

Aprendizaje Automático

Máquinas de Vectores Soporte

Ramon Ruiz Dolz
Aitor Signes Cuco
4CO21-2017

1. Introducción

Esta primera sesión de laboratorio trata sobre las máquinas de vectores soporte o SVM como nos referiremos durante la memoria. Este método de clasificación se basa en que dado un conjunto de muestras de entrenamiento y sus etiquetas, se puede entrenar una SVM para construir un modelo que se encargue de predecir la etiqueta de las nuevas muestras o muestras de test. En esta memoria presentaremos los datos obtenidos en el entrenamiento de una SVM para muestras linealmente separables, otra para muestras linealmente no separables y finalmente haremos unas pruebas de clasificación sobre el dataset SPAM con dos clases y sobre el dataset USPS, en este caso multiclase.

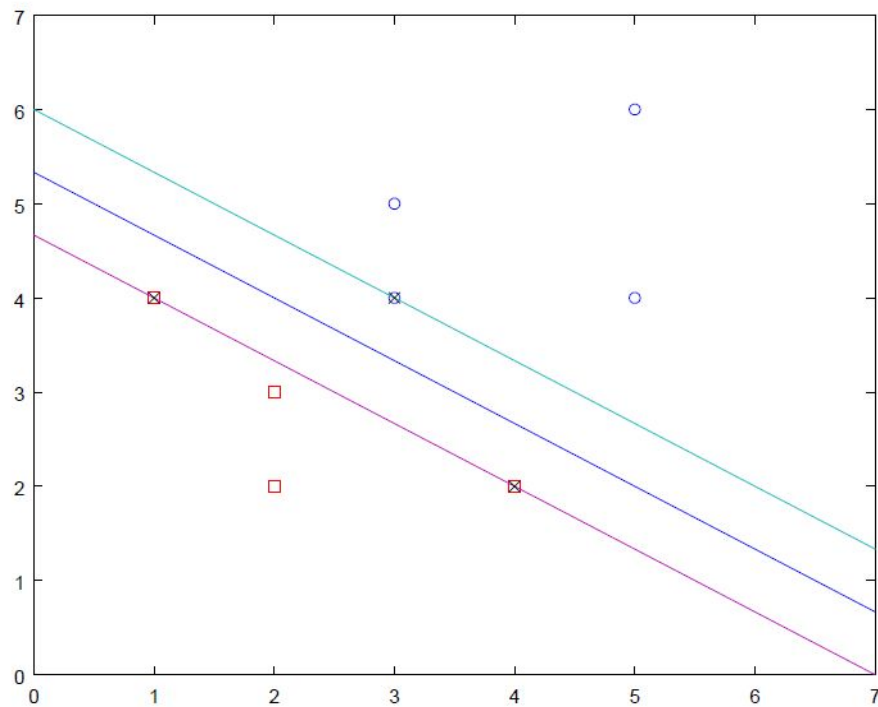
2. Obtención de las SVM con muestras linealmente separables.

En este primer ejercicio hemos obtenido la SVM con kernel lineal y un valor de C elevado mediante el comando `svmtrain(trlabels, tr, '-t 0 -c 1000')` y a partir de esto hemos obtenido una serie de datos relevantes para la representación gráfica de estas muestras linealmente separables.

