SVM de una clase: aplicación a detección de novedad

Alejandro Reyna

Introducción al Reconocimiento de Patrones

Trabajo Final 2008

Febrero 2009, IIE, FING

Introducción

El presente trabajo pretende presentar los conceptos básicos del uso de Máquinas de Vectores de Soporte (SVM) aplicado a problemas de una clase (OC-SVM), y en específico a problemas de detección de novedad.

La idea básica de la técnica es aprender de forma no supervisada, dado un conjunto de datos, la densidad que los rige de forma de poder detectar outliers, o muestras que pueden considerarse apartadas del comportamiento del resto.

Desde el punto de vista práctico se trabajó la librería LIBSVM [4], probándose sobre todo la versión en Java tanto utilizando un port para Weka en las pruebas como modificando algunas aplicaciones que se suministran en la librería a fin de lograr implementar entre otros la búsqueda de los mejores parámetros para ser usados al entrenar el algoritmo.

La parte final de trabajo toma algunos ejemplos tomados de la base USPS de dígitos escritos a mano (las primeras aplicaciones de SVM se refieren a temas de OCR) así como datos de tráfico en radio bases celular, en particular se estudió el de una base con un comportamiento bastante diferente en baja temporada con respecto a la alta temporada de verano, si llegar a tener problemas graves de congestión.

El trabajo comienza por una breve introducción a las nociones básicas de SVM, se concentra un poco más en el caso de oc-SVM, presenta algunos detalles de la librería LIBSVM utilizada y presenta algunos de los resultados experimentales realizados y mencionados anteriormente.

SVM: Máquinas de Vectores de soporte

Las máquinas de vectores de soporte (SVM por sus siglas en ingles, "Support Vector Machine"), fueron derivadas de la teoría de aprendizaje estadístico postulada por Vapnik y Chervonenkis. Las SVM fueron presentadas en 1992 y adquirieron fama cuando dieron resultados muy superiores a las redes neuronales en el reconocimiento de letra manuscrita, usando como entrada pixeles.

SVM esta ganando además gran popularidad como herramienta para la identificación de sistemas no lineales, esto debido principalmente a que SVM esta basado en el principio de minimización del riesgo estructural (SRM por sus siglas en ingles, "Structural Risk Minimization"), principio originado de la teoría de aprendizaje estadístico desarrollada por Vapnik, el cual ha demostrado ser superior al principio de minimización del riesgo empírico (ERM por sus siglas en ingles .Empirical Risk Minimization"), utilizado por las redes neuronales convencionales.

Algunas de las razones por las que este método ha tenido éxito es que no padece de mínimos locales y el modelo solo depende de los datos con más información llamados vectores de soporte (SV por sus siglas en ingles, "Support Vectors"). Las grandes ventajas que tiene SVM son:

- Una excelente capacidad de generalización, debido a la minimización del riesgo estructurado.
- Existen pocos parámetros a ajustar; el modelo solo depende de los datos con mayor información.
- La estimación de los parámetros se realiza a través de la optimización de una función de costo convexa, lo cual evita la existencia de un mínimo local.
- La solución de SVM es sparse, esto es que la mayoría de las variables son cero en la solución de SVM, esto quiere decir que el modelo final puede ser escrito como una combinación de un número muy pequeño de vectores de entrada, llamados vectores de soporte.
- Lo anterior implica que la complejidad del clasificador depende de la cantidad de vectores que determinan la frontera y no de la dimensión del espacio.

En [1], [3] y [5] se puede encontrar una buena introducción a este método. SVM resuelve un problema cuadrático donde el número de coeficientes es igual al número de entradas o datos de entrenamiento. Este hecho hace que para grandes cantidades de datos las técnicas numéricas de optimización, existentes para resolver el problema cuadrático, no sean admisibles en términos computacionales.

Este es un problema que impide el uso de SVM para la identificación de sistemas no lineales en línea, esto es, en casos en los que las entradas son obtenidas de manera secuencial y el aprendizaje se realiza en cada paso.

Clasificación por hiperplanos

Supongamos que hay m observaciones y cada una consiste en un par de datos:

- un vector $x_i \in R^n, i = 1,...,m$
- una etiqueta $y_i \in \{+1,-1\}$

Supóngase que se tiene un hiperplano que separa las muestras positivas (+1) de las negativas (-1). Los puntos **x**i que están en el hiperplano satisfacen **w**·x+b=0.

W es normal al hiperplano, siendo $\frac{|b|}{\|w\|}$ la distancia perpendicular del plano al origen . Lo que se quiere es definir dos hiperplanos que separen las muestras según sus etiquetas y_i de forma que **w**·x_i+b=+1 para y_i = +1 y **w**·x_i+b= -1 para y_i = -1 o lo que es lo mismo y_i(**w**·x_i+b)=+1. (1)

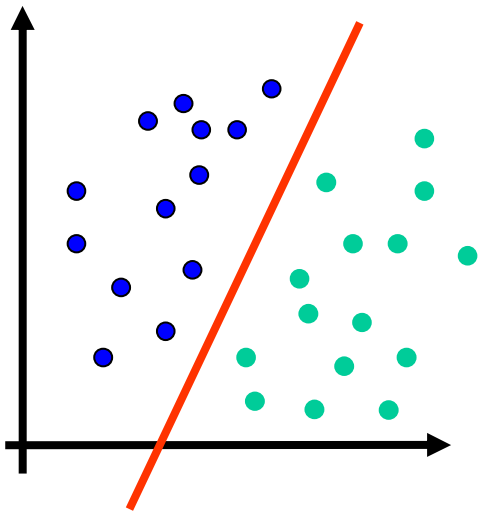


Fig.1: Hiperplano que separa las dos clases

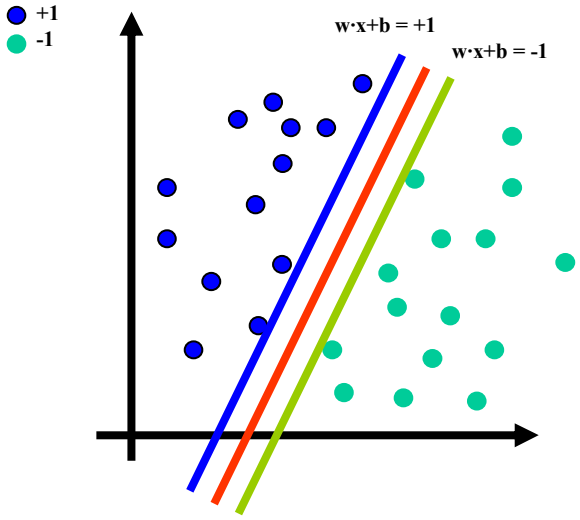


Fig.2: Hiperplanos de mayor margen negativo y positivo

Sea d+ (d-) la distancia más corta entre el hiperplano positivo (negativo) y el punto positivo (negativo) más cercano. Definimos como "margen" a la distancia entre los hiperplanos "positivo" y "negativo". El margen es igual a: $\frac{2}{\|\mathbf{w}\|}$ (2)

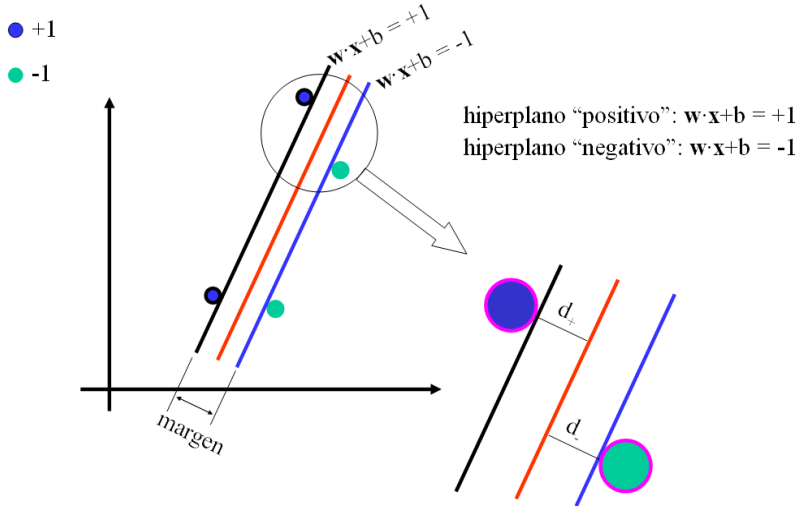


Fig.3: Hiperplanos y margen

La idea es encontrar un hiperplano con el máximo "margen". Esto es un problema de optimización: maximizar $\frac{2}{\|\mathbf{w}\|}$ condicionado a $y_i(\mathbf{w}\cdot x_i+b)=+1$. (3)

Los cual se puede expresar como: minimizar $\|w\|^2$ sujeto a $y_i(\mathbf{w}\cdot x_i+b)=+1$. (4)

Se introducen los multiplicadores de Lagrange para que todas las restricciones se agrupen en una única ecuación:

$$L_P \equiv \tfrac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{ml} \alpha_i y_i \left(\mathbf{w}\cdot\mathbf{x}_i + b\right) + \sum_{i=1}^{m} \alpha_i \quad (5)$$

Haciendo que los gradientes de Lp respecto a w y b sean cero, se obtienen las siguientes condiciones:

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \qquad \sum_{i=1}^{m} \alpha_i y_i = 0 \quad (6) \text{ y } (7)$$

Lo que sustituido en Lp nos da el llamado problema dual:

$$L_D = \sum_{i=1}^{m} \alpha_i + \sum_{i=1,j=1}^{m} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \quad (8)$$

El problema de optimización queda entonces dado por:

$$\text{minimizar } L_D = \sum_{i=1}^{m} \alpha_i + \sum_{i=1,j=1}^{m} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \text{ sujeto a } (10)$$

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \text{ y } \sum_{i=1}^{m} \alpha_i y_i = 0 \text{ . } (11)$$

Cuando los datos no se pueden separar linealmente se hace un cambio de espacio mediante una función Φ, que transforme los datos de manera que se puedan separar linealmente en el nuevo espacio.

