Facultad de Matemática, Astronomía, Física
y Computación
Universidad Nacional de Córdoba

Trabajo Especial

Aplicando máquinas de soporte vectorial al análisis de pérdidas no técnicas de energía eléctrica

Estefanía Nievas Lio

Directores: Dr. Germán Ariel Torres,

Dr. Damián Fernandez Ferreyra.

2 de mayo de 2016

Agradecimientos

A mi familia por estimularme, acompañarme y alentarme en cada paso de mi vida.
A mis amigas por estar siempre presente incondicionalmente.
A Ale por acompañarme en cada renglón de este trabajo, como en cada momento de los últimos años.

A mis compañeros de la facu por compartir estos hermosos años y alentarme siempre.
A los profesores que nunca dudaron en abrirme las puertas de sus oficinas.
A la Universidad que me permitió estudiar gratuitamente.

Resumen

Las pérdidas no técnicas en la distribución de energía eléctrica generan grandes gastos a las empresas encargadas de prestar el servicio de energía eléctrica y son extremadamente difíciles de detectar. En este proyecto se usará una técnica de aprendizaje automático (más conocida como Machine Learning) basada en máquinas de soporte vectorial (SVM, siglas en inglés de Support Vector Machine) para poder clasificar, de la manera más confiable posible, a los usuarios de la red en dos grupos diferenciados: los que cometen fraude y los que no. El entrenamiento se realizará a partir de una base de datos ya clasificada y tomando en cuenta el consumo de los usuarios a lo largo de un período de tiempo. Tales datos, en este proyecto, serán de usuarios de la ciudad de Córdoba. En nuestro trabajo implementaremos un algoritmo que construya el clasificador y luego analizaremos su confiabilidad clasificando a consumidores de la ciudad que han sido sometidos a una auditoría. Luego de obtener un clasificador confiable el mismo servirá para detectar posibles fraudes de los usuarios.

Palabras claves: Máquina de soporte vectorial (SVM), aprendizaje automático, pérdida de energía eléctrica.

Códigos de clasificación:

- 65-04 Explicit machine computation and programs (not the theory of computation or programming)
- 62H30 Classification and discrimination; cluster analysis
- 68T10 Pattern recognition, speech recognition
- 90C20 Quadratic programming
- 90C90 Applications of mathematical programming
- 90B50 Management decision making, including multiple objectives

Índice

Índice de figuras

Índice de tablas

1. Introducción

1.1. Aprendizaje automático

Una de las tareas más desafiantes en la ciencia de la computación es construir máquinas o programas que sean capaces de aprender. Existen varias formas de definir aprendizaje [2]:

"Cambios adaptivos en el sistema para hacer la misma tarea de la misma población de una manera más eficiente y efectiva la próxima vez" [22].

"Un programa de computadora se dice que aprende de una experiencia E con respecto a una clase de tareas T y medida de desempeño D, si su desempeño en las tareas en T, medidas con D, mejoran con experiencia E" [17].

Existen diferentes tipos de aprendizaje entre ellos. Uno de ellos es el aprendizaje inductivo, que crea modelos de conceptos a partir de generalizar ejemplos simples. Como un ejemplo de éste tenemos el aprendizaje automático, o Machine Learning, que es una rama de la inteligencia artificial que se encarga de desarrollar programas que sean capaces de aprender, es decir, de crear programas capaces de generalizar comportamientos a partir de una información no estructurada dada como ejemplos. Por lo tanto, el aprendizaje automático es un proceso de inducción del conocimiento.

Podemos clasificar los diferentes tipos de algoritmos de aprendizaje automático en dos grandes grupos:

- aprendizaje inductivo supervisado, y
- aprendizaje inductivo no supervisado.

La única diferencia entre ambos tipos es que en el primero, las muestras se encuentran etiquetadas. Estas etiquetas podrían ser un valor numérico, y el objetivo del algoritmo será encontrar un conjunto de características a partir de las mustras, las cuales se denominan atributos, que permitan predecir con precisión el valor de etiqueta correcto de cada objeto del conjunto de entrenamiento, produciendo así una función que tenga como salida un valor real, como es en los problemas de regresión, o una etiqueta de clase, como es en los problemas de clasificación.

Para resolver los problemas de clasificación, una opción es crear clasificadores lineales, los cuales utilizan hiperplanos como herramienta de desición. Un modelo que propone esto es el de las Máquinas de Soporte Vectorial, o SVMs en sus siglas en inglés. Sin embargo, este modelo suele ser utilizado también para problemas de regresión, u otros problemas de aprendizaje tanto supervisado como no supervisado.

Como ejemplos de otros modelos para resolver problemas de aprendizaje supervisado podemos encontrar: redes neuronales artificiales, algoritmos genéticos, árboles de decisión, entre otros.

1.2. Máquinas de soporte vectorial

Las Máquinas de Soporte Vectorial (SVMs en su abreviatura en inglés) son un modelo para resolver problemas de aprendizaje automático. Seleccionan un pequeño número de instancias críticas, denominadas vectores de soporte, de cada clase, y construyen una función discriminante lineal que las separa a la mayor distancia posible.