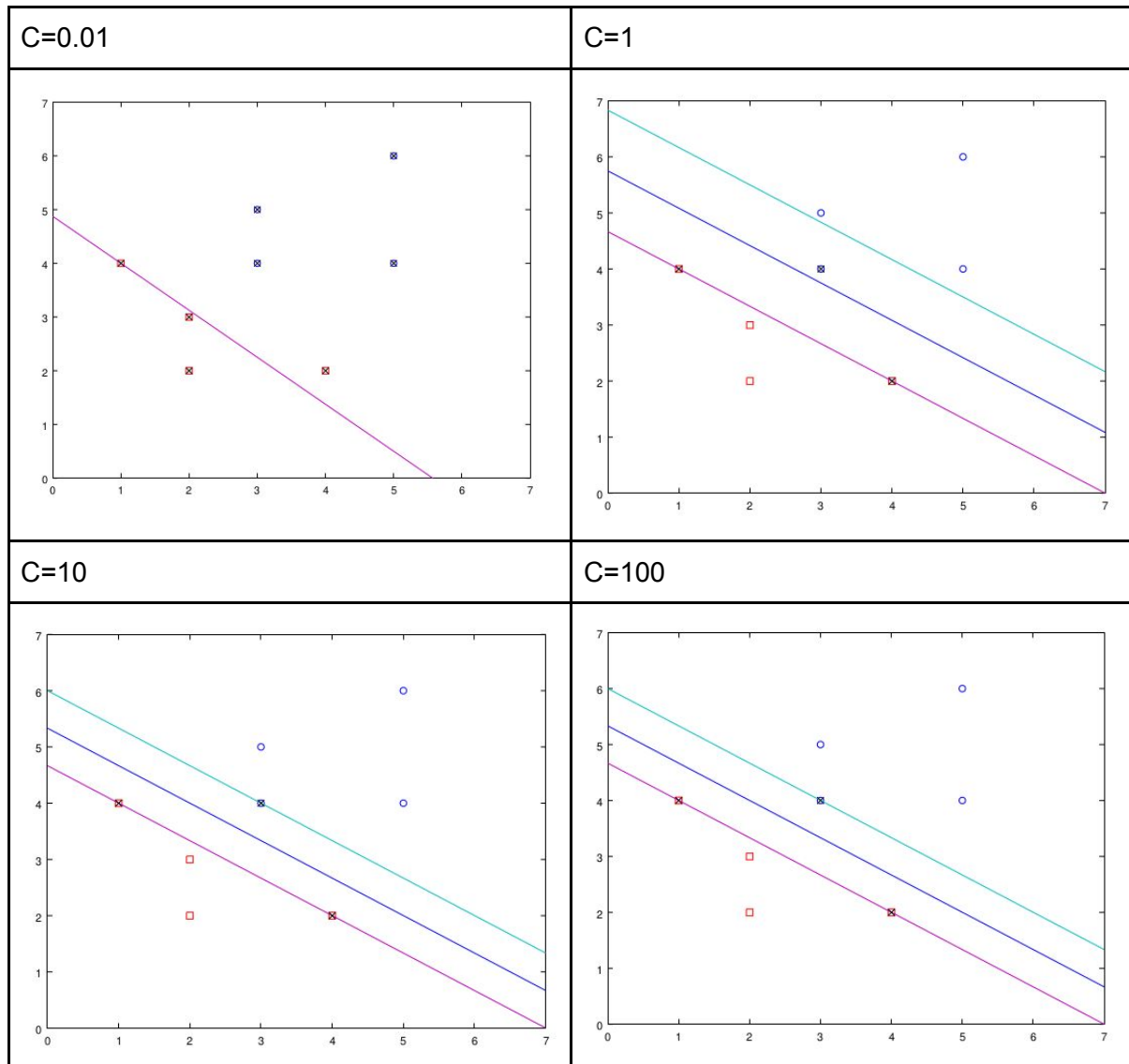
Los valores o parámetros a calcular son los siguientes, los multiplicadores de Lagrange obtenidos con los coeficientes del `svmtrain`. El vector de pesos obtenido de la suma de las componentes del vector resultado de multiplicar los multiplicadores de Lagrange por las muestras de soporte de `tr`. El umbral aplicando la fórmula estudiada en teoría y, finalmente la pendiente de la recta, su desplazamiento y los desplazamientos de los márgenes positivo y negativo.

Multiplicadores de Lagrange	Vector de pesos	Umbral	Parámetros de la recta y los márgenes
alpha = 0.87472 0.74989 -1.62461	w = -0.99955 -1.49978	umbral = 7.9987	m = 0.66647 d = -5.3332 d1 = -4.6665 d2 = -6

Finalmente obtenemos mediante un plot, la representación gráfica siguiente. Como podemos observar todas las muestras están correctamente clasificadas y los vectores de soporte marcados con una X para diferenciarlos.



A demás de este experimento propuesto, también hemos probado con diferentes valores de C . En este caso los valores 0.01, 1, 10 y 100. Y las gráficas resultantes mostradas debajo respectivamente. Como se puede observar para los valores de C tanto 10, 100 y 1000 son iguales, por lo tanto vemos que a partir de 10 ya se estabiliza el error y no varían las gráficas.