Este nuevo espacio también tiene definido un producto interno, lo que permite utilizando el llamado "kernel trick", calcular el producto interno de la imagen de las muestras en el nuevo espacio de características utilizando funciones núcleo sin tener que expresar explícitamente el mapeo que hace Φ.

Algunos de estos núcleos son los *polinómicas* o las *Funciones de Base Radial* (RBF).

En la próxima sección se ve esto en más detalle aplicándolo al caso de SVM de una clase.

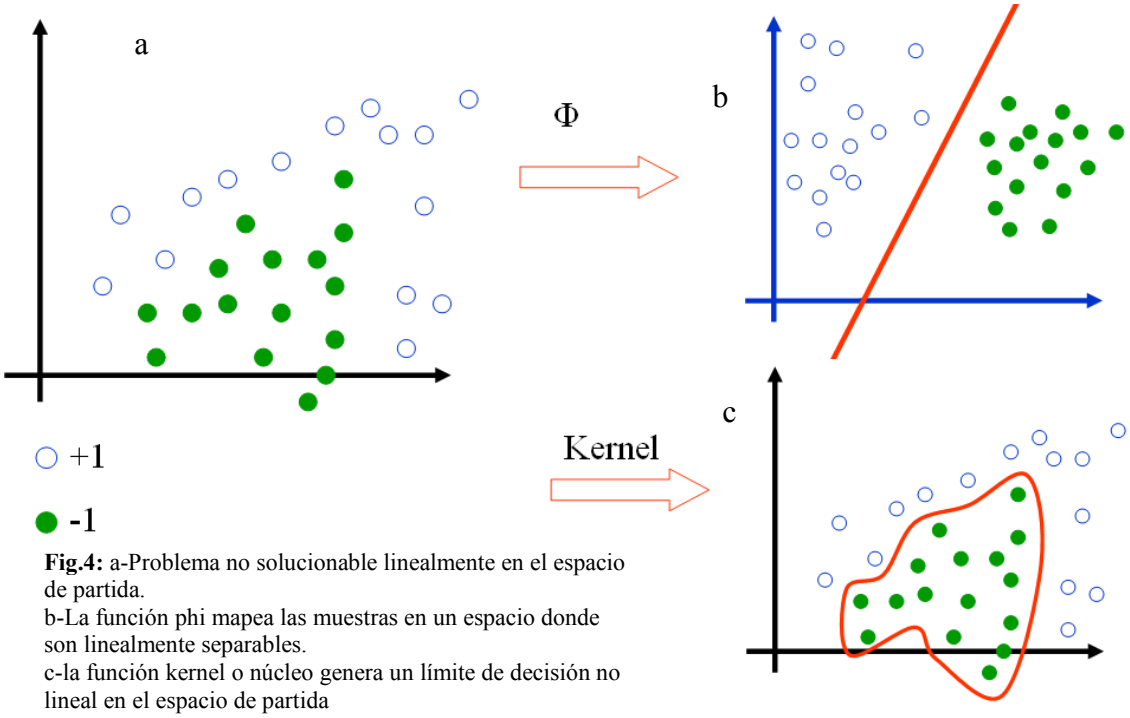


Fig.4: a-Problema no solucionable linealmente en el espacio de partida.
b-La función phi mapea las muestras en un espacio donde son linealmente separables.
c-la función kernel o núcleo genera un límite de decisión no lineal en el espacio de partida

SVM aplicado a problemas de clasificación de una clase

El algoritmo SVM es en general aplicado como un algoritmo de dos clases. En varios ejemplos [2],[6],[9] se presentan modificaciones a este algoritmo que permiten su aplicación a problemas donde solo se busca establecer si una muestra pertenece o no a la clase con la cual se ha entrenado. La idea es la búsqueda de "outliers" entre los datos disponibles considerándolos como ejemplos de la clase "lo otro" o "lo nuevo", respecto a la clase con que se entrena. Estos métodos son usualmente conocidos como one-class SVM (OC-SVM).

Schölkopf et al. en [3] sugirieron un método para adaptar SVM a problemas de una clase. Al igual que en SVM multi clase la idea es transformar el espacio de características a través de un kernel o núcleo, la diferencia en oc-SVM es que se trata al origen como el único miembro de la segunda clase de "outliers". Luego utilizando parámetros de relajación se separa la imagen de la clase que nos interesa estudiar del origen utilizándose las técnicas estándar de SVM para dos clases.

Supongamos por ejemplo que se tiene una distribución P en el espacio de características. Se busca un subconjunto S, simple dentro de dicho espacio de características de forma de que la probabilidad de que un punto de test regido por P caiga fuera del subconjunto S está acotada por un cierto valor prefijado de antemano. En otras palabras de todos los puntos regidos por P, no más de cierto número puede caer fuera de dicha región S.

Llamemos $\upsilon \in (0,1]$ a dicho parámetro que acota la cantidad de puntos por fuera de S. La solución a este problema se obtiene estimando una función f positiva para los puntos dentro de S y negativa en el complemento de S. En otras palabras, se define f de forma de que valga +1 en una región "pequeña" que contenga a la mayoría de los vectores de datos y que valga -1 en el resto del espacio de características.

$$f(x) = \begin{cases} +1 \text{ si } x \in S \\ -1 \text{ si } x \in \bar{S} \end{cases} \quad (12)$$

Teniendo en cuenta esto asumamos que $x_1,\ldots, x_m \in X$ son muestras de entrenamiento pertenecientes a una clase X, siendo x un subconjunto compacto de R^N.

Sea $\Phi : X \to H$ un mapeo que transforma dichas muestras en un espacio de características *H* con producto interno (espacio de Hilbert), de forma que el producto interno de la imagen de Φ puede ser computado evaluando alguna función núcleo K, en forma simple en el nuevo espacio:

$$K(x_i, x_j) = \left\langle \Phi(x_i), \Phi(x_j) \right\rangle = \Phi(x_i)^T \Phi(x_j) \quad (13)$$

Si bien se han propuesto diferentes núcleos los más comúnmente usados son:

- lineal: $K(x_i, x_j) = x_i^T x_j$. (14)
- polinómico: $K(x_i, x_j) = (\gamma . x_i^T x_j + r)^d$, $\gamma > 0$. (15)
- función de base radial (RBF): $K(x_i, x_j) = e^{-\gamma \|x - x'\|^2}$, $\gamma > 0$ (16) (muchas veces se expresa $\gamma = \frac{1}{2\sigma^2}$ aunque en general es un parámetro general del núcleo).
- sigmoideo: $K(x_i, x_j) = \tanh(\gamma . x_i^T x_j + r)$ (17)

(γ, r y d son parámetros de los núcleos mencionados, su denominación es compatible con la librería LIBSVM que veremos más adelante)

Una vez elegido el núcleo adecuado, la estrategia es como dijimos mapear los datos en el espacio de características correspondiente al núcleo y separarlos del origen con el mayor margen posible.

Para un nuevo punto *x*, el valor *f(x)* queda determinado evaluando de que lado del hiperplano cae en el nuevo espacio de características. Debido a la variedad de núcleos posibles este simple planteo geométrico se corresponde a un igual variedad de estimadores no lineales en el espacio de partida.

Para separar entonces dichas muestras del origen se requiere resolver el problema cuadrático siguiente:

$$\min_{w \in H, \rho \in \Re, \xi \in \Re^N} \frac{1}{2}\|w\|^2 + \frac{1}{\upsilon m}\sum_{i=1}^{m} \xi_i - \rho \quad (18)$$

restringido por $\langle w \cdot \Phi(x_i) \rangle \geq \rho - \xi i \quad i = 1,2,...,m \quad \xi_i \geq 0$ (19)

El parámetro $\upsilon \in (0,1]$ como mencionáramos antes acota la cantidad de muestras que se deja fuera de al región S del espacio de partida.

Dado que las variables de relajación ξ_i son penalizadas en la función objetivo podemos esperar que si *w* y ρ resuelven el problema, tenemos que la función de decisión es:

