:

$$[2] \quad E_i = \gamma(\text{old}) - y_i$$

Nota: Puede ser que sea imposible seleccionar un a_2 que satisfaga las restricciones junto a a_1 , en ese caso se descarta la primera variable del problema y se prosigue su resolución.

Resolución de multiplicadores de Lagrange (técnicas analíticas)⁴

Este problema es representable mediante los ejes cartesianos (x,y). Las restricciones de dimensión hacen que el problema siempre se represente dentro de una caja y la restricción de equivalencia lineal (mínimos cuadrados) causa que la línea sea diagonal.

Por estas razones, se optimizan simultáneamente al menos dos multiplicadores de Lagrange. También es mencionable, que existen variantes para la optimización lineal analítica de hasta cuatro multiplicadores.

⁴ [\[Microsoft Research\]](#)

Práctica 5 - Handwritten Digit Recognition

Caso de dos multiplicadores⁵

El algoritmo primeramente computará la línea (y_2) en función del segundo multiplicador (a_2) y se compara con la línea (y_1) computada según el primer multiplicador (a_1).

Posteriormente se multiplican, para determinar la pendiente de la recta que separa el hiperplano (γ).

$$\gamma = a_1 + s * a_2 \quad (\text{los valores de los multiplicadores son los antiguos})$$

$$s = y_1 * y_2$$

La comparación de las rectas permite determinar las cotas del problema [1] y minimizar los costes de su solución. Posteriormente se derivará [2] para optimizar el problema y calcular los nuevos valores de los multiplicadores de *Lagrange* teniendo en cuenta el error involucrado en la instancia de entrenamiento [3] :

[1] If (y_1) not equal (y_2) then:

$$L = \max(0, a_2(\text{old}) - a_1(\text{old})) ;$$

$$H = \min(C, C + a_2(\text{old}) - a_1(\text{old}));$$

else if (y_1) is equal (y_2) then

$$L = \max(0, a_2(\text{old}) - a_1(\text{old}) - C) ;$$

$$H = \min(C, a_2(\text{old}) + a_1(\text{old})) ;$$

Nota: L = longitud (horizontal) y H = altura constituyen las restricciones del problema de programación lineal.

[2] $\eta = 2k(x_1, x_2) - k(x_1, x_1) - k(x_2, x_2)$

Nota I: donde x_i es el vector de la recta y_i

Nota II: La derivada será negativa normalmente salvo casos excepcionales como instancias iguales donde será 0 o positiva.

⁵ [\[Microsoft Research\]](#)

Práctica 5 - Handwritten Digit Recognition

$$[3] \quad a_2(\text{new}) = a_2(\text{old}) - \frac{y_2 * (E_1 - E_2)}{\eta}$$

Nota: donde i = subíndice del multiplicador a estudiar

Para calcular $a_1(\text{new})$ debemos aplicar la función clipped para garantizar que los nuevos valores óptimos se encontrarán dentro de las restricciones del problema.

H:

if $a_2(\text{new}) \geq H$;

$a_2(\text{new, clipped}) = a_2(\text{new})$;

if $L < a_2(\text{new}) < H$;

L:

if $a_2(\text{new}) \leq L$;

$a_1(\text{new}) = a_1(\text{old}) + s * (a_2(\text{old}) - a_2(\text{new,clipped}))$;

Cálculo de la constante b (umbral de error y caché)⁶

La constante b constituye el umbral que debe calcularse [1] en cada iteración del algoritmo para garantizar que las variables siguen cumpliendo las condiciones KKT tras la aplicación del algoritmo SMO.

[1]

$$b_1 = E_1 + y_1 * (a_1(\text{new}) - a_1(\text{old})) * k(x_1, x_1) + y_2 (a_2(\text{new,clipped}) - a_2(\text{old})) * k(x_1, x_2) + b_1(\text{old})$$

$$b_2 = E_2 + y_1 * (a_1(\text{new}) - a_1(\text{old})) * k(x_1, x_2) + y_2 (a_2(\text{new,clipped}) - a_2(\text{old})) * k(x_2, x_2) + b_2(\text{old})$$

Cuando $b_1 = b_2$, $a_1(\text{new})$ y $a_2(\text{new})$ se encuentran dentro de $H * L$; $H \neq L$ entonces son valores válidos y todos los valores entre b_1 y b_2 serán posibles valores para $b(\text{new})$, el algoritmo cogerá el valor medio entre ellos.