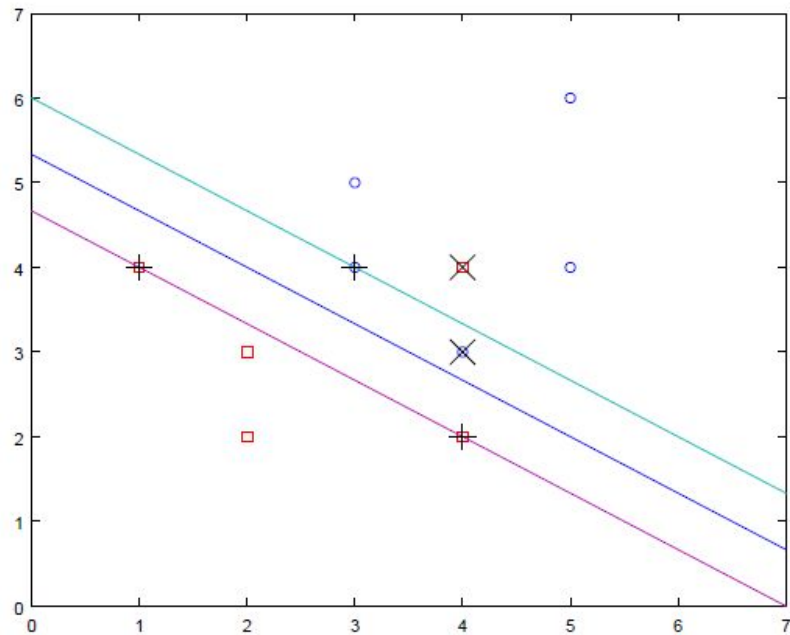


3. Obtención de las SVM con muestras linealmente no separables.

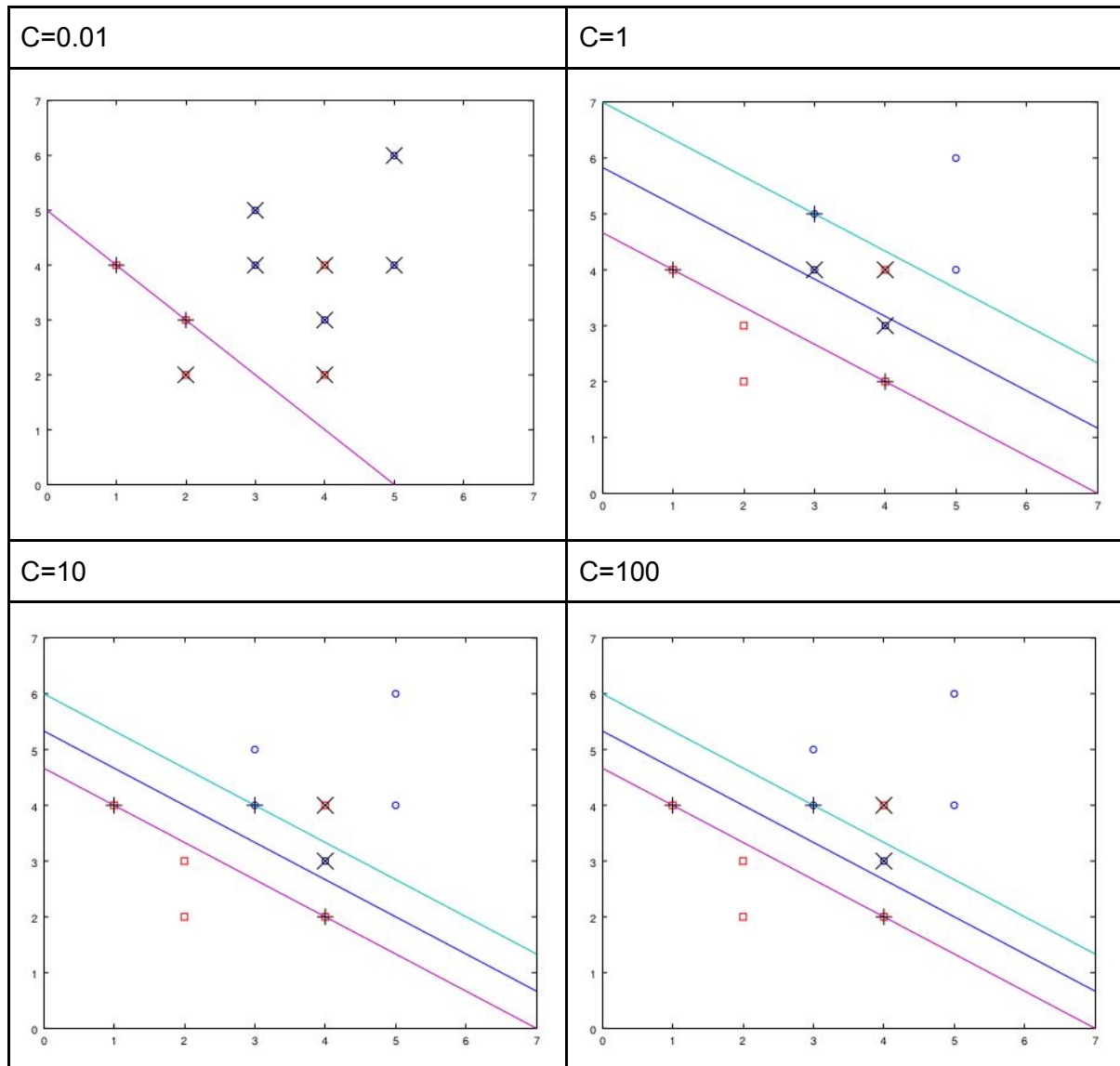
En este segundo experimento hemos realizado los mismos pasos que en el anterior añadiendo el cálculo de las ζ para las muestras mal clasificadas o bien clasificadas pero por debajo del margen.