En particular, el modelo de las SVMs fueron propuestas en 1995 por Cortes y Vapnick [5] para resolver problemas de clasificación de vectores en un cierto espacio en dos grupos, con la idea de mapearlos llevándolos a un espacio de dimensión mayor donde se pueda construir una separación lineal.

En [5] se realiza una pequeña restrospección histórica sobre el avance del aprendizaje automático que describimos a continuación. El estudio de reconocimiento de patrones comenzó hace aproximadamente 80 años atrás con Fisher, quien propuso el primer algoritmo en 1936. Él consideró un modelo de dos poblaciones distribuidas normalmente en dimensión n y muestra que la solución óptima es una decisión dada por una función cuadrática, pero que si las matrices de covarianza de las poblaciones son iguales, la función resulta ser lineal, y si son distintas, recomienda usar una combinación lineal fija de ambas matrices como matriz de covarianza nueva, y para los casos no normales también recomienda usar una decisión lineal. Por lo tanto, los algoritmos de reconocimiento de patrones están asociados muy en sus orígenes con la construcción de una superficie de decisión lineal, es decir, un hiperplano que al dividir el espacio en dos partes brinda una herramienta de decisión para clasificar los puntos dependiendo en que sector se encuentra.

En 1962, Rosenblatt experimenta un nuevo tipo de aprendizaje automático: perceptrones o neural networks (redes neuronales), que consiste en conectar neuronas, donde cada una implementa un hiperplano separador, generando entonces una superficie de separación lineal a trozos. En aquellos tiempos no había sido encontrado un algoritmo que minimizara el error sobre un conjunto de vectores por ajuste de pesos de la red. Rosenblatt sugirió un esquema donde se tuvo en cuenta esto, y acorde con este arreglo, los vectores iniciales son no–linealmente transformados en un espacio de dimensión mayor, en el cual se contruye una decisión lineal. Por ende, la construcción de una regla de decisión fue asociada con la construcción de hiperplanos lineales en algún espacio.

En el año 1995 Cortes y Vapnik proponen un nuevo tipo de machine learning, con la idea de llevar los vectores a un espacio de dimensión mayor mediante un mapeo no lineal elegido previamente, en el cual una superficie de decisión lineal es construida con propiedades especiales que aseguran una gran generalidad a la red.

En ese momento, las SVMs fueron desarrolladas para resolver problemas de clasificación, pero luego, en 1997 [23] extendieron el dominio a problemas de regresión (SVR en sus siglas en inglés).

Desde entonces las SVMs han sido un tema muy estudiado y de mucho interés para la comunidad dentro de aprendizaje automático. Han sido en particular muy aceptadas en campos como el de identificación de imágenes o reconocimiento de rostros. Por ejemplo, en el 2001, se mostró que el rendimiento de esta técnica supera al de otros clasificadores no lineales como es el caso de redes neuronales, k-nearest neighbors (k-vecinos cercanos) o Nearest Center (centro vecino) como en [8]. De manera similar, en el campo de reconocimiento del habla, en problemas

de clasificación fonética, se logra ver en [7] que esta técnica supera a los clasificadores Gaussian mixture (Gaussianos mixtos). Estos son sólo algunos ejemplos del gran abanico de aplicaciones y mejoras que se han desarrollado en las últimas dos décadas.

1.3. Clasificación

El problema de clasificación puede ser restringido a considerar un problema de dos clases sin pérdida de generalidad. En este problema el objetivo es separar las dos clases por una función que es inducida por los ejemplos disponibles. El objetivo es producir un clasificador que trabaje bien en los ejemplos no vistos a tiempo de entrenamiento [7].

Intuitivamente, dado un conjunto de puntos de dos clases, un SVM encuentra el hiperplano que separa la mayor cantidad de puntos de la misma clase en el mismo lado, mientras maximiza la distancia de las clases con el hiperplano. Acorde con Vapnik en [25], este hiperplano es llamado hiperplano de separación óptima, el cual minimiza el riesgo de clasificar mal tanto los ejemplos del conjunto de datos como los futuros ejemplos [8].

Mientras que muchas de las técnicas usadas para entrenar el clasificador mencionado están basadas en la idea de minimizar el error de entrenamiento, más usualmente llamado riesgo empírico (empirical risk), las SVMs operan en otro principio de inducción llamado estructura de riesgo mínimo (structural risk minimization), el cual minimiza un margen superior en la generalización del error [7].

Desde el punto de vista de la implementación, entrenar a una SVM es equivalente a resolver un problema de programación cuadrática con restricciones lineales con un número de variables igual al número de datos de entrenamiento. Este problema es un desafío cuando el tamaño del conjunto es mayor que algunos miles.

En este trabajo comenzaremos explicando desde un punto de vista teórico, gráfico y computacional, la aplicación de esta técnica a casos base de estudio. Luego resumiremos la aplicación de ella en problemas reales de variada índole, y para finalizar trataremos de aplicar lo estudiado a un conjunto de datos de entrenamiento brindado por la red distribuidora de energía electrica de la ciudad de Córdoba con el objetivo de distinguir los puntos de la ciudad que generan pérdidas no técnicas de electricidad.