$$f(x) = signo\left(\langle w \cdot \Phi(x) \rangle - \rho\right) \quad (20)$$

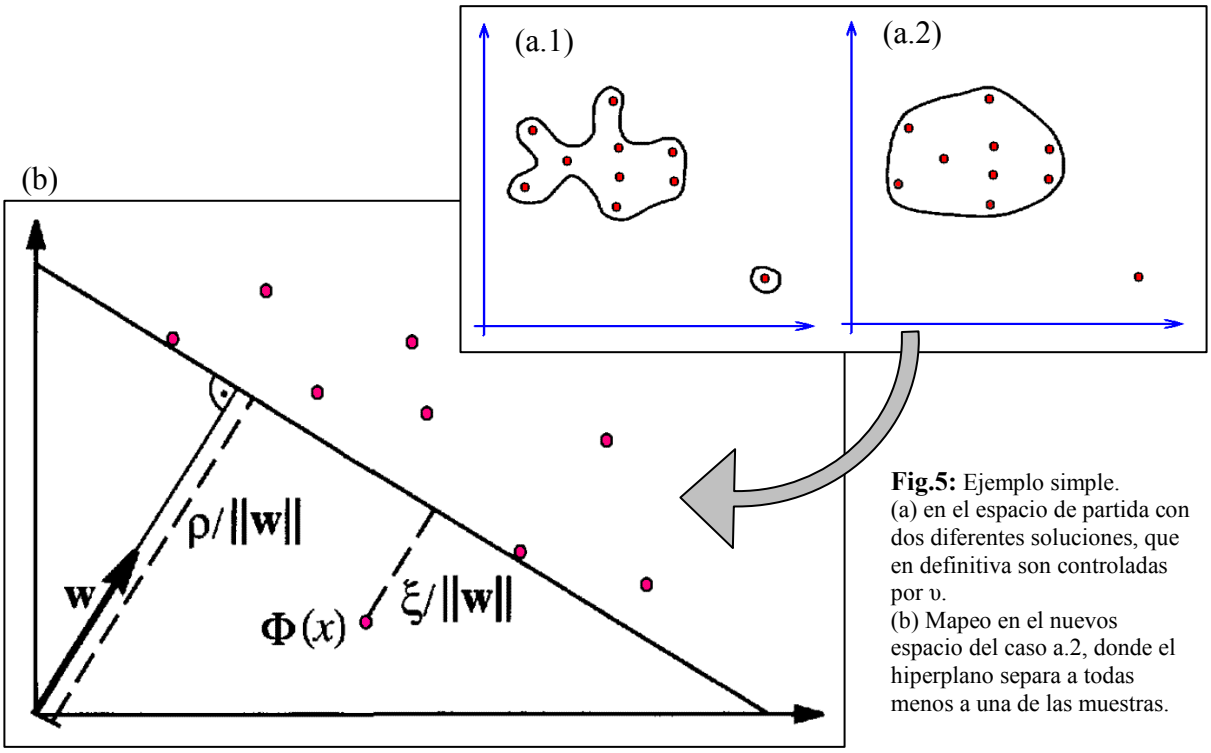


Fig.5: Ejemplo simple.
(a) en el espacio de partida con dos diferentes soluciones, que en definitiva son controladas por υ.
(b) Mapeo en el nuevos espacio del caso a.2, donde el hiperplano separa a todas menos a una de las muestras.

Esta será positiva (+1) para la mayoría de las muestras x_i pertenecientes al conjunto de entrenamiento, mientras el término regularizador, $\|w\|$ (8.2 de [1]) será todavía pequeño. Al igual que en υ-SVM, el compromiso entre estos dos parámetros es controlado por el parámetro υ (ver Fig.5).

Utilizando multiplicadores de Lagrange $\alpha_i, \beta_i \geq 0$ el Lagrangeano que nos permite llegar a la solución con las restricciones planteadas es:

$$L(w, \xi, \rho, \alpha, \beta) = \frac{1}{2}\|w\|^2 + \frac{1}{\upsilon m}\sum_{i=1}^{m}\xi_i - \rho - \sum_{i=1}^{m}\alpha_i\left(\langle w, \Phi(x_i)\rangle - \rho + \xi_i\right) - \sum_{i=1}^{m}\beta_i\xi_i \quad (21)$$

Igualando a cero las derivadas respecto a las variables primarias w, ξ y ρ nos queda:

$$w = \sum_{1}^{m}\alpha_i\Phi(x_i) \quad \text{y} \quad \alpha_i = \frac{1}{\upsilon m} - \beta_i \leq \frac{1}{\upsilon m}, \quad \sum_{1}^{m}\alpha_i = 1 \quad (22) \text{ y } (23),$$

lo que junto con los visto para *f(x)*, nos da: $f(x) = signo\left(\sum_{1}^{m}\alpha_i k(x_i, x) - \rho\right)$ (24).

Sustituyendo en el Lagrangeano obtenemos el problema cuadrático dual:

$$\min_{\alpha \in \Re^N} \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j k(x_i, x_j) \quad (25) \qquad \text{restringido por } 0 \le \alpha_i = \frac{1}{\upsilon m}, \quad \sum_i \alpha_i = 1 \quad (26)$$

Según [1], se muestra que las dos desigualdades en las restricciones originales (18 y 19) se vuelven igualdades si α_i y β_i son diferentes de cero, lo que implica que $0 \le \alpha_i = \frac{1}{\upsilon m}$.(27)

Por lo tanto podemos recuperar ρ considerando que para cada α_i que cumple lo anterior, el patrón correspondiente cumple con:

$$\rho = \langle \mathrm{w}, \Phi(\mathrm{x_i}) \rangle = \sum_j \alpha_j k(x_j, x_i) \quad (28)$$

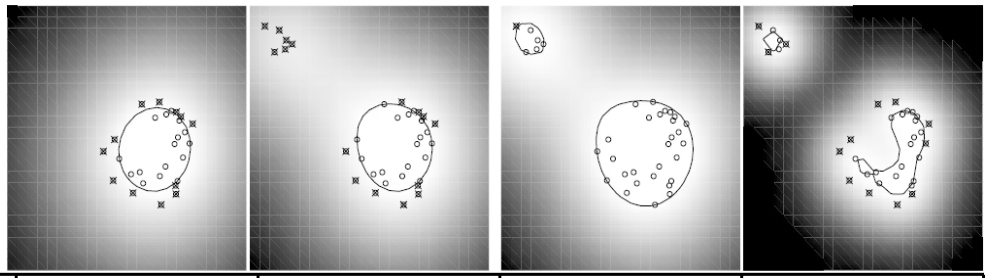
El parámetro υ

Para explicar su significado se enuncia un teorema expresado en [1] y demostrado en [3].

Proposición: Propiedad υ:

Asumiendo que la solución de (18) y (19) cumple que ρ ≠ 0, se cumple que:

i) υ es una cota superior de la fracción de outliers.
ii) υ es una cota inferior de la fracción de las muestras que son vectores de soporte.
iii) Si los datos X son generados independientemente de una distribución P(x), que no contiene componentes discretos, y demás el núcleo es analítico y no constante, entonces con probabilidad 1, asintóticamente, υ iguala tanto la fracción de muestras que son outliers y la fracción que son vectores de soporte.



ν, width c	0.5, 0.5	0.5, 0.5	0.1, 0.5	0.5, 0.1
frac. SVs/OLs	0.54, 0.43	0.59, 0.47	0.24, 0.03	0.65, 0.38
margin $\rho/\|w\|$	0.84	0.70	0.62	0.48

Fig.6: Influencia de υ y c=$\frac{1}{\gamma}$ (parámetro de RBF) (fuente [1]).

La figura 6, sacada de [1], nos muestra la influencia de los parámetros υ y γ . Los dos primeros cuadros muestran en dos problemas distintos como, al menos una fracción $1-\upsilon$ de todas las muestras se encuentra dentro de la región de interés.

El valor relativamente grande de υ ($\upsilon = 0.5$), hace que los puntos de la esquina superior izquierda no tengan casi influencia en el límite de decisión. Para valores menores de este parámetro, ya no pueden ser dejados por fuera de este límite (tercer cuadro). En forma

alternativa como nos muestra el cuarto cuadro, cambiando el valor del parámetro del núcleo (c en este caso, o γ en el nuestro), se puede lograr que el algoritmo tenga en cuenta dentro de su límite de decisión dichos puntos que serían outliers de otro modo, lo que cambia la forma de la función de decisión.

Notar que el cambio del parámetro del núcleo tiene una influencia similar al visto en el curso cuando se cambiaban los parámetros de las ventanas de Parzen al estimar las densidades de cada clase.

Como se dijo al principio la complejidad del clasificador depende de la cantidad de vectores de soporte, esto implica que (y se ha verificado en las pruebas) que al usar υ más grandes (mas cerca de 1), y al aumentar la cantidad de vectores de soporte en consecuencia, tanto la clasificación como el entrenamiento demoran más tiempo para un mismo set de datos.

Cómo se hace notar en 8.3 de [1], si suponemos que se usa un núcleo que puede ser normalizado como una densidad en el espacio de entrada, como es el caso de las Gaussianas, si se usa υ=1 en (26), las dos restricciones solo admiten la solución $\alpha_1 = ... = \alpha_m = 1/m$. Con esto (24) se reduce a un estimador por ventanas de Parzen de la densidad que nos interesa aunque según [4] el ρ que devuelve el algoritmo que permitiría a todos los puntos ser outliers según el teorema visto antes no puede ser usado por eso mismo. En este caso todos las muestras pasan a ser vectores de soporte.

La librería LIBSVM

Según sus autores [4] LIBSVM es un software sencillo, eficiente y de fácil uso para SVM tanto para problemas de clasificación como de regresión. De las pruebas que se han ehcho con la librería podemos decir que dichas aseveraciones, sobre todo en lo que repecta ala senciellez no son falsas.

Resuelve problemas de clasificación de C-SVM, nu-SVM y lo que más nos interesa problemas de SVM de una clase. Además contempla la resolución de problemas de regresión del tipo épsilon-SVM, y nu-SVM. También proporciona un instrumento modelo automático de selección para clasificación de C-SVM. Como parte de este trabajo se desarrolló una sencilla herramienta que permite hacer algo similar para oc-SVM, basado en búsqueda exhaustiva dentro de un rango elegido por el usuario.

LIBSVM resulte una versión escalada de (25) y (26), llegando a la función de decisión (24).

Si consideramos la siguiente forma genérica del problema dual (24 y 25), válida no solo para oc-SVM sino para C-SVM y Epsilon-SVM:

$$\min_{\alpha} \frac{1}{2}\alpha^T Q\alpha + \rho^T\alpha \text{ sujeto a } y^T\alpha = \Delta \text{ con } 0 \le \alpha_t \le C, \text{ t=1,….,m.} \quad (29)$$

y donde $y_t = \pm 1$ son las etiquetas de las muestras.

La dificultad al resolver (29) es la densidad de Q ya que Q_{ij} es en general diferente de cero. La librería LIBSVM utiliza el método de descomposición para sobreponerse a ese problema.

Este método modifica solamente un subconjunto de α por iteración. Este subconjunto, denominado conjunto de trabajo B, conduce a minimizar un sub-problema más pequeño en cada iteración.

Un caso extremo de este tipo de algoritmo es SMO (Sequential Minimal Optimzation – Platt,1998), donde B solo se reduce a dos elementos, por lo que en cada iteración se resuelve un problema simple de dos dimensiones sin necesidad de software de optimización.