Multiplicadores de Lagrange	Vector de pesos	Umbral	Parámetros de la recta y los márgenes	Zetas(ζ)
$\alpha =$ 250.87 500.75 1000.00 -751.62 -1000.00	$w =$ -0.99955 -1.49977	umbral = 7.9986	$m = 0.66647$ $d = -5.3332$ $d1 = -4.6665$ $d2 = -6.0000$	$z =$ 0.00000 -0.00000 3.00000 0.00000 0.50000

Tras obtener todos los parámetros y valores, en el plot la única diferencia que vemos ahora es que los vectores soporte están marcados con un + y aquellos que tienen un valor de ζ mayor que 0 están marcados con una X, puesto que son muestras mal clasificadas o clasificadas por debajo de los márgenes.



A demás de este experimento propuesto, también hemos probado con diferentes valores de C . En este caso los valores 0.01, 1 y 10. Y las gráficas resultantes mostradas debajo respectivamente. Como se puede observar para los valores de C tanto 10, 100 y 1000 son iguales, por lo tanto vemos que a partir de 10 ya se estabiliza el error y no varían las gráficas.



4. Clasificación de SPAM bidimensional.

Para este segundo ejercicio, buscamos la mejor configuración de kernel y C para obtener el mínimo error al clasificar los datos de SPAM. Para ello tenemos un primer loop que iterará por cada tipo de kernel lineal, polinomial, radial o sigmoide, y posteriormente, para cada kernel se probará con diferentes valores de C (0.1,1,10,100 y 1000). Tras lanzar las ejecuciones del script obtenemos la siguiente salida:

```
Kernel:0 C:0.01 Error:0 Intervalo1:0 Intervalo2:0
Kernel:1 C:0.01 Error:0.01231 Intervalo1:0.018126 Intervalo2:0.0064943
Kernel:2 C:0.01 Error:0.39392 Intervalo1:0.41969 Intervalo2:0.36815
Kernel:3 C:0.01 Error:0.45547 Intervalo1:0.48173 Intervalo2:0.4292

Kernel:0 C:0.1 Error:0.00072411 Intervalo1:0.0021429 Intervalo2:-0.00069463
Kernel:1 C:0.1 Error:0.0094135 Intervalo1:0.014507 Intervalo2:0.0043204
Kernel:2 C:0.1 Error:0.26285 Intervalo1:0.28607 Intervalo2:0.23964
```

```

Kernel:3 C:0.1 Error:0.56988 Intervalo1:0.59599 Intervalo2:0.54376

Kernel:0 C:1 Error:0.0028965 Intervalo1:0.0057309 Intervalo2:6.2043e-05
Kernel:1 C:1 Error:0.0079652 Intervalo1:0.012654 Intervalo2:0.0032769
Kernel:2 C:1 Error:0.13251 Intervalo1:0.15039 Intervalo2:0.11463
Kernel:3 C:1 Error:0.66474 Intervalo1:0.68963 Intervalo2:0.63984

Kernel:0 C:10 Error:0.0028965 Intervalo1:0.0057309 Intervalo2:6.2043e-05
Kernel:1 C:10 Error:0.0036206 Intervalo1:0.0067884 Intervalo2:0.00045275
Kernel:2 C:10 Error:0.091962 Intervalo1:0.1072 Intervalo2:0.076721
Kernel:3 C:10 Error:0.68284 Intervalo1:0.70738 Intervalo2:0.65829

Kernel:0 C:100 Error:0.0028965 Intervalo1:0.0057309 Intervalo2:6.2043e-05
Kernel:1 C:100 Error:0.0021723 Intervalo1:0.0046279 Intervalo2:-0.00028322
Kernel:2 C:100 Error:0.092686 Intervalo1:0.10798 Intervalo2:0.077392
Kernel:3 C:100 Error:0.32078 Intervalo1:0.3454 Intervalo2:0.29616

Kernel:0 C:1000 Error:0.0028965 Intervalo1:0.0057309 Intervalo2:6.2043e-05
Kernel:1 C:1000 Error:0.0021723 Intervalo1:0.0046279 Intervalo2:-0.00028322
Kernel:2 C:1000 Error:0.092686 Intervalo1:0.10798 Intervalo2:0.077392
Kernel:3 C:1000 Error:0.28602 Intervalo1:0.30986 Intervalo2:0.26219