2. Desarrollo Teórico

En esta sección desarrollaremos la teoría de las SVMs, analizando primero el caso base de dos clases linealmente separables, luego el caso de dos clases no linealmente separables, después el caso no lineal en general, y finalizaremos con la generalización a multiclases y la extensión de esta técnica a problemas de regresión. Nos basaremos en los aportes realizados en los trabajos [4] y [5].

2.1. Caso linealmente separable

La idea central detrás de las SVMs desarrollada por Vapnik en 1998 es definir un margen entre dos clases por la separación máxima entre ellas. Para introducir los conceptos, asumiremos que tenemos dos clases linealmente separables en el espacio $\mathbb{R}^p$.

Definiremos una función lineal a valores reales desde el plano donde se encuentran los vectores de tal forma que la intersección de la gráfica de ésta con el dominio separe a ambas clases de "la mejor manera posible". Es decir, dentro del conjunto de funciones lineales que cumplan que a dos vectores de la misma clase se le asignen valores del mismo signo en la imagen, elegiremos "la óptima".

Por ejemplo, si tenemos $\mathbb{R}^p = \mathbb{R}^2$ y las clases contienen las etiquetas 1 y -1, un ejemplo de datos de entrenamiento linealmente separable puede verse en la Figura 1.

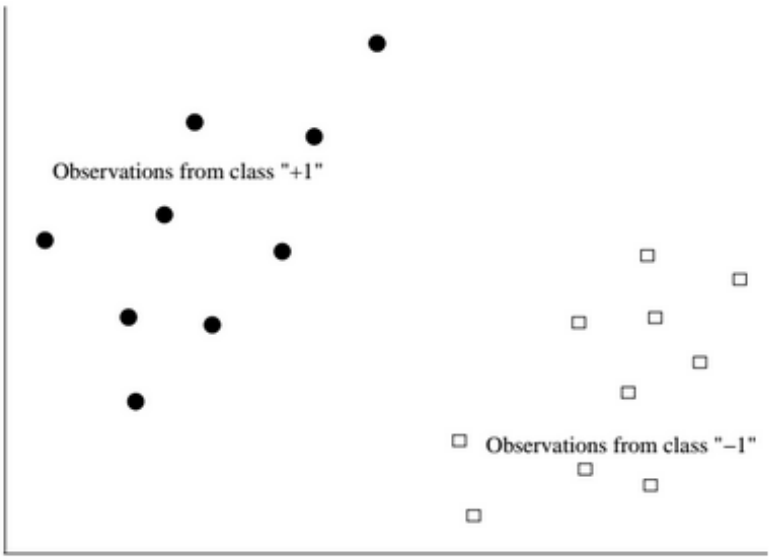


Figura 1: Conjunto de entrenamiento.

Luego, el gráfico de una función lineal de $\mathbb{R}^2$ en $\mathbb{R}$ es un hiperplano en $\mathbb{R}^3$ que interseca, salvo paralelismo, al dominio $\mathbb{R}^2$ dividiéndolo en dos semiplanos. Esta intersección coincide con la curva de nivel cero de la función lineal. Si los datos de entrenamiento se ubican en el dominio

Fijado un hiperplano, se denomina **margen** a la menor distancia perpendicular entre el conjunto de nivel cero del mismo que pertenezca al dominio y las observaciones, por ende, el hiperplano óptimo será aquél que maximice el margen (ver Figura 3).

$\mathbb{R}^2$, se busca que cada clase quede en un semiplano distinto, es decir, en lados distintos de la curva. Pero esta separación puede realizarse de diferentes maneras como se observa en la Figura 2.