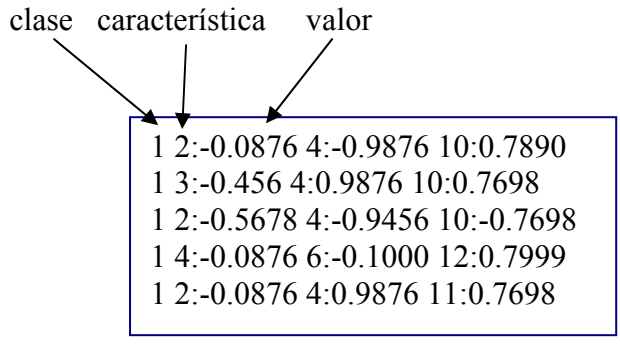
LIBSVM utiliza un método similar a la descomposición SMO propuesta por Fan et al. en 2005.

En [4] se muestran los detalles de este algoritmo y referencias al respecto. Nos concentramos ahora en comentar algunas guías para el uso de la librería y cómo comenzar con su uso, guía que también está disponible en [4] y que permite introducirse rápidamente a hacer pruebas a pesar de que hay otros problemas que pueden aparecer sobre todo a la hora de adaptar lo datos al formato de entrada o al elegir los parámetros υ y γ, ya que las herramientas suministradas para encontrar el mejor conjunto por búsqueda exhaustiva no se implementan para oc-SVM.

Para el presente trabajo se utilizó la versión Java de LIBSVM así como un port para Weka en el mismo lenguaje.

Al empezar a trabajar con LIBSVM los autores sugieren:

- Transformar los datos de interés al formato utilizado por LIBSVM. Este requiere que los datos estén representados en el llamado "Sparse format", donde los datos en formato texto son representados indicando la posición (o característica), seguido por ":" y luego el valor, solo incluyendo aquellos parámetros diferentes de cero.
-



- Escalar los datos. Para evitar que una característica tome mayor peso que las demás es conveniente escalar los datos ya sea para que tomen valores en el rango [-1;1] o [0;1]. Tener especial cuidado en que se escalen los datos de test con los coeficientes usados al escalar los datos de entrenamiento. Esto evita problemas numéricos al calcular los productos internos sobre todo en núcleos polinómicos, evitando problemas de overflow por ejemplo. LIBSVM en todas sus versiones Java y Python proveen una aplicación que resuelve esto (svm_scale).
- Se sugiere comenzar por utilizar núcleos del tipo RBF.

- Usar validación cruzada para seleccionar los parámetros (en nuestro caso υ y γ). El método de validación cruzada puede prevenir el problema de overfitting muchas veces asociado a SVM al elegir parámetros no adecuados. El método sugerido es la búsqueda exhaustiva, ya sea completa o comenzando por una grilla más grande y achacándolo la grilla cerca del mejor valor obtenido.

- Entrenar con los parámetros hallados anteriormente.

- Testear sobre los datos de test.

LIBSVM requiere que los datos estén representados como un vector de números reales, por lo que si los atributos no son numéricos los mismos deben ser convertidos a un formato adecuado. Se recomienda además utilizar n números para representar un atributo de n características posibles. Por ejemplo para representar las clases 1,2 y 3, usar 001, 010 y 100. Esto hace en general que el algoritmo obtenga resultados de mejor forma que si se etiquetan los datos directamente.

La salida nos da el valor de ρ, así como los vectores de soporte con su α asociado.

Para una explicación de los parámetros a considerar en LIBSVM (ingresados vía svm_train) ver el Anexo A.

Pruebas con la librería LIBSVM y algunas aplicaciones.

Más allá de las primeras pruebas para familiarizarse con los parámetros de la librería (en primera instancia con Weka), lo primero que fue necesario hacer fue crear el código necesario que permitiera convertir datos en formato texto separado por comas en el formato Sparse admitido por la librería.

Pruebas con datos de la base USPS: detección dedigitos “no 0”

Para probar la librería LIBSVM y en particular el código para hacer búsqueda exhaustiva de parámetros se tomaron los datos de la base USPS de dígitos escritos a mano. Debido a que la base viene etiquetada pensando en clasificación multi clase, para poder probar y comparar resultados con los presentados en [4], se modificaron los datos para sacar todas las muestras del carácter 0, como muestras de entrenamiento, y se comparó con las otras muestras (de 1 a 9) de forma de analizar si se llegaban a las mismas conclusiones que se postulaban en [4].

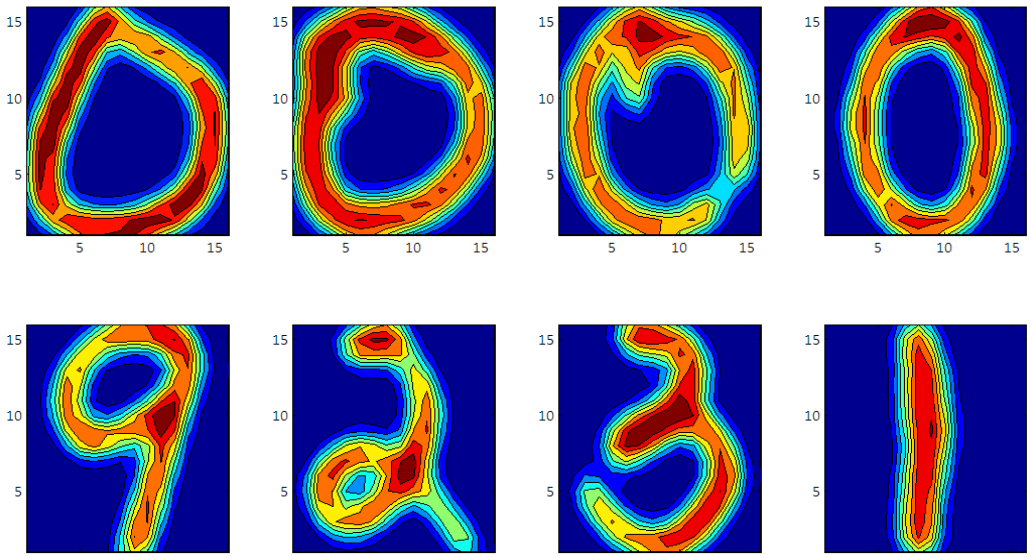


Fig.7: Algunos ejemplos de la base USPS. Se entrenó con los ‘0’, y se vio la habilidad de identificar los ‘no 0’

Entre estas conclusiones está el comparar los resultados tanto en cantidad de vectores de soporte de diferentes valores de υ (en este caso se presentarán para valores 0.05 y 0.5).

Se confirma que realmente la cantidad de vectores de soporte queda determinada en esencia por el valor de υ, y que en el caso de υ=0.5, el algoritmo es capaz de detectar como outliers al 100% de las muestras de test que no eran 0, aunque esto implica que alrededor del 50% de las muestras que son cero son catalogadas como outliers.

Búsqueda exhaustiva

Se implementó una variación del programa svm_train, agregando código que permite ingresar el rango de υ y γ, así como el resto de los parámetros usuales de train_svm (ver anexo A).

Este programa recorre todos los valores del rango ingresado, chequeando el error de validación cruzada (explícitamente 1-error), quedándose con los parámetros υ y γ de menor

error en la grilla. En casi todos los casos, dado el cómo juega el parámetro υ en el algoritmo, en la práctica solo se hizo búsqueda exhaustiva en γ, aunque está contemplado hacerlo en ambos parámetros en el programa.

En el primer ejemplo, fijando υ=0.5 se ingresa un rango de 0.001 a 0.9 de γ, chequeado 30 valores de γ, y con una validación cruzada de 19 subconjuntos (el set de entrenamiento tiene poco más de 1900 elementos, con lo que cada subgrupo queda de unos 100 elementos).

Tabla 1: Grid search variando γ entre 0.0001 y 0.9. Se ve que los mejores resultados están en los valores ceranos a 0.0001 y 0.003

υ	gamma	% correctos
0.50	0.000100	50.083752
0.50	0.031134	49.162479
0.50	0.062169	45.226131
0.50	0.093203	36.599665
0.50	0.124238	25.795645
0.50	0.155272	15.996650
0.50	0.186307	8.458961
0.50	0.217341	4.271357
0.50	0.248376	2.428811
0.50	0.279410	0.670017
0.50	0.310445	0.167504
0.50	0.341479	0.083752
0.50	0.372514	-
0.50	0.403548	-
0.50	0.434583	-
0.50	0.465617	-
0.50	0.496652	-
0.50	0.527686	-
0.50	0.558721	-
0.50	0.589755	-
0.50	0.620790	-
0.50	0.651824	-
0.50	0.682859	-
0.50	0.713893	-
0.50	0.744928	-
0.50	0.775962	-
0.50	0.806997	-
0.50	0.838031	-
0.50	0.869066	-
0.50	0.900100	-

Tabla 2: Grid search luego de otras corridas que aproximan al mejor valor. Γ entre 0.008 y 0.01.