```

Esta información es la que se representa en la tabla de abajo, donde podemos apreciar que para obtener el mínimo error de clasificación el mejor kernel es el lineal y al aplicar valores cercanos al 0 el error se reduce hasta casi llegar al 0, es decir que es linealmente separable.

Kernel	C	Error	Intervalo
0	0.01	0	[0,0]
0	0.1	0.00072411	[0.0021429,-0.00069463]
0	1	0.0028965	[0.0057309,6.2043e-05]
0	10	0.0028965	[0.0057309,6.2043e-05]
0	100	0.0028965	[0.0057309,6.2043e-05]
0	1000	0.0028965	[0.0057309,6.2043e-05]
1	0.01	0.01231	[0.018126,0.0064943]
1	0.1	0.0094135	[0.014507,0.0043204]
1	1	0.0079652	[0.012654,0.0032769]
1	10	0.0036206	[0.0067884,0.00045275]
1	100	0.0021723	[0.0046279,-0.00028322]

1	1000	0.0021723	[0.0046279,-00028322]
2	0.01	0.39392	[0.41969,0.36815]
2	0.1	0.28607	[0.28607,0.23964]
2	1	0.13251	[0.15039,0.11463]
2	10	0.091962	[0.1072,0.076721]
2	100	0.092686	[0.10798,0.077392]
2	1000	0.092686	[0.10798,0.077392]
3	0.01	0.45547	[0.48173, 0.4292]
3	0.1	0.56988	[0.59599,0.54376]
3	1	0.66474	[0.68963,0.063984]
3	10	0.68284	[0.70738,0.65829]
3	100	0.32078	[0.3454,0.29616]
3	1000	0.28602	[0.309986,0.26219]

Como se puede observar en la tabla, los mejores resultados se obtienen con los kernel lineal (0) y polinómico (1). El menor error se ha obtenido con Kernel 0 y C 0.01 obteniendo un error del 0%. Otros errores muy bajos se han obtenido con el Kernel 1 y valores de la C 100 y 1000.