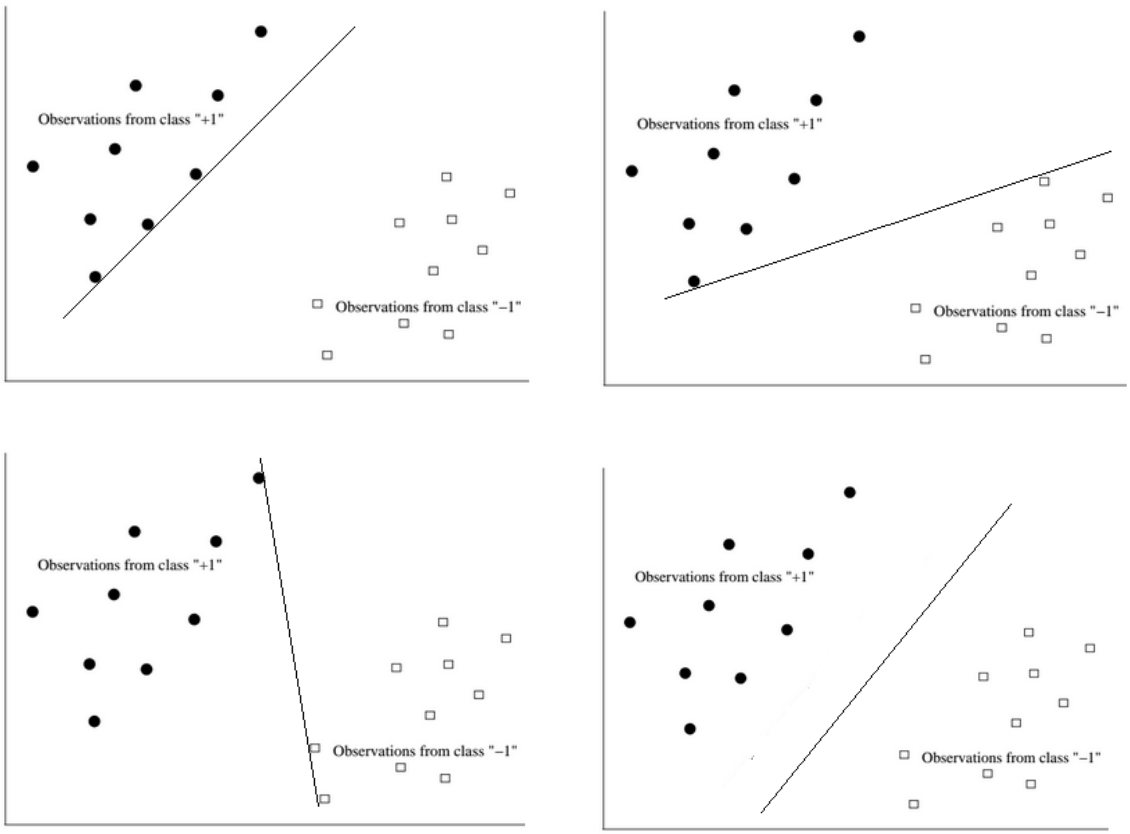


Figura 2: Posibles hiperplanos separadores.

Ahora nos preguntamos, ¿Cuál de las infinitas posibilidades de intersecciones es la mejor? ¿Cuál será el hiperplano óptimo? El objetivo es que la distancia entre la intersección del hiperplano separador con el plano X–Y, y cada una de las clases, sea la mayor posible.

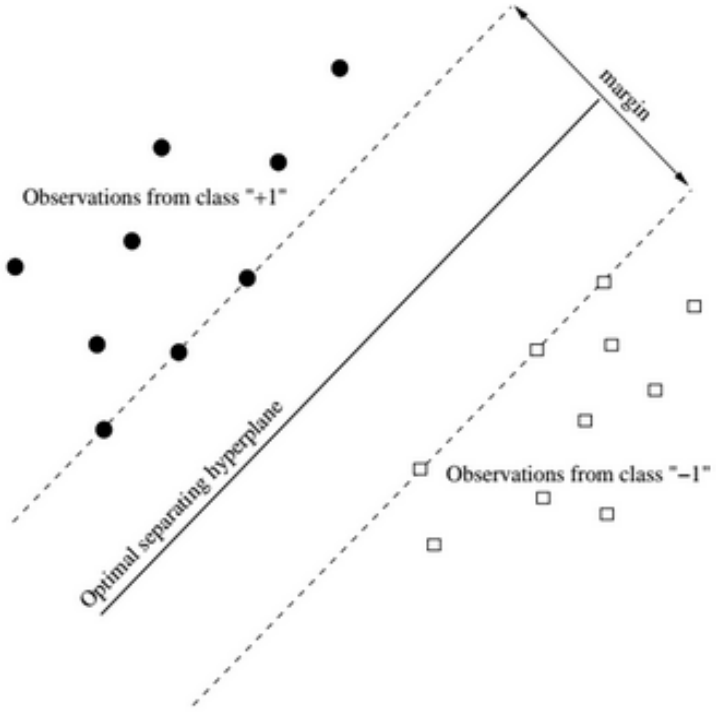


Figura 3: Hiperplano separador óptimo.

Definamos entonces esta intersección, para desarrollar una definición de margen y con esto plantear nuestro problema. Dado $\omega = (\omega_1, \omega_2, \ldots, \omega_p) \in \mathbb{R}^p$ y $b \in \mathbb{R}^p$ un vector constante, escribimos la función lineal $h : \mathbb{R}^p \to \mathbb{R}$ como:

$$h(x) = \omega' x + b, \quad x \in \mathbb{R}^p.$$

Donde ω' denota a ω transpuesta, y consideramos a los vectores en $\mathbb{R}^p$ como vectores columna. Ahora, para cada $c \in \mathbb{R}$, el conjunto

$$H_c(w, b) = \{x : h(x) = c\},$$

es un hiperplano de dimensión $p-1$. Si $c = 0$, entonces $H_c(w, b) = \{x : h(x) = 0\}$, y es denotado por $H(w, b)$.