⁶ [\[Microsoft Research\]](#)

Práctica 5 - Handwritten Digit Recognition

Cómo se apuntaba anteriormente el módulo heurístico hace uso de la CPU para calcular aquellas variables sobre las que no se hayan aplicado reglas analíticas, esto es posible gracias al almacenamiento de errores parciales en caché, los errores de las variables implicadas en la optimización se ponen a 0 y se actualizan los valores de error de las demás variables del problema según la fórmula siguiente:

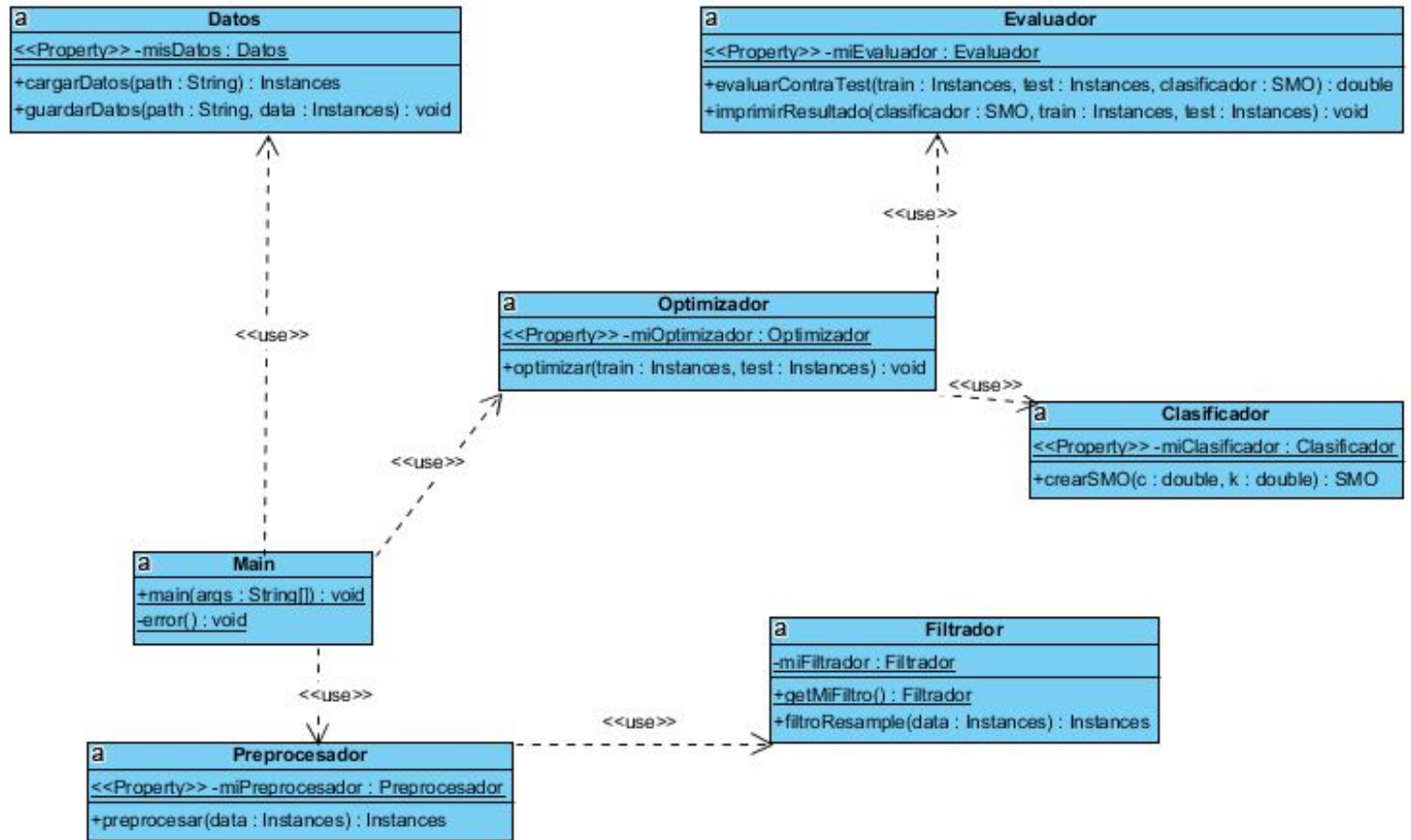
$$E_k(\text{new}) = E_k(\text{old}) + y_1(a_1(\text{new}) - a_1(\text{old})) \cdot k(x_1, x_k) + y_2(a_2(\text{new, clipped}) - a_2(\text{old})) \cdot k(x_2, x_k) + b(\text{old}) - b(\text{new})$$