5. Clasificación USPS multidimensional.

Para este último ejercicio hemos realizado un clasificador para las muestras de USPS. El objetivo, igual que en el ejercicio anterior consiste en encontrar el mejor Kernel y la mejor C de manera que al entrenar una SVM con estos parámetros se obtenga el menor error de clasificación posible. Tras lanzar las ejecuciones del script obtenemos estos datos:

```
Kernel:0 C:0.01 Error:0.068261 Intervalo1:0.079295 Intervalo2:0.057228
Kernel:1 C:0.01 Error:0.44395 Intervalo1:0.46568 Intervalo2:0.42221
Kernel:2 C:0.01 Error:0.25311 Intervalo1:0.27214 Intervalo2:0.23409
Kernel:3 C:0.01 Error:0.36672 Intervalo1:0.3878 Intervalo2:0.34563

Kernel:0 C:0.1 Error:0.071749 Intervalo1:0.08304 Intervalo2:0.060458
Kernel:1 C:0.1 Error:0.14599 Intervalo1:0.16144 Intervalo2:0.13054
Kernel:2 C:0.1 Error:0.085202 Intervalo1:0.097416 Intervalo2:0.072987
Kernel:3 C:0.1 Error:0.11111 Intervalo1:0.12486 Intervalo2:0.097362

Kernel:0 C:1 Error:0.073742 Intervalo1:0.085176 Intervalo2:0.062308
Kernel:1 C:1 Error:0.071251 Intervalo1:0.082505 Intervalo2:0.059996
Kernel:2 C:1 Error:0.057798 Intervalo1:0.068007 Intervalo2:0.047588
```



```

Kernel:3 C:1 Error:0.11161 Intervalo1:0.12539 Intervalo2:0.097833

Kernel:0 C:10 Error:0.07424 Intervalo1:0.08571 Intervalo2:0.06277
Kernel:1 C:10 Error:0.061784 Intervalo1:0.072317 Intervalo2:0.05125
Kernel:2 C:10 Error:0.049826 Intervalo1:0.059345 Intervalo2:0.040306
Kernel:3 C:10 Error:0.16293 Intervalo1:0.17909 Intervalo2:0.14677

Kernel:0 C:100 Error:0.07424 Intervalo1:0.08571 Intervalo2:0.06277
Kernel:1 C:100 Error:0.058794 Intervalo1:0.069086 Intervalo2:0.048502
Kernel:2 C:100 Error:0.050822 Intervalo1:0.060431 Intervalo2:0.041213
Kernel:3 C:100 Error:0.19432 Intervalo1:0.21163 Intervalo2:0.17701

Kernel:0 C:1000 Error:0.07424 Intervalo1:0.08571 Intervalo2:0.06277
Kernel:1 C:1000 Error:0.059292 Intervalo1:0.069625 Intervalo2:0.04896
Kernel:2 C:1000 Error:0.050822 Intervalo1:0.060431 Intervalo2:0.041213
Kernel:3 C:1000 Error:0.19781 Intervalo1:0.21524 Intervalo2:0.18038

```

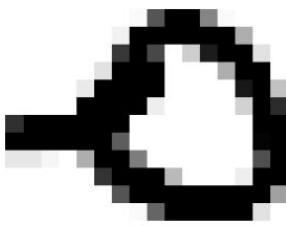

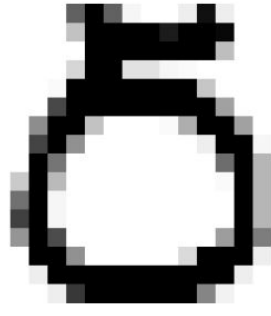
Tras el procesamiento de los datos los hemos organizado en la siguiente tabla, aquí podemos observar los errores obtenidos con kernel y C determinados así como el intervalo de confianza al 95% respectivo.

Kernel	C	Error	Intervalo
0	0.01	0.068261	[0.079295,0.057228]
0	0.1	0.071749	[0.08304,0.060458]
0	1	0.073424	[0.085176,0.062308]
0	10	0.07424	[0.08571,0.6277]
0	100	0.07424	[0.08571,0.6277]
0	1000	0.07424	[0.08571,0.6277]
1	0.01	0.44395	[0.46568,0.42221]
1	0.1	0.14599	[0.16144,0.13054]
1	1	0.071251	[0.82505,0.059996]
1	10	0.061784	[0.072317,0.05125]
1	100	0.058794	[0.069086,0.048502]
1	1000	0.059292	[0.069625,0.04896]
2	0.01	0.25311	[0.27214,0.23409]
2	0.1	0.085202	[0.097426,0.072987]

2	1	0.057798	[0.0680007,0.047588]
2	10	0.049826	[0.59345,0.040306]
2	100	0.050822	[0.060431,0.041213]
2	1000	0.050822	[0.060431,0.041213]
3	0.001	0.36672	[0.3878,0.34563]
3	0.1	0.11111	[0.12486,0.097362]
3	1	0.11161	[0.12539,0.097833]
3	10	0.16293	[0.17909,0.]
3	100	0.19432	[0.21163,0.17701]
3	1000	0.19781	[0.21524,0.18038]

Como podemos observar, en este caso, al ser un dataset multiclase la cosa es diferente. Con el kernel lineal obtenemos poco error, sin embargo es notablemente superior al del ejemplo de SPAM. En cambio, podemos observar como con el uso del kernel radial (2) estamos obteniendo una tasa inferior de error. Concretamente el mínimo error lo obtenemos con kernel radial (2) y C 10. Como se puede observar en la última página veremos algunos ejemplos de muestras vector de soporte representadas gráficamente.

Con el uso del comando imshow podemos mostrar cómo serían los vectores de soporte de la máquina, es decir las muestras en el límite:

Muestra de vs 0	Muestra de vs 9	Muestra de vs 5
		

Muestra de vs 7	Muestra de vs 3