Por ende, teniendo fija una función lineal h, y denotando al conjunto de datos como

$$D = \{(x_1, y_1), \ldots, (x_n, y_n)\},$$

donde los $x_i \in \mathbb{R}^p$ conforman el conjunto de entrenamiento e $y_i \in \{-1, +1\}$ corresponde a las etiquetas, definimos margen de la siguiente forma:

$$\min_{y_i=-1} d(x_i, H(w, b)) + \min_{y_i=+1} d(x_i, H(w, b)).$$

Para calcular la distancia entre un punto y un hiperplano en $\mathbb{R}^p$ utilizaremos la siguiente fórmula:

$$d\left(z, H_c\left(w,b\right)\right) = \frac{|w'z + b - c|}{\|w\|}.$$

Podemos encontrar la prueba de este resultado en [4, pág. 265]. Luego, como queremos encontrar una h de tal forma que maximice el margen, nuestro problema se reduce a encontrar una función $h : \mathbb{R}^p \to \mathbb{R}$ tal que $h\left(x\right) = w'x + b$ donde los coeficientes w y b están dados por la solución del siguiente problema de optimización:

$$\begin{aligned} \max_{w,b} \quad & \left[\min_{y_i=-1} d\left(x_i, H\left(w,b\right)\right) + \min_{y_i=+1} d\left(x_i, H\left(w,b\right)\right)\right] \\ \text{sujeto a} \quad & y_i\left(w'x_i + b\right) \geq 1, \quad i = 1, \ldots, n. \end{aligned}$$

Reemplazando quedaría:

$$\begin{aligned} \max_{w,b} \quad & \left[\min_{y_i=-1} \frac{|w'x_i + b - c|}{\|w\|} + \min_{y_i=+1} \frac{|w'x_i + b - c|}{\|w\|}\right], \\ \text{sujeto a} \quad & y_i\left(w'x_i + b\right) \geq 1, \quad i = 1, \ldots, n. \end{aligned}$$

Notemos que si b y w son solución, también lo son kb y kw para toda constante k positiva.

Luego, si tomamos como márgenes de las clases 1 y -1 a $H_{c=1}$ y $H_{c=-1}$ respectivamente y ocurre que $\|kw\| \geq \|w\|$, entonces, estos márgenes se acercan y contradeciría el objetivo del problema. Por ende, buscamos el menor $k \in (0, 1]$ que siga cumpliendo la restricción $y_i\left(w'x_i + b\right) \geq 1$, $i = 1, \ldots, n$, es decir, buscamos que al menos un par (x_i, y_i) de cada clase tal que $y_i\left(w'x_i + b\right) = 1$, ya que, si existe al menos uno de cada clase que cumpla esto, los demás cumplirán $y_i\left(w'x_i + b\right) \geq 1$.

Usando esto, la expresión del margen quedaría:

$$M = \min_{y_i=-1} d\left(x_i, H\left(w,b\right)\right) + \min_{yi=+1} d\left(x_i, H\left(w,b\right)\right) = \frac{1}{\|w\|} + \frac{1}{\|w\|} = \frac{2}{\|w\|}.$$

Así, nuestro problema se resumiría en:

$$\begin{aligned} \max_{w,b} \quad & \frac{2}{\|w\|}, \\ \text{sujeto a} \quad & y_i\left(w'x_i + b\right) \geq 1, \quad i = 1, \ldots, n. \end{aligned}$$

Lo que equivale a:

$$\boxed{\begin{aligned} \min_{w,b} \quad & \frac{1}{2}\|w\|^2, \\ \text{sujeto a} \quad & y_i\left(w'x_i + b\right) \geq 1, \quad i = 1, \ldots, n. \end{aligned}}$$

Estamos entonces frente a un problema de optimización. Para entender un poco más de qué se trata y cómo trabajar con este tipo de problemas consultamos en [15] y [14].

A modo general, el siguiente problema es considerado un problema de optimización:

$$\min_{y} \quad f(y),$$

$$\text{sujeto a} \quad y \in \Omega \subset \mathbb{R}^m,$$

donde f es llamada función objetivo y el conjunto Ω, el cual frecuentemente es definido por un conjunto de igualdades y desigualdades, es llamado el conjunto factible. Los puntos de Ω serán los puntos factibles del problema.

En nuestro caso,

$$\begin{aligned} f: \quad \mathbb{R}^p \times \mathbb{R} &\to \mathbb{R}, \\ (w,b) &\mapsto \frac{1}{2}\|w\|^2, \end{aligned}$$

y

$$\Omega = \{(w,b) \in \mathbb{R}^p \times \mathbb{R} : y_i (w' x_i + b) \geq 1, \quad i = 1, \ldots, n\}.$$

Dentro del estudio de optimización hay varios subcampos de estudio, y varios tipos de problemas dependiendo de las propiedades de la función objetivo y del tipo de restricciones con el que se obtiene el conjunto factible. En este sentido, la formulación anterior de SVM es un problema de optimización con restricciones no lineales.

El conjunto de puntos en el cual se cumplen las igualdades en las restricciones de desigualdad es particularmente muy importante. Decimos que una restricción de desigualdad $h_i(x) \geq 0$ será activa (o efectiva) en un punto y^* si $h(y^*) = 0$. Luego, dado y^*, el conjunto de restricciones que son activas para y^* es:

$$\mathcal{A}(y^*) = \{i : h_i(y^*) = 0\}.$$

En nuestro caso, $h_i(w,b) = y_i (w' x_i + b) - 1$. Luego, en optimización con restricciones, las condiciones Karush–Kuhn–Tucker (también conocidas como las Kuhn-Tucker o las condiciones KKT) son necesarias y suficientes, pues nuestra función es convexa, para que un problema de programación matemática sea óptimo. Éstas generalizan la condición de los multiplicadores de Lagrange, y son una forma útil de construir una solución.