Nota: el módulo heurístico hace uso de estos valores para la selección de multiplicadores

Los problemas que cuentan con más de dos clases (problemas multi-clases) se resuelven utilizando una clasificación “pairwise”, es decir, 1-vs-1. En el caso de haber más de dos clases, las probabilidades predecidas son emparejadas utilizando el método de emparejamiento “*Hastie and Tibshirani*”. [[SMO](#)]

Práctica 5 - Handwritten Digit Recognition

Diseño



Marco experimental

Datos

Análisis train-digits.arff

Este fichero cuenta con 1893 instancias y no tiene ningún “missing value”. Hay 257 atributos que se pueden justificar de la siguiente manera:

- 1 atributo para la clase {0,1,2,3,4,5,6,7,8,9} de tipo nominal.
- 256 atributos para los píxeles componentes de la imagen ($16 \times 16 = 256$) de tipo numérico. La cantidad de tinta en cada píxel está indicada mediante una escala $[-1,1]$, donde el límite inferior representa el color blanco, y el límite superior en cambio, representa el negro.

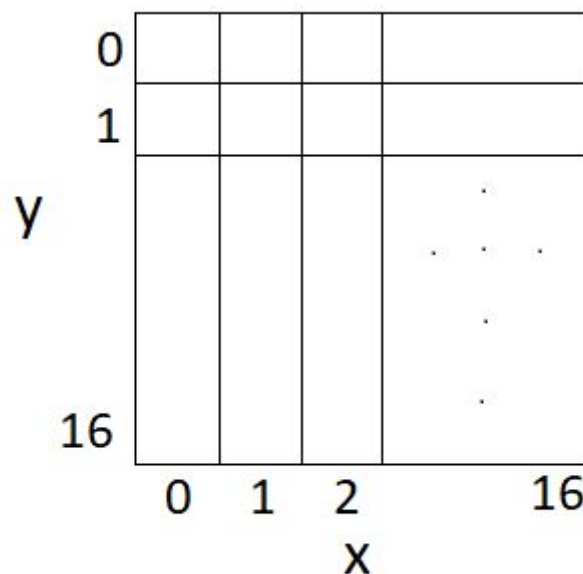


Fig. 3 - Cuadrícula píxeles de un número

En cuanto a los valores extremos y outliers, se ha utilizado el filtro “*InterquartileRange*” para comprobar la existencia de éstos. El resultado de este análisis se puede observar de forma gráfica en las Fig. 4 y Fig. 5.