υ	gamma	% correctos
0.50	0.008000	49.664992
0.50	0.008069	49.497487
0.50	0.008138	50.082750
0.50	0.008207	49.748744
0.50	0.008276	49.832496
0.50	0.008345	49.664992
0.50	0.008414	49.832496
0.50	0.008483	49.497487
0.50	0.008552	49.748744
0.50	0.008621	49.664992
0.50	0.008690	49.581240
0.50	0.008759	50.083752
0.50	0.008828	49.832496
0.50	0.008897	49.497487
0.50	0.008966	49.916248
0.50	0.009034	49.581240
0.50	0.009103	49.497487
0.50	0.009172	49.916248
0.50	0.009241	50.000000
0.50	0.009310	49.581240
0.50	0.009379	49.832496
0.50	0.009448	49.497487
0.50	0.009517	49.916248
0.50	0.009586	49.664992
0.50	0.009655	49.581240
0.50	0.009724	49.664992
0.50	0.009793	49.497487
0.50	0.009862	49.748744
0.50	0.009931	49.413735
0.50	0.010000	49.329983

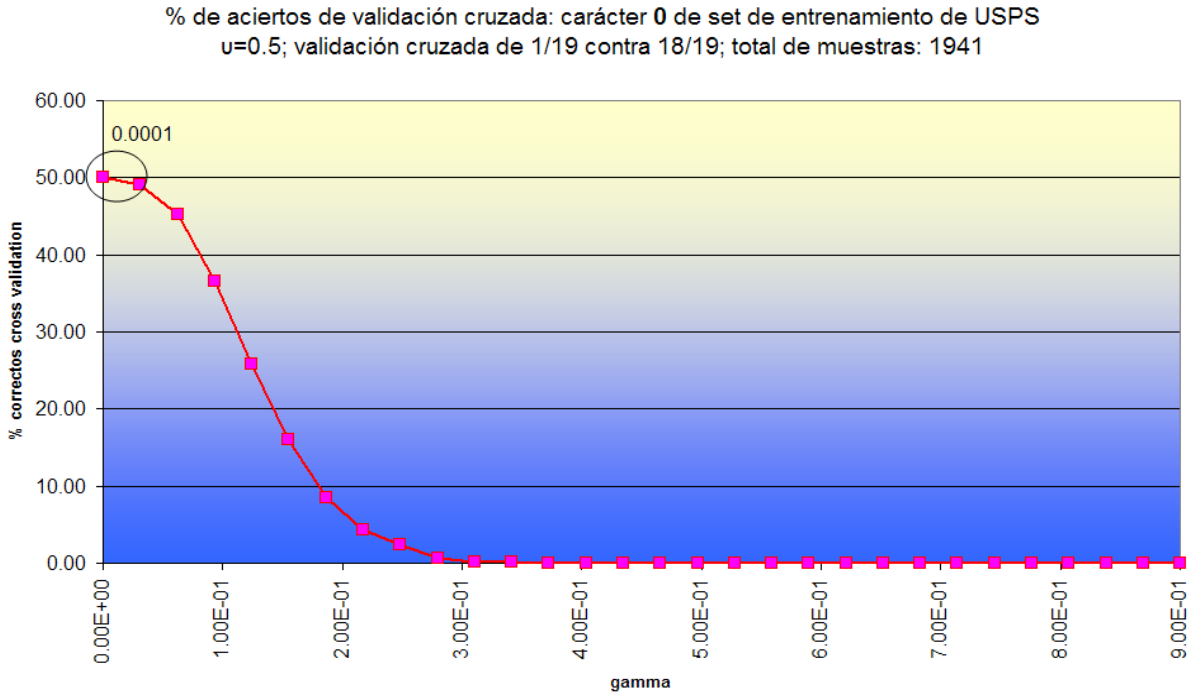


Fig.8: Porcentaje de '0s' bien clasificados en la validación cruzada de entrenamiento para los valores de 0.0001 a 0.9

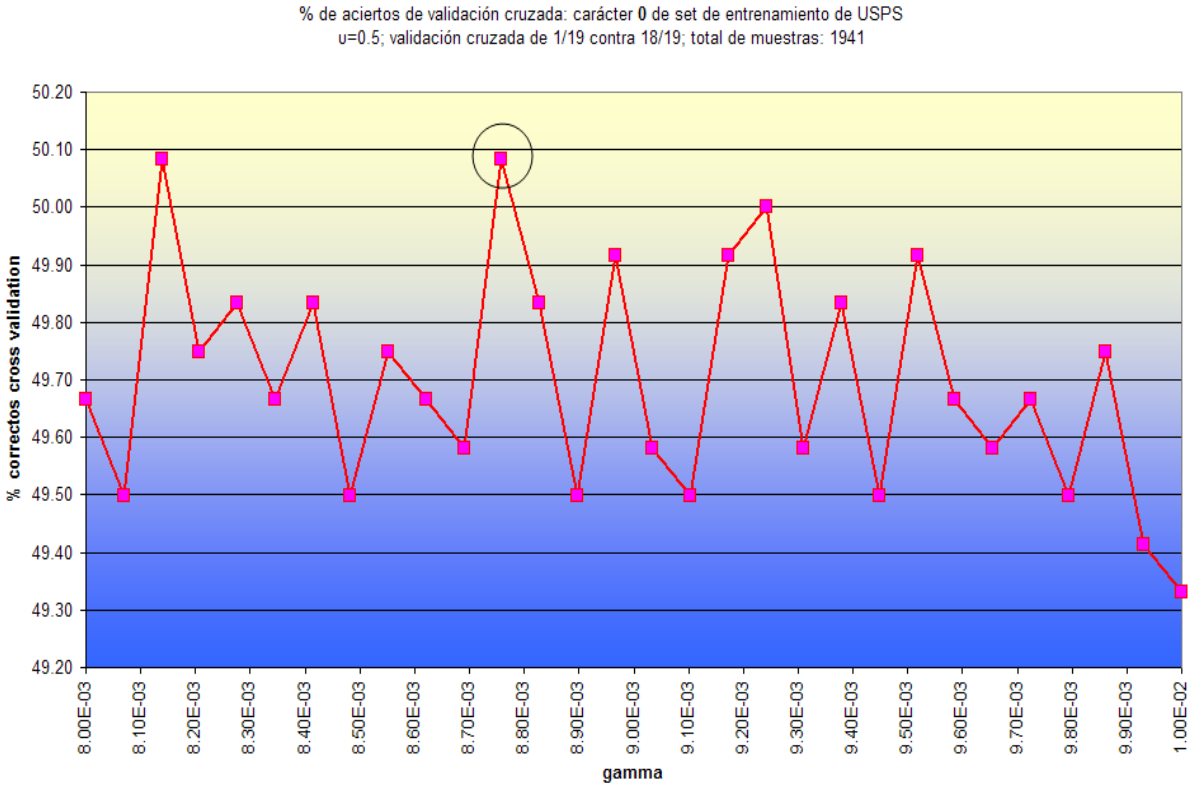


Fig.9: Idem figura 8, para valores de γ entre 0.008 y 0.01.

El método de variación de γ utilizado primero lo hace variar en forma amplia, y reduciendo luego el rango. Se vio que el valor de menor error de validación cruzada fue muy cercano al mencionado en [4] como de uso habitual en este set de datos. De todas formas se vio que para un rango pequeño de variación de γ, el hecho de que en cada variación se elija al azar el conjunto de validación cruzada hace que llegado un punto, diferentes valores de γ resulten valores similares de error, teniendo en medio valores donde el error empeora. Se verifica que la cantidad de '0s' considerados outliers siempre está en el orden del porcentaje determinado por υ, siendo para el mejor valor de γ hallado lo valores siguientes:

% val.cruzada: 50.083753%
γ: 0.00875862
total_sv: 600
rho: 226.87529482

Notar que la cantidad de vectores de soporte es del orden del 50% de los 1194 muestras totales tal cual lo viéramos anteriormente.

Con los parámetros hallados y el archivo de modelo obtenido, se ejecutó la aplicación svm_predict que aplica el método de clasificación, sobre los vectores de test, que en este caso eran todo el resto de los dígitos del set de entrenamiento de USPS. En este caso el clasificador clasificó correctamente al 100% de los datos como outliers o como 'no ceros'.

A continuación repetimos los resultados arriba mencionados pero para υ=0.05. En este caso el nivel de dígitos 'no cero' identificados como ouliers llegó al 81.35% (siendo el resto falsos positivos). Es decir que el bajar υ, deja menos '0s'como outliers, pero a la vez se hace más

tolerante a dígitos parecidos al cero. En la figura 10 se muestran solo dos ejemplos de 'no ceros' tomados como tales. Se ve que si uno es 'tolerante' el algoritmo puede confundir estos con valores similares a ceros.

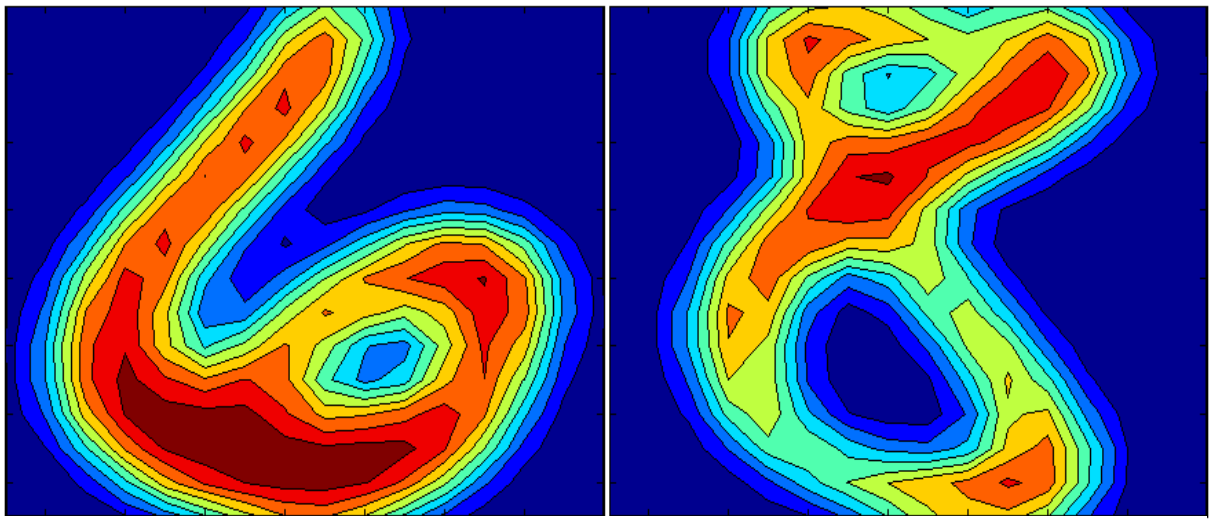


Fig.10: Dos ejemplos de patrones incorrectamente clasificados como cero al bajar υ de 0.5 a 0.05, haciendo "más tolerante" al clasificador a los outliers pero teniendo menos falsos negativos.