Teorema 1 (Condiciones necesarias o KKT para un mínimo local)
Sea $\mathcal{X} \subset \mathbb{R}^p$ y $f : \mathcal{X} \to \mathbb{R}$, con restricciones diferenciables $h_i : \mathcal{X} \to \mathbb{R}$, $i = 1, \ldots, n$. Supongamos que x^* es un mínimo local de f en el conjunto:

$$\mathcal{Y} = \mathcal{X} \cap \{x \in \mathbb{R}^p : h_i(x) \geq 0, \quad i = 1, \ldots, n\},$$

y que $\{\nabla h_i(x^*) : i \in \mathcal{A}(x^*)\}$ de las derivadas de las restricciones activas en x^* son un conjunto de vectores linealmente independientes. Luego, existen $\alpha_1^*, \ldots, \alpha_n^* \in \mathbb{R}$ tal que

$$\begin{aligned} \nabla f(x^*) - \sum_{i=1}^{n} \alpha_i^* \nabla h_i(x^*) &= 0, \\ \alpha_i^* &\geq 0, \quad i = 1, \ldots, n, \\ \alpha_i^* h_i(x^*) &= 0, \quad i = 1, \ldots, n. \end{aligned}$$

Se puede consultar la prueba en [14, p. 342]. En nuestro caso, no es necesario corroborar que los gradientes sean linealmente independientes, ya que las restricciones son funciones afines. Luego, de las condiciones KKT para mínimo local se puede mostrar que

$$\begin{aligned} \alpha_i^* &\geq 0 \text{ donde } h_i(x^*) = 0, \\ \alpha_i^* &= 0 \text{ donde } h_i(x^*) > 0, \end{aligned}$$

es decir, α_i^* y $h_i(x^*)$ no pueden ser ambos no nulos. Los α_i^* son llamados los multiplicadores de Lagrange. Claramente, $\alpha_i \geq 0$ sólo en el conjunto de restricciones activas, como será visto luego con SVMs.

Además, el problema primal (cuya función objetivo es f) frecuentemente es transformado a un problema sin restricciones utilizando los multiplicadores de Lagrange, y la reformulación correspondiente es

$$L(x, \alpha) = f(x) - \sum_{i=1}^{n} \alpha_i h_i(x).$$

Luego, se puede reformular el problema primal a un espacio dual en el cual el problema es más fácil de manipular y con interpretaciones más intuitivas. Básicamente, la intuición es la siguiente: si la solución del problema primal es expresada en términos de $\alpha' = (\alpha_1, \ldots, \alpha_n)$, el nuevo problema dual tendrá una nueva función objetivo (que denotaremos E_D) donde los roles están cambiados, es decir, α se convierte en la variable. Más especificamente, el Lagrangiano del problema dual es

$$E_D(\alpha) = \inf_{x \in \mathcal{X}} E_P(x, \alpha),$$

y por las condiciones KKT tenemos que $\alpha_i^* \leq 0$ en el mínimo local x^*, el problema dual puede ser formulado como

$$\begin{aligned} \max_{\alpha} \quad & E_D(\alpha) \\ \text{sujeto a} \quad & \alpha \geq 0. \end{aligned}$$

Uno de los beneficios inmediatos de esta reformulación es que las restricciones se simplifican. Finalmente, por un resultado llamado Teorema de dualidad, la solución del problema dual coincide con la del problema primal.[1, pág. 299]

Volviendo ahora a nuestro problema, utilizamos los multiplicadores de Lagrange y la función Lagrangiana:

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i \left[y_i (w' x_i + b) - 1 \right],$$

donde $\alpha_i \in \mathbb{R}$ son los multiplicadores. Luego, derivando e igualando a cero, obtenemos que un mínimo local tiene que satisfacer que,

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i, \quad \sum_{i=1}^{n} \alpha_i y_i = 0.$$

Aplicando las condiciones KKT a este caso, existe un α^* tal que $\alpha_i^* = 0$ para todo x_i que sastiface $y_i (w^{*\prime} x_i + b^*) > 1$ y $\alpha_i^* \geq 0$ cuando $y_i (w^{*\prime} x_i + b^*) = 1$. Como los vectores x_i para los cuales $\alpha_i > 0$ son el soporte de la solución, estos son llamados vectores soporte.