Práctica 5 - Handwritten Digit Recognition

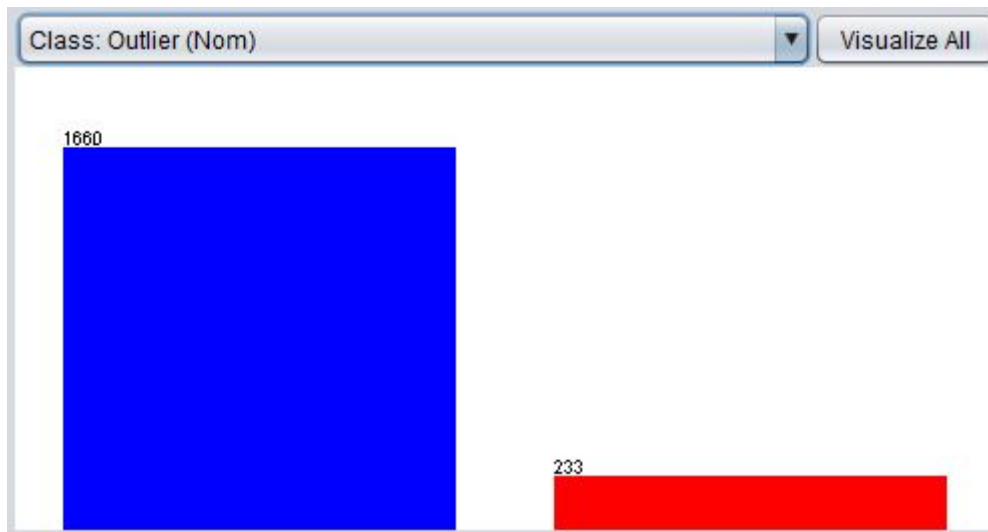


Fig.4 - Instancias con "Outlier" (rojo) y sin (azul)

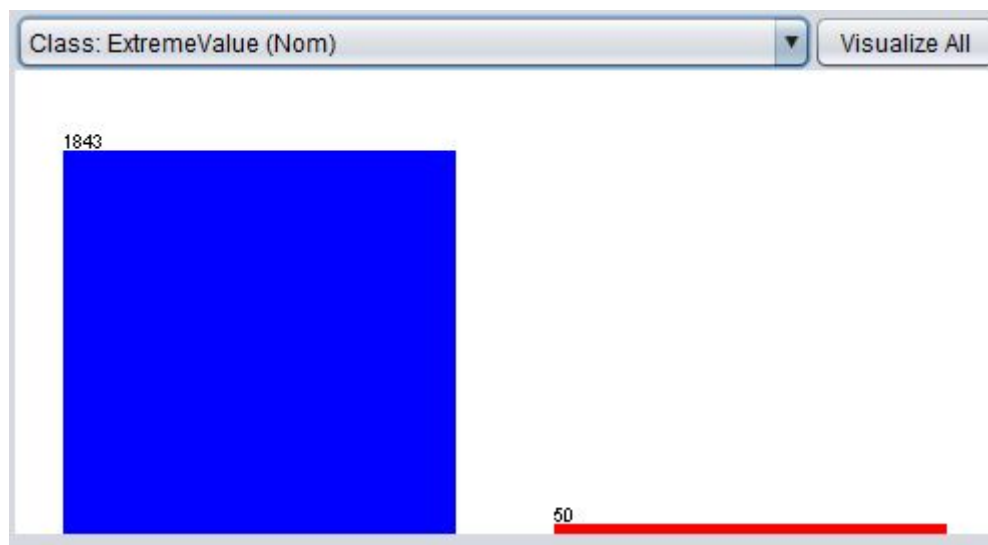


Fig. 5 - Instancias con "ExtremeValue" (rojo) y sin (azul)

El rojo indica la cantidad de "outliers" y "extreme values" que se han encontrado.

Los datos aunque no están balanceados del todo, sí que se puede apreciar un equilibrio cercano en el número de instancias de cada clase (Fig. 6).