Esto nos permite ver que jugando con υ uno puede determinar si se clasifican las muestras de la propia clase mejor, teniendo algunos falsos positivos, o si se desea que todo outlier sea detectado, lo que aumenta la posibilidad de falsos negativos (integrantes de la clase tomados como outliers).

Pruebas con datos de la base USPS: detección muestras mal etiquetadas

Este ejemplo sugerido en [1], permite ver como con oc-SVM es posible detectar outliers dentro de la propia muestra de test, si se manejan los datos adecuadamente.

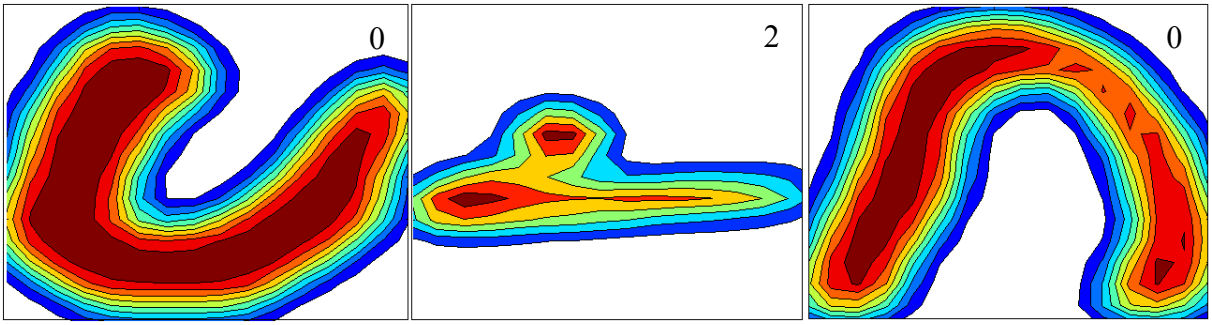
Previo a correr el algoritmo se sustituyeron las etiquetas numéricas por etiquetas sugeridas en el capitulo de LIBSVM anterior, es decir agregando una característica por cada categoría (por ejemplo 4 = 0000001000).

Con esto todas las muestras de cada categoría tienen valores comunes en las primeras dimensiones, asociadas a sus etiquetas. Se entrena el algoritmo y luego se corre el clasificador svm_predict para clasificar. Al usar υ=0.05, alrededor de 94.5% de las muestras son consideradas correctas, siendo las restantes las consideradas outliers.

En la fig.11 se tomaron algunos de los ejemplos con sus etiquetas, pero relevando todos los casos se vio que la gran mayoría eran del estilo, aunque algunas a priori deberían ser clasificables presentan algún tipo de distorsión.

Otra vez, el parámetro υ puede resolver el tema ya que bajándolo un poco (0.03), algunos de los casos dudosos dejaron de serlo. Otros como los mostrados requieren ser demasiado tolerantes.

Fig.11: Algunos ejemplos de dígitos mal etiquetados o no clasificables de USPS encontrados con oc_SVM



Datos de tráfico de una radio base GSM:

Se utilizaron para esta prueba datos de una radio base GSM ubicada en una zona balnearia del periodo 11/09/08 a 25/01/09.

De ese periodo, se extrajeron los contadores más relevantes, como ser tráfico total cursado, MHTIME, intentos originados y llamadas cursadas (propias y de otras radiobases). Se eligió en una primera instancia como dimensión extra la hora del día en una escala de 0 a 1 (0=00:00 y 0.96=23:00).

Esto es importante porque iguales valores de tráfico no significan lo mismo si ocurren a las 03:00 que si ocurren a las 16:00.

Por ser una primera prueba no se hicieron más adiciones pero es de esperarse que se agregue una nueva característica que indique por ejemplo el día de la semana.

La idea en una primera instancia era entrenar con un cierto periodo de tiempo, en este caso con las cuatro semanas que van del 02/11/08 al 30/11/08. Se testearon datos de diferentes periodos anteriores y posteriores. Lo que cualitativamente se esperaba encontrar en este caso, en particular era tener niveles de coincidencia en los meses de setiembre y octubre similares a los de entrenamiento, ya que en general se considera que noviembre, para esta zona, es un punto de quiebre en el comportamiento invernal-temporada.

Se esperaba además que en los periodos posteriores se tuvieran apartamientos importantes y crecientes ya que se incluían la segunda semana de diciembre, las dos semanas de diciembre que incluyen (en la última) a Navidad y la tercera de enero, donde se está en alta temporada.

Se aplicó la herramienta de grid search, para un un=0.05 (5%), haciendo chequeos de validación cruzada de 1 a 4 (parámetro –v 4), ya que pretendíamos que se tuviera la mayor variedad de horas y días en cada subconjunto, y al tener 4 semanas y ser la elección aleatoria, parecía un buen número.

Se puede ver en la tabla que sigue que los resultados cualitativamente coincidieron con nuestras expectativas y en principio para la aplicación que se tiene pensado aplicar esta técnica que es la de poder automatizar la revisión de radio bases en busca de apartamientos sin tener que llegar a que se vea congestión es más que útil o al menos alentadora.

Tabla 3: Resultados radiobase GSM	
Periodo de entrenamiento: 02/11/08 a 30/11/08. nu=0.05 γ_opt = 7.54e-5 Total Vect. Sop. : 34 de 672 %correctos valida. cruz. al elegir γ_opt = 97.62%	
Periodo	**% de muestras dentro de la clase**
14/09/08 a 27/09/08	99.12%
20/10/08 a 26/10/08	98.80%
02/11/08 a 30/11/08	97.47%
08/12/08 a 14/12/08	29.76%
15/12/08 a 28/12/08	15.02%
12/01/09 a 18/01/09	0.00%

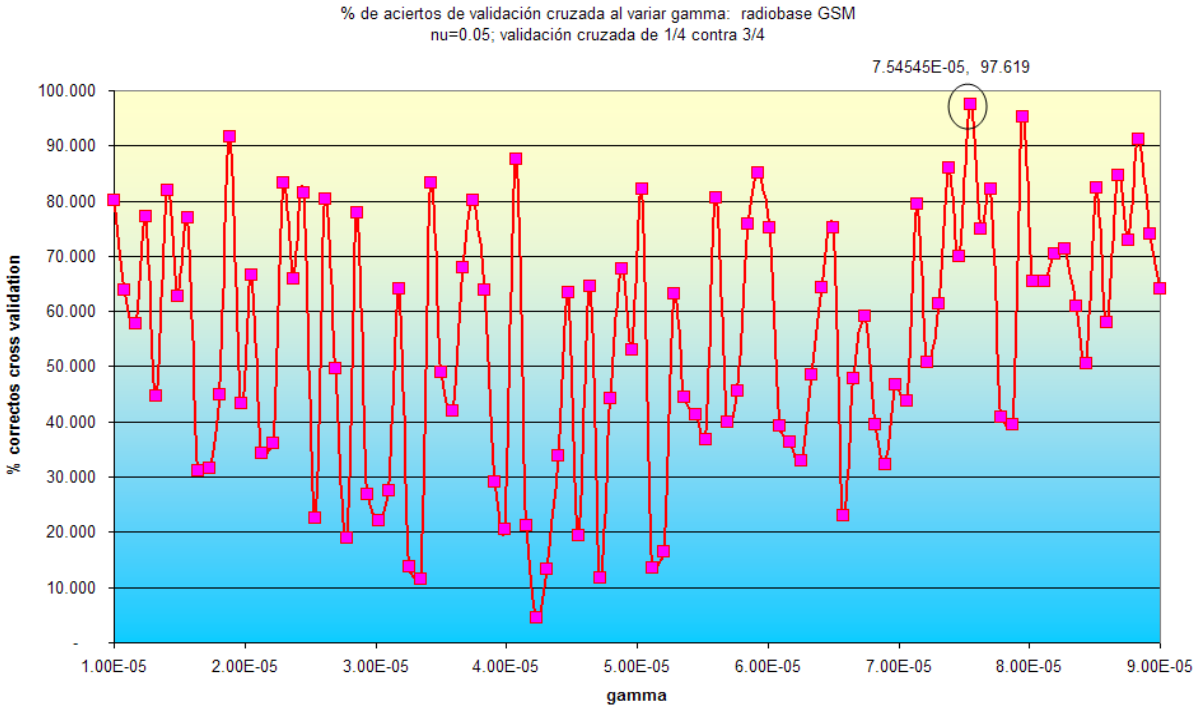


Fig.12: Búsqueda del mejor γ para el caso de radiobase GSM, nu=0.05.

Notar que se cumple que el porcentaje de no-outliers supera el 97% (bastaría con que supere el 95%), y que los vectores de soporte son también del orden del 5% de las muestras. La fig.12, muestra los valores de acierto en al validación cruzada en una de las corridas finales en busca del mejor γ.

Al igual que en el caso de USPS, en la última fase y por la variabilidad en los patrones usados en cada validación cruzada hay varios γ que difieren muy poco que dan el menor error entre corrida y corrida del grid search.
Puede llamar la atención lo valores menores de error previo a noviembre, ero eso significa que hay menos apartamientos que en algunas horas de noviembre sobre tdo al final donde los valores son más comparables a los de diciembre.

Datos de tráfico de ruta luego de cambio de configuración:

Otro ejemplo donde se trabajó aplicando LIBSVM fue con datos de tráfico de una ruta interurbana luego de un cambio de configuración que afectó el tráfico que cursaba, por enrutarse el tráfico de otra ruta en forma adicional al que ya tenía, tráfico cuyo perfil era diferente al que tenía hasta ese momento, ya que suele tener picos en la noche y no sobre las 11:00.