Luego, reemplazando estas condiciones se deduce el problema dual y este quedaría:

$$\begin{aligned} \max_{\alpha} \quad & E_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i' x_j \\ \text{sujeto a} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0, \quad \alpha_i \geq 0, \quad i = 1, \ldots, n. \end{aligned}$$

Si $c = (-1, \ldots, -1)'$ (dimensión n) y $Q_{ij} = y_i y_j x_i' x_j$, el problema queda:

$$\begin{aligned} \min_{\alpha} \quad & E_D(\alpha) = \frac{1}{2} \alpha' Q \alpha + c' \alpha \\ \text{sujeto a} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0, \quad \alpha_i \geq 0, \quad i = 1, \ldots, n. \end{aligned}$$

El problema es un problema de optimización cuadrática, y además, la matriz Q es semi-definida positiva, por lo que los algoritmos tradicionales de programación cuadrática serán suficientes para resolverlo. Luego, teniendo α, tenemos w, y faltaría sólo b. Pero, como dijimos, los vectores soporte cumplen $y_i (w' x_i + b) = 1$, lo cual implica que $\hat{b} = y_i - \hat{w}' x_i$ (ya que y_i es 1 o -1). Se debe seleccionar para $\{y_i = +1\}$ el b más grande, y para $\{y_i = -1\}$ el más chico, para que se cumpla $y_i \left(\hat{w}' x_i + \hat{b} \right) \geq 1$ para todo i. Tenemos entonces,

$$\begin{aligned} \hat{b} &= \min_{y_i = -1} (-1 - \hat{w}' x_i) = -1 - \max_{y_i = -1} \hat{w}' x_i, \\ \hat{b} &= \max_{y_i = +1} (-1 - \hat{w}' x_i) = 1 - \min_{y_i = +1} \hat{w}' x_i. \end{aligned}$$

Juntando tenemos que:

$$\hat{b} = -\frac{1}{2} \left(\min_{y_i = +1} \hat{w}' x_i + \max_{y_i = -1} \hat{w}' x_i \right),$$

y el problema está terminado, siendo el clasificador de este caso base el siguiente:

$$f(x) = \text{sign} \left(\sum_{i=1}^{n} \hat{\alpha}_i y_i x_i' x + \hat{b} \right).$$

El problema de maximizar E_D bajo las condiciones dadas puede ser resuelto de distintas maneras. Algunas de ellas detallaremos en la subsección 2.5.3.

2.2. Caso no linealmente separable

El problema de optimización descripto en la sección previa puede no tener solución si el conjunto es no separable. En este caso, se quiere separar el conjunto de entrenamiento con el mínimo número de errores, y con este propósito, modificamos las restricciones del problema

por pérdidas tal que la penalización ocurre por un error de clasificación. Para expresar más formalmente esto, introducimos algunas variables no negativas, que denominamos de holgura, y denotamos con $\xi_i \geq 0$, $i = 1, \ldots, l$. Cambiamos entonces la condición $y_i (w'x_i + b) - 1 \geq 0$ por $y_i (w'x_i + b) - 1 + \xi_i \geq 0$. Entonces ahora la nueva regla de clasificación es:

$$\begin{aligned} w'x_i + b &\geq +1 - \xi_i, \quad \text{si } y_i = 1, \\ w'x_i + b &\leq -1 + \xi_i, \quad \text{si } y_i = -1. \end{aligned}$$

Si un x_i no es bien clasificado, tendremos, $\xi_i > 1$. Entonces el número de errores es menor que $\sum_{i=1}^{n} \xi_i$. Por ende, reemplazamos la función a minimizar por $\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i$, donde $C > 0$ es un parámetro de penalización.

Luego, nuestro problema a resolver es el siguiente:

Encontrar $h : \mathbb{R}^p \to \mathbb{R}$ de la forma $h(x) = w'x + b$ tal que w y b sean solución de

$$\begin{aligned} \min_{w,b,\xi} \quad & E_P(w, \xi) = \frac{2}{\|w\|} + C\sum_{i=1}^{n} \xi_i \\ \text{sujeto a} \quad & y_i (w'x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1, \ldots, n. \end{aligned}$$

El problema dual quedaría:

$$\begin{aligned} \max \quad & E_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i' x_j \\ \text{sujeto a} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, n. \end{aligned}$$

Vapnik (1998) mostró que las condiciones KKT en el caso no–linealmente separable se reducen a las siguientes tres condiciones:

$$\begin{aligned} \alpha_i = 0 &\Rightarrow y_i (w'x_i + b) \geq 1, \quad \xi_i = 0, \\ 0 < \alpha_i < C &\Rightarrow y_i (w'x_i + b) = 1, \quad \xi_i = 0, \\ \alpha_i = C &\Rightarrow y_i (w'x_i + b) \leq 1, \quad \xi_i \geq 0. \end{aligned}$$

De esto agregamos al conjunto de vectores soporte a aquéllos que $\alpha_i = C$ y $0 \leq \xi_i \leq 1$. Los puntos que sastifacen esta condición son correctamente clasificados pero se encuentran entre los hiperplanos H_{+1} y H_{-1} (puntos B), y los puntos mal clasificados son aquéllos que $\xi_i > 1$ (puntos F). Ver Figura 4.

Luego, usando los mismos detalles que la subsección anterior, se obtiene el clasificador luego de resolver el problema de optimización.