Práctica 5 - Handwritten Digit Recognition

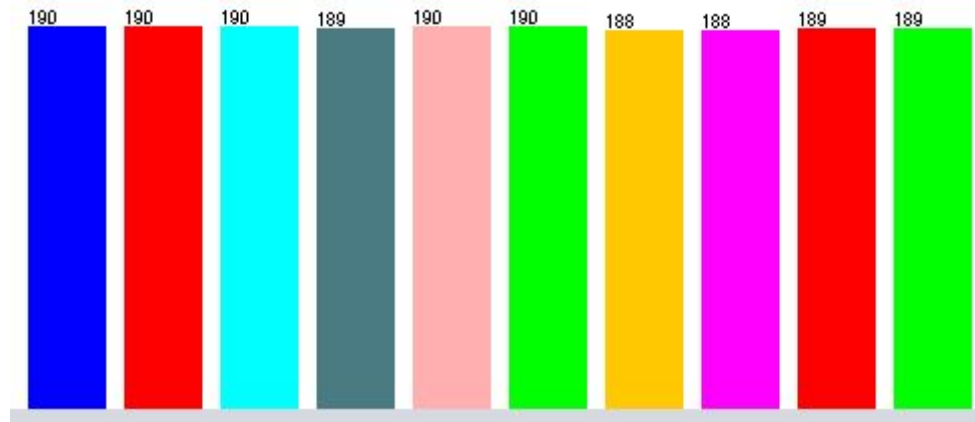


Fig. 6 - Número de instancias de las clases (0-9)

Si quisiéramos que todas las clases tuviesen el mismo número de instancias, tan solo se tendría que aplicar el filtro “Resample”.

Análisis test-digits.arff

Este fichero cuenta con 1796 instancias y no tiene ningún “missing value”. Hay 257 atributos, cuya justificación se puede apreciar en apartados anteriores.

En cuanto a los valores extremos y outliers, se ha utilizado el filtro “InterquartileRange” para comprobar la existencia de éstos. El resultado de este análisis se puede observar de forma gráfica en las Fig. 7 y Fig. 8.

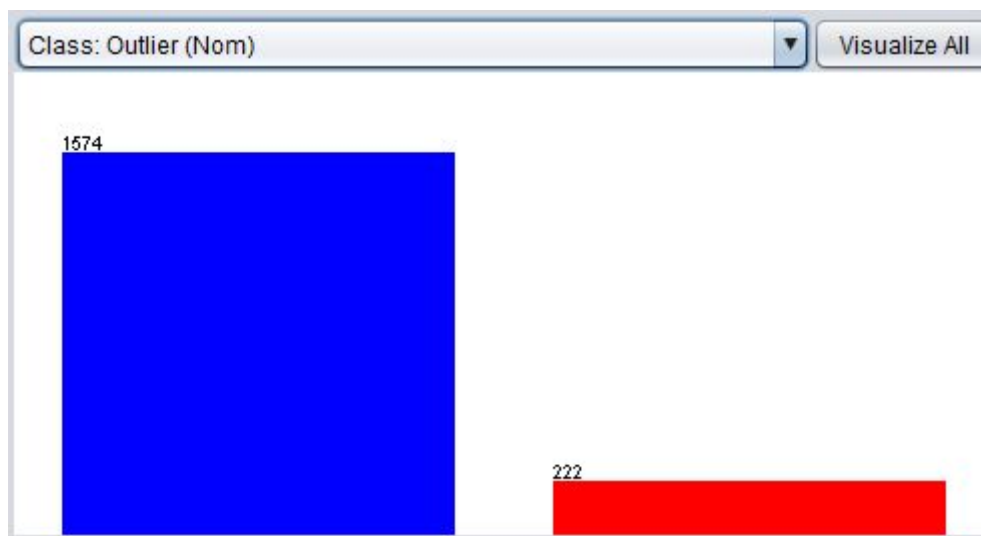


Fig. 7 - Instancias con “Outlier” (rojo) y sin (azul)

Práctica 5 - Handwritten Digit Recognition



Fig. 8 - Instancias con "ExtremeValue" (rojo) y sin (azul)

El rojo indica la cantidad de "outliers" y "extreme values" que se han encontrado.

Pre-proceso elegido

Se han realizado varias iteraciones en las que se han aplicado varios filtros (por separado y conjuntamente) a train y luego se ha intentado evaluar estos datos contra test mediante varios algoritmos.

Primeramente se comentarán los filtros probados, y finalmente se justificará el elegido o los elegidos a aplicar.