Se entrenó con los datos de la semana del 19 al 25 de enero, ya que el cambio empezó a las 00:00 del 26/01. Se usaron dos valores de υ, (0.05 y 0.1), obteniéndose un valores de γ "óptimos" de 4.96e-5 y 2.54e-5 respectivamente.

En la tabla 4 se ve la hora 23:00 de todo el periodo, indicándose para ambos valores de υ si el clasificador lo cataloga como perteneciente a la clase "normal" (+1) o como outlier (-1).

Tabla 4: Una de las horas con cambios más fuertes luego del cambio de configuración del 25/01.

Fecha	Hora	nu=0.05	nu=0.1	Intentos	Nivel Trafico	Trafico Entrante	Trafico Saliente	Repuestas B	Carga (%)
2009/01/19	23:00	1.0	1.0	373	30.23	13.68	16.55	267	24
2009/01/20	23:00	1.0	1.0	362	29.73	12.71	17.02	281	23
2009/01/21	23:00	1.0	1.0	377	30.27	13.35	16.92	293	24
2009/01/22	23:00	1.0	1.0	425	26.92	10.08	16.84	312	21
2009/01/23	23:00	1.0	1.0	438	21.84	8.47	13.37	317	17
2009/01/24	23:00	1.0	-1.0	311	14.32	5.47	8.85	219	11
2009/01/25	23:00	1.0	1.0	400	22.81	9.05	13.76	300	18
2009/01/26	23:00	1.0	-1.0	341	47.77	32.95	14.82	259	37
2009/01/27	23:00	1.0	-1.0	355	44.58	29.74	14.84	286	35
2009/01/28	23:00	1.0	-1.0	299	42.81	31.66	11.15	250	33
2009/01/29	23:00	1.0	-1.0	354	44.06	31.37	12.69	277	34
2009/01/30	23:00	1.0	1.0	377	37.47	24.77	12.69	296	29
2009/01/31	23:00	1.0	1.0	427	38.06	23.26	14.81	316	30
2009/02/01	23:00	1.0	-1.0	437	54.47	37.11	17.35	338	43
2009/02/02	23:00	1.0	-1.0	356	48.13	32.37	15.76	289	38
2009/02/03	23:00	1.0	-1.0	403	49.82	33.02	16.81	336	39
2009/02/04	23:00	1.0	-1.0	411	41.37	29.45	11.92	325	32
2009/02/05	23:00	1.0	-1.0	351	43.15	28.77	14.37	287	34

En la Tabla 5, se ven los datos del día 05/02, uno de los más afectados respecto a los datos previos al cambio. Ver que para el caso de υ=0.05, solo los picos de las 10:00 y 11:00 se apartan del comportamiento normal, hecho que al ver el resto de los días en general afecta a la mayoría de los picos que ocurren a esa hora.

Al usar υ = 0.1, más horas son vistas como "outliers", detectándose no solo horas que tienen más tráfico sino casos como el del 24/01, donde se lo marca por tener menor nivel al normal sobre todo en la característica % de carga.

Este valor de υ o uno intermedio puede llegar a ser un buen compromiso en este caso. El hecho de que además se marquen datos no solo de valor superior sino inferior es una característica interesante a la hora de analizar los recursos de la red al ser usados en este tipo de datos.

Tabla 5: Diferencias en la clasificación para diferentes υ, ver que con υ=0.05, solo las horas que más apartan son vistas como "outliers"

Fecha	Hora	nu=0.05	nu=0.1	Intentos	Nivel Trafico	Trafico Entrante	Trafico Saliente	Repuestas B	Carga (%)
2009/02/05	00:00	1.0	1.0	218	23.6230	16.2787	7.3443	160	18
2009/02/05	01:00	1.0	1.0	106	15.0820	9.2459	5.8361	96	12
2009/02/05	02:00	1.0	1.0	59	11.0984	5.8689	5.2295	52	9
2009/02/05	03:00	1.0	1.0	80	8.9836	4.3770	4.6066	72	7
2009/02/05	04:00	1.0	1.0	53	6.2787	1.6066	4.6721	42	5
2009/02/05	05:00	1.0	1.0	58	5.3443	1.2787	4.0656	51	4
2009/02/05	06:00	1.0	1.0	98	6.0164	1.5246	4.4918	84	5
2009/02/05	07:00	1.0	1.0	239	9.5246	3.6230	5.9016	204	7
2009/02/05	08:00	1.0	1.0	837	29.1639	12.8033	16.3607	608	23
2009/02/05	09:00	1.0	1.0	1740	55.5574	29.2623	26.2951	1181	43
2009/02/05	10:00	**-1.0**	**-1.0**	**2460**	**76.3607**	**38.7377**	**37.6230**	**1483**	**60**
2009/02/05	11:00	**-1.0**	**-1.0**	**2778**	**81.4262**	**38.3443**	**43.0820**	**1661**	**64**
2009/02/05	12:00	1.0	-1.0	2076	66.8197	33.9836	32.8361	1307	52
2009/02/05	13:00	1.0	1.0	1377	56.8689	30.9344	25.9344	988	44
2009/02/05	14:00	1.0	1.0	1453	58.1639	32.8361	25.3279	1017	45
2009/02/05	15:00	1.0	-1.0	1677	62.4426	35.3770	27.0656	1160	49
2009/02/05	16:00	1.0	-1.0	1572	66.4262	35.9180	30.5082	1154	52
2009/02/05	17:00	1.0	-1.0	1500	65.7377	36.1967	29.5410	1095	51
2009/02/05	18:00	1.0	-1.0	1330	62.8033	33.0984	29.7049	939	49
2009/02/05	19:00	1.0	-1.0	1059	58.7705	33.7869	24.9836	778	46
2009/02/05	20:00	1.0	-1.0	993	57.7869	31.5738	26.2131	717	45
2009/02/05	21:00	1.0	-1.0	1146	68.9016	39.4590	29.4426	824	54
2009/02/05	22:00	1.0	-1.0	697	63.2623	42.3279	20.9344	544	49
2009/02/05	23:00	1.0	-1.0	351	43.1452	28.7742	14.3710	287	34

Conclusiones

Se presentó el método de SVM aplicado a una clase, primero presentando en forma general las nociones de SVM, y luego detallando un poco más cómo se resuelve el problema de SVM de una clase.

Describimos las generalidades de la librería LIBSVM utilizada para hacer pruebas.

Dividimos las mismas en dos, por un lado y como forma de validar el código realizado para hacer búsqueda exhaustiva y entender los parámetros comparándolos con la literatura, utilizamos la base USPS de dígitos escritos a mano. Se entrenó el algoritmo con los datos 0 de la base de entrenamiento USPS, testeando con el resto de los datos no cero, de l set de entrenamiento

Los resultados obtenidos fueron comparables, y estimando con la herramienta de búsqueda exhaustiva que el mejor γ obtenido era muy cercano que el enunciado en [1] como el que da mejores resultados en la mencionada base de muestras.

Por otro lado se hicieron pruebas con datos reales provenientes de medidas de tráfico. En a particular nos concentramos en ver si era posible usar el algoritmo para detectar datos de tráfico que se aparten del comportamiento de un cierto conjunto de medidas usadas como modelo, con miras a su uso en una aplicación real que facilite el análisis de datos de tráfico.

Se obtuvieron resultados interesantes, en particular resultó muy ilustrativo el ejemplo sobre una ruta que había experimentado un cambio de configuración, lo que provocó además de una aumento del tráfico global (que no causó problemas de congestión), un cambio de perfil, ya que se agregó tráfico con un perfil nocturno a uno que era más del tipo comercial diurno. El algoritmo fue capaz de "marcar" las horas afectadas así como otras que se apartan del comportamiento común por tener valores menores a los estándar.

Entre los datos relacionados con el algoritmo y la librería en sí, está el hecho de la influencia de los parámetros en el resultado final, siendo estos bastante distintos si los mismos no son adecuados a lo que se busca.

En particular el parámetro υ, se nota un importante enlentecimiento del entrenamiento al acercarse a 1 (confirmando lo expresado en 8.7 de [1]).

Para valores medios de υ (0.5) el algoritmo tiene poca tolerancia a outliers teniendo un alto nivel de falsos negativos. Ver por ejemplo el caso de USPS donde hasta un 50% de los "0s" queda por fuera y es identificado como outlier.

Al hacer bajar υ a niveles de por ejemplo 0.05, se hace aumentar el nivel de falsos positivos (muestras dadas como de la clase), lo que en definitiva se traduce en que si una muestra se identifica como outlier, uno puede estar más "seguro" de que lo es, que con valores más altos de υ. Estos valores menores de υ implican como se vio, menores tiempos de entrenamiento y clasificación ya que implican menor cantidad de vectores de soporte.

Es manejo de υ se vio útil además en el segundo ejemplo de USPS, donde dentro de ciertos límites, uno puede decidir qué tan mal etiquetados o no clasificados se admitía que fueran los datos de la base de test de USPS.

El tema de sensibilidad a los parámetros se pudo apreciar al hacer búsqueda exhaustiva del mejor γ, dado el set de datos. Lo que se verificó en todos los casos, es el hecho de que cuando uno está lejos del valor óptimo el error empeora y mejora al acercarse, como es obvio, pero una vez que se afina más la grilla y se acerca al mejor valor, valores cercanos de γ pueden producir diferencias comparativamente notables en el error de la validación cruzada ("oscilando" alrededor del mejor).

Esta diferencia, creemos se explica, por el hecho de que en cada cambio de γ en la búsqueda se hace un nuevo sorteo de validación cruzada por LIBSVM, lo que hace que llegado el momento, para diversos gamma se entrena y chequea diferentes subconjuntos en la validación cruzada, lo que genera pequeñas diferencias a la hora de analizar el error. Quizás para esto se deba reformular un poco más el algoritmo, de forma que la propia validación cruzada dentro de sí haga la búsqueda exhaustiva para todos los γ del rango, y luego se proceda a un nuevo sorteo de validación cruzada. Es decir sortear los subconjuntos, y variar en todos los γ, al revés de lo que sucede ahora.