Filtros probados en train-digits.arff

- "InterquartileRange" para detectar las instancias que contienen valores outliers o extremos. Posteriormente se ha utilizado el filtro "RemoveWithValues" para borrar aquellas instancias que tienen dichos outliers o valores extremos.
- "Resample" para reducir el número de instancias, y que las clases tuviesen el mismo número de instancias. En [\[Digits - A Dataset for Handwritten Digit Recognition\]](#) se comenta que el error de clasificación aumenta con grandes cantidades de instancias. Se ha pensado probar a clasificar con el 80%-90% de las instancias, y se ha notado un incremento notable en la precisión de la clasificación.

Práctica 5 - Handwritten Digit Recognition

- Se ha probado a reducir la cuadrícula de los números a 15x15 eliminando el borde externo. La precisión ha aumentado, pero si se combina con los filtros anteriores, se obtiene una precisión bastante más baja a la ya obtenida anteriormente.
- “*Discretize*” se ha usado en los datos originales y en los obtenidos tras aplicar el filtro “*Resample*”. Se han obtenido buenos resultados, pero bajos en comparación con los ya obtenidos previamente.

Los filtros anteriores se han mezclado y probado en todas las combinaciones posibles, y la combinación más óptima consiste en aplicar el filtro “*Resample*” con un porcentaje de 88% para la selección de instancias a los datos train-digits.arff originales.

Ha sido un poco desconcertante el hecho de que la eliminación de los outliers y los valores extremos no haya mejorado la precisión en gran manera, es más, en ciertas ocasiones ésta ha disminuido considerablemente.

También se ha probado a cambiar los valores de los atributos de cada instancia teniendo en cuenta la imagen del número. Para esto se ha creado un programa que imprima por pantalla los números actualmente definidos en el archivo (.arff).

Se genera una ventana con cada instancia representada por colores según franjas de valores y se aumenta su resolución. El programa está pensado para variar sus franjas o colores y visualizar la composición final de la instancia.

Este programa complementado con otro capaz de sustituir valores del dataset train según franjas, y esto no ha permitido discretizar el conjunto arbitrariamente y comprobar sus efectos visualmente.

Práctica 5 - Handwritten Digit Recognition

Resultados experimentales

Se han evaluado datos de train contra test en varios algoritmos (*éstos optimizados al conjunto de datos*), los resultados más destacables han sido los siguientes:

Sin aplicar filtros

- SMO
 - Train vs test: 92,3%
- Bagging + SMO
 - Train vs test: 92,8%

Cuadrícula números 15x15

- SMO
 - Train vs test: 92,8%

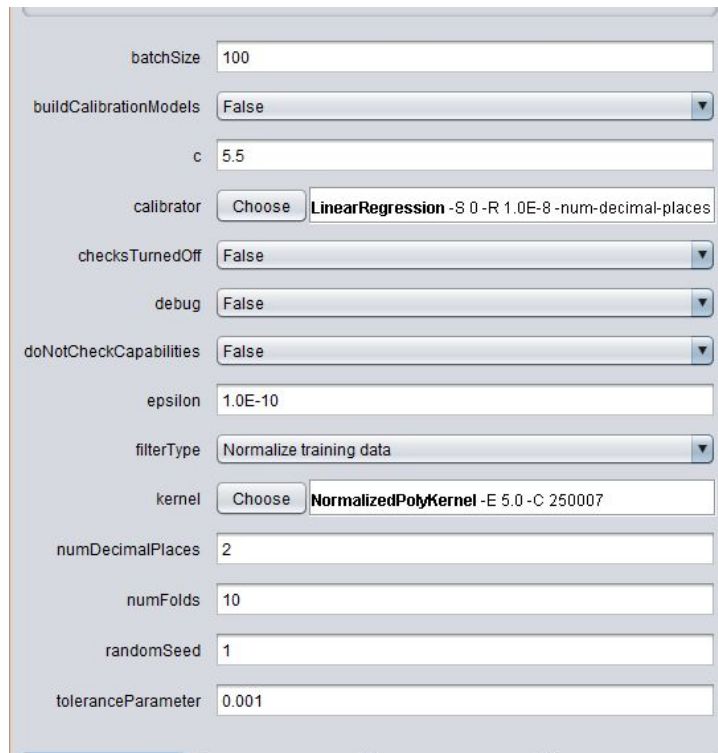
Filtro Resample

- SMO
 - Train vs test: **94,15%**
- KNN
 - Train vs test: 91%
- Multilayer Perceptron
 - Train vs test: 90%

Los no mencionados no han tenido un porcentaje de aciertos muy óptimo. Poco a poco se han ido orientando las pruebas de calidad hacia el modelo SMO al observar los altos porcentajes obtenidos.

El modelo final propuesto es el SMO, y su configuración es la siguiente:

Práctica 5 - Handwritten Digit Recognition



The image shows a configuration window for the Sequential Minimal Optimization (SMO) algorithm. The parameters are as follows:

Parameter	Value
batchSize	100
buildCalibrationModels	False
c	5.5
calibrator	Choose LinearRegression -S 0 -R 1.0E-8 -num-decimal-places
checksTurnedOff	False
debug	False
doNotCheckCapabilities	False
epsilon	1.0E-10
filterType	Normalize training data
kernel	Choose NormalizedPolyKernel -E 5.0 -C 250007
numDecimalPlaces	2
numFolds	10
randomSeed	1
toleranceParameter	0.001

Fig. 9 -Parámetros óptimos del SMO

Ejemplo de ejecución del ejecutable (descritos en la sección 5.1).

```
java -jar equipoDreamTeam_P5.jar train-digits.arff test-digits.arff
```

- train-digits.arff será el fichero train sin preprocesar.
- test-digits.arff será el fichero test sin preprocesar.

Se medirá el tiempo en segundos de lo que se tarde en entrenar el modelo óptimo y en hacer las correspondientes predicciones.

Práctica 5 - Handwritten Digit Recognition

```
C:\Users\Paula\Google Drive\UPV_EHU\3º\2º Cuatrimestre\SAD\Practicas\Practica 5>java -jar equipoDreamTeam_P5.jar
train-digits.arff test-digits.arff
1.54 segundos en entrenar el modelo optimo.
7.86 segundos en clasificar test.
9.4 segundos en total.
```

Correctly Classified Instances	1691	94.1537 %
Incorrectly Classified Instances	105	5.8463 %
Kappa statistic	0.935	
Mean absolute error	0.1605	
Root mean squared error	0.2729	
Relative absolute error	89.1501 %	
Root relative squared error	90.9644 %	
Total Number of Instances	1796	

```
=== Confusion Matrix ===
```

a	b	c	d	e	f	g	h	i	j	<-- classified as
172	0	0	0	4	0	1	0	1	2	a = 0
0	178	0	0	0	0	0	0	0	2	b = 1
0	2	176	0	0	0	0	0	1	0	c = 2
0	0	1	163	0	1	0	3	3	9	d = 3
0	0	0	0	178	0	0	1	0	1	e = 4
0	0	1	3	8	168	0	0	0	0	f = 5
1	0	0	0	3	2	174	0	0	0	g = 6
0	1	0	0	4	0	0	166	0	9	h = 7
0	1	2	4	3	0	1	0	160	8	i = 8
0	1	0	11	0	3	0	2	5	156	j = 9

```
=== Detailed Accuracy By Class ===
```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,956	0,001	0,994	0,956	0,975	0,972	0,990	0,971	0
	0,989	0,003	0,973	0,989	0,981	0,979	0,998	0,976	1
	0,983	0,002	0,978	0,983	0,981	0,978	0,998	0,972	2
	0,906	0,011	0,901	0,906	0,903	0,892	0,983	0,874	3
	0,989	0,014	0,890	0,989	0,937	0,931	0,994	0,900	4
	0,933	0,004	0,966	0,933	0,949	0,944	0,990	0,931	5
	0,967	0,001	0,989	0,967	0,978	0,975	0,993	0,972	6
	0,922	0,004	0,965	0,922	0,943	0,937	0,993	0,933	7
	0,894	0,006	0,941	0,894	0,917	0,908	0,982	0,874	8
	0,876	0,019	0,834	0,876	0,855	0,839	0,967	0,764	9
Weighted Avg.	0,942	0,006	0,943	0,942	0,942	0,936	0,989	0,917	

Fig. 10 - Ejemplo de ejecución programa

Conclusiones y trabajo futuro

El trabajo realizado durante este cuatrimestre nos ha enseñado a realizar un proceso, mediante el cual hemos descubierto la importancia de todas las fases que son necesarias para el análisis de datos.