Si bien no se hicieron comparaciones explícitas con otros algoritmos como por ejemplo el mencionado y visto en el curso de ventanas de Parzen, teniendo en cuenta que el mismo es asimilable al caso de oc-SVM con υ=1, nos queda claro que dicho algoritmo no tiene en principio la flexibilidad y sí adolece de los inconvenientes a la hora de tener que considerar todas las muestras de entrenamiento a la hora de clasificar.

Se relevaron a titulo informativo otras técnicas similares como las mostradas en [8], [9] y [10], aunque por falta de tiempo y sobre todo de librerías que permitieran probarlos, no se estudiaron más fondo.

Trabajos futuros

Dado que los resultados cualitativos obtenidos en el estudio del apartamiento en el comportamiento de las medidas de tráfico fueron positivos, se planea seguir afinando la técnica sobre todo de cara a:

- poder obtener los datos directamente de un servidor de base de datos sin tener que hacer todo el proceso de conversión y escalado en forma separada.
- poder afinar los parámetros y seleccionar los campos relevantes en este tipo y otro tipo de datos como ser datos celular o datos en DSLAMs, para esto se deberá definir qué se busca.
- poder automatizar el camino de vuelta a la detección de ouliers (hecho a mano en el ejemplo de tráfico), es decir una vez se detecta uno, identificar claramente quién y cómo se apartó, de cara al manejo de múltiples nodos independientes y la identificación no solo de que ciertas horas se apartaron de un comportamiento de referencia, sino saber explícitamente qué horas y/o contadores causaron ese apartamiento.

Si bien en principio parece excesivo utilizar esta técnica teniendo todas las herramientas clásicas de teoría de tráfico, debe aclararse que lo que se pretende es poder llegar a una herramienta que separa la paja del trigo a la hora de analizar los datos en forma global, dada la cada vez más gran cantidad de nodos generadores de datos y así concentrarse solo en los datos de interés sin tener que esperar que se generen problemas, o por otro lado ver en forma global como afectan ciertos cambios (promociones, configuraciones, etc) a los datos de ciertos nodos generadores de datos.

Anexo A
A.1-Parámetros de svm_train.

Parámetros de LIBSVM (svm_train)

```
svm-train [opciones] archivo de entrenamiento [archivo_de_modelo]
options:
  -s tipo de SVM : selecciona el tipo de SVM a usar (por defecto 0)
        0 -- C-SVC
        1 -- nu-SVC
        2 -  SVM de una clase
        3 -- epsilon-SVR (regresión por vectores de soporte)
        4 -- nu-SVR (regresión por vectores de soporte)
  -t tipo de núcleo: selecciona el tipo denúcleo, (por defecto 2)
        0 -- lineal: u'*v
        1 -- polinómico: (γ*u'*v + coef0)^grado
        2 -- RBF: exp(-γ*|u-v|^2)
        3 -- sigmoide: tanh(γ*u'*v + coef0)
        4 - núcleo precomputado (el archivo de entrenamiento tiene los
valores precomputados)

  -d grado : selecciona el grado del núcleo (por defecto 3)
  -g γ : selecciona γ en función núcleo (por defecto 1/máximo valor de las
caracteriticas)

  -r coef0 : selecciona coef0 de la función núcleo (por defecto 0)
  -c cost0 : selecciona parámetro C de costo de C-SVC, epsilon-SVR, y nu-
SVR (por defecto 1)

  -n nu : selecciona el parámetro nu de nu-SVC, one-class SVM, y nu-SVR
(por defecto 0.5)

  -p epsilon : selecciona el epsilon en la función de pérdida de epsilon-
SVR (por defecto 0.1)

  -m cachesize : selecciona el caché de datos en memoria en MB (por defecto
100)
  -e epsilon : selecciona la tolerancia del criterio de terminación (por
defecto 0.001)

  -h reducción: determina si se usa o no la eurística de redución
(shrinking) al resolver el problema de opcytimización de SVM, 0 o 1 (por
defecto 1)

  -b estimaciones de probabilida: determina se entrena un modelo SVC o SVR
para estimacion de probabilidad, 0 o 1 (por defecto 0)

  -wi pesos: selecciona el parñametro C de la clase i a  peso*C en C-SVC
(por defecto 1 en todos)

  -v n: validación cruzada en n grupos
```

A.2-Parámetros de gris_svm.

Modificación de svm_train, que hace búsqueda exhaustiva y se queda con el conjunto un, γ de menor error de validación cruzada.

```
Uso: scsvm_grid [opciones] archivo_entrenamiento [archivo_modelo_destino]
opciones:

-t nucleo a usar: por defecto es RBF (2)
        0 -- lineal: u'*v
        1 -- polinómico: (γ*u'*v + coef0)^degree
        2 -- RBF: exp(-γ*|u-v|^2)
        3 -- sigmoide: tanh(γ*u'*v + coef0)
        4  --  núcleo  pre  computado  (los  valores  se  ingresan  en
archivo_entrenamiento)

-d grado : Grado de función núcleo si corresponde (por defecto 3)

-Gi γ ini: inicio rango de γ de la función núcleo para grid search
(dflt.:0.001)

-Gs γ steps: entero:cantidad de pasos entre ini y fin (dflt.:2)

-Gf γ fin: fin de rango de γ de la función núcleo para grid search
(dflt.:0.001)

-Xb γ base: base usada en caso de que se elija la opción X (el rango es
Gx^Gi:Gx^Gf / dflt.:2)

-Xi : Si aparece indica que Gi y Gf son rangos de exponentes al que se
eleva a Gx.

-Xf : Si aparece indica que Gi y Gf son rangos de exponentes al que se
eleva a Gx.

-Ni nu ini: inicio rango de nu de la función núcleo para grid search (entre
0 y 1)(dflt.:0.001)

-Ns nu steps: entero:cantidad de pasos entre ini y fin (1 solo toma Ni)
(dflt.:2)

-Nf nu fin: fin de rango de nu de la función núcleo para grid search (entre
0 y 1) (dflt.:0.001)

-r coef0 : coef0 de la función núcleo (por defecto 0)

-v n: validación cruzada en subconjuntos de an (por defecto n=10)

-m cachesize : tamaño memoria cache en MB (por defecto 100)

-e epsilon : tolerancia del criterio determinación (por defecto 0.001)

-h shrinking: uso o no de la eurística de reducción, 0 or 1 (por defecto 1)
```

En todos los casos se debe tener java, y hacer referencia a la librería libsvm.jar:

java -classpath ruta_clases;ruta_libsvm\libsvm.jar -Xmx1024m [grid_svm o trainsvm] [parámetros]

Referencias:

[1] Learning with Kernels; Support Vector Machines, Regularization, Optimization, and Beyond; Bernhard Scholkopf
Alexander J. Smola. The MIT Press; Cambridge, Massachusetts;London, England; ISBN 0-262-19475-9 (alk. paper)

[2] One-Class SVMs for Document Classification; Larry M. Manevitz manevitz@cs.haifa.ac.il; Malik Yousef yousef@cs.haifa.ac.il;
Department of Computer Science; University of Haifa; Haifa 31905 Israel; *J* Journal of Machine Learning Research 2 (2001) 139-154 Submitted3/01; Published12/01

[3] Estimating the support of a high-dimensional distribution.; B. Scholkopf, J. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. TR 87, Microsoft Research, Redmond, WA, 1999. htrp://www.research.microsoft.com/scripts/pubs/view.asp?TR_ID=MSR-TR-99-87. Abbreviated
version published in Neural Computation, 13(7), 2001.

[4] LIBSVM: LIBSVM -- A Library for Support Vector Machines; Chih-Chung Chang and Chih-Jen Lin; ***http://www.csie.ntu.edu.tw/~cjlin/libsvm/***

[5] A Practical Guide to Support Vector Classi_cation; Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin
Department of Computer Science; National Taiwan University, Taipei 106, Taiwan; http://www.csie.ntu.edu.tw/~cjlin
Last updated: October 2, 2008

[6] Context changes detection by one-class SVMs; Gaëlle Loosli1, Sang-Goog Lee2, and Stéphane Canu1
1 PSI, CNRS FRE2645, INSA de Rouen, FRANCE; 2 Interaction Lab./Context Awareness TG,Samsung Advanced Institute of Technology, Korea

[7] Kernel methods and the exponential family; St_ephane Canu1 and Alex J. Smola2; 1- PSI - FRE CNRS 2645 INSA de Rouen, France; St Etienne du Rouvray, France; Stephane.Canu@insa-rouen.fr; 2- Statistical Machine Learning Program; National ICT Australia and ANU; Alex.Smola@nicta.com.au

[8] A Class of Single-Class Minimax Probability; Machines for Novelty Detection; James T. Kwok, Ivor Wai-Hung Tsang, and Jacek M. Zurada, *Fellow, IEEE*

[9] Using One-Class SVMs and Wavelets for Audio Surveillance Systems;
Asma Rabaoui¤, Manuel Davyy, St´ephane Rossignoly, Zied Lachiri¤ and Noureddine Ellouze; ¤ Unit de recherche Signal, Image et Reconnaissance des formes, ENIT, BP 37, Campus Universitaire,1002 le Belvdre, Tunis Tunisia. y LAGIS, UMR CNRS 8146, and INRIA SequeL Team, BP 48, Cit Scientifique, 59651 Villeneuve d'Ascq Cedex, Lille France. e-mails: Manuel.Davy@inria.fr, Asma.Rabaoui@enit.rnu.tn

[10] SVMC: Single-Class Classification With Support Vector Machines; H wan jo Yu, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana,IL 61801 USA, hwanjoyu@uiuc.edu