El proceso de preproceso, por ejemplo, a pesar de no otorgarle inicialmente una importancia especial, a través de las diferentes pruebas que se han ido realizando hemos podido comprobar que es un proceso indispensable al que se debe dar una rigurosa atención.

También hemos podido ser conscientes de la relevancia del proceso de optimización de parámetros. Para obtener un buen f-measure y/o un buen porcentaje de aciertos, aparte de realizar un estudio muy exhaustivo de los datos con los que se va a trabajar, es necesario ajustar el modelo. Para esto último, se debe tener en cuenta cual va a ser el algoritmo a utilizar en proceso (en este caso *SMO*), ya que cada algoritmo hace uso de distintos parámetros.

En el caso del programa realizado, los parámetros óptimos que se han obtenido son excesivamente dependientes del preprocesamiento de los datos. A pesar de ello, se ha conseguido obtener un 94,15% de precisión en el análisis de los datos.

Teniendo en cuenta todo lo anteriormente mencionado, sería de interés poder realizar una optimización del algoritmo mayor sabiendo que la cota superior es del 100% (evaluación no-honesta).

Esto anterior significa que todavía hay margen para optimizar el algoritmo, ya sea bien por una optimización mayor de los parámetros como de un pre-procesamiento de datos más eficiente.

Bibliografía

➤ Introducción

- [Enunciado Practica 5 : Handwritten Digit Recognition]
- [Seewald, 2005] Seewald, A. K. (2005). Digits-a dataset for handwritten digit recognition. Austrian Research Institut for Artificial Intelligence Technical Report, Vienna (Austria).

➤ Marco teórico. SMO

- [Digits - A Dataset for Handwritten Digit Recognition]
- [SMO]<http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/SMO.html>
- [Microsoft Research]
<https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/smo-book.pdf>
- [Sequential Minimal Optimization]
<https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-98-14.pdf>

Valoración subjetiva

¿Ha alcanzado el equipo los objetivos que se plantean?

Sí, se plantea generar un modelo clasificador capaz de catalogar dígitos escritos a mano, por otro lado los objetivos documentales creemos que han sido cumplidos ya que hemos abordado el problema teórica y experimentalmente. En cuanto a la minimización del tiempo de clasificación, hemos logrado 9,4s. Por último en cuanto a objetivos de calidad hemos logrado 0,942 en la f-measure train vs test.

¿Cuánto tiempo se ha trabajado en esta tarea en promedio?

En esta tarea se han trabajado 15h más o menos .

¿Ha resultado de utilidad para el equipo la tarea planteada?

Sí, en un principio tratamos de abordar el problema mediante el algoritmo Multilayer Perceptron y tras la vista de los resultados tuvimos que emplear el algoritmo SMO. Hemos tenido que descomponerlo y entenderlo, por lo tanto a partir de ahora sabremos usarlo.

¿Qué aspectos de la tarea han despertado mayor interés? Sugerencias para mejorarla

Los aspectos que más interés nos han generado en la práctica, principalmente son los premios a los que se puede aspirar por lograr completarla. Por otro lado, el hecho de poder reutilizar trabajos previos de la asignatura para completarlo ha sido crítico para la decisión de llevarlo a cabo.

Sugerencias para que se consiga despertar mayor interés y motivación.

Para despertar mayor interés y motivación, se podría por un lado enfocar la tarea al aprendizaje de la teoría, y por otra lado a trabajar con un único modelo de clasificación no utilizado previamente para no favorecer a ningún grupo en la competición (en caso de buscar la puntuación).