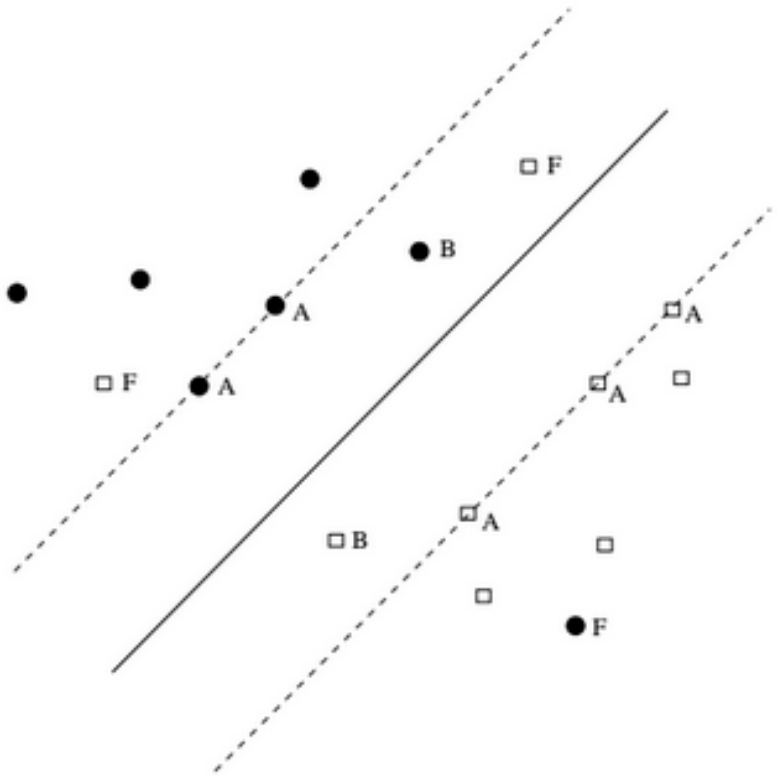


Figura 4: Datos no linealmente separables.

2.3. Caso no lineal en general

La clasificación descripta hasta el momento es limitada a la separación lineal. Las SVMs solucionan este problema mapeando los puntos de muestra a un espacio de dimensión mayor usando un mapeo no–lineal elegido previamente. Es decir, elegimos $\Phi : \mathbb{R}^p \to \mathcal{H}$, donde la dimensión de $\mathcal{H}$ es mayor que p, y de tal manera que transforme los datos de entrenamiento en un conjunto linealmente separable. Queremos entonces realizar una separación por hiperplano en el espacio de dimensión mayor, lo cual es equivalente a una separación por una superficie no–lineal en $\mathbb{R}^p$. La idea intuitiva del problema en SVMs para el problema no lineal es reemplazar el producto interno ${x_j}'x$ en

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i x_i' x + b\right),$$

la expresión del clasificador lineal SVM, con $\Phi\left(x_j\right)' \Phi\left(x\right)$, quedando como resultado

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i \Phi\left(x_j\right)' \Phi\left(x\right) + b\right).$$

El producto interno euclidiano es calculado en el primer caso en $\mathbb{R}^n$, mientras que en el segundo caso usa una transformación Φ para convertir los puntos iniciales x en puntos del espacio de dimensión mayor $\mathcal{H}$, y recién alli, realizar el producto entre los vectores.

Por ejemplo, sea $x' = (x_1, x_2)$ y consideramos el vector $z' = (z_1, z_2, z_3)$ en $\mathbb{R}^3$. Definamos $\Phi : \mathbb{R}^2 \to \mathbb{R}^3$ por:

$$\Phi(x) = \Phi(x_1, x_2) = \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right) = z'.$$

Con esta Φ, un problema difícil no–lineal en 2D es convertido a un problema estandar de clasificación lineal en 3D, como se muestra en la Figura 5).

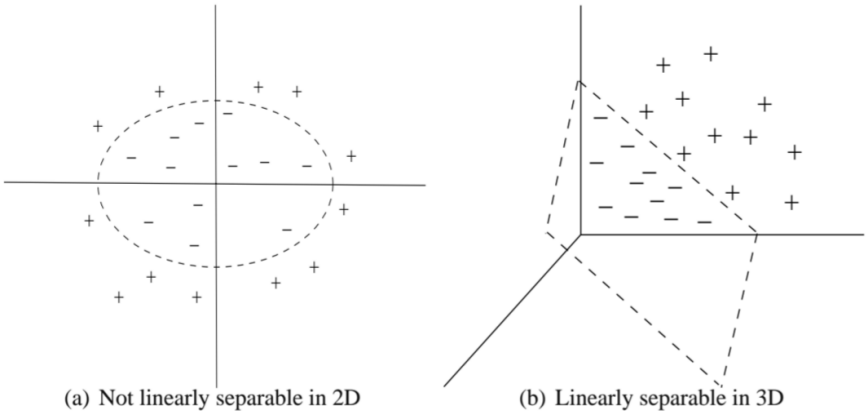


Figura 5: Mapeo no lineal para un conjunto de entrenamiento que no puede ser separado linealmente.

El problema en implementar esta estrategia es saber si hay alguna transformación Φ que haga los datos separable en algún espacio de dimensión mayor.

Notemos que los datos iniciales en el caso linealmente separable sólo aparecen en la forma de producto punto, entonces, en el espacio de dimensión mayor nosotros sólo tratamos con los datos en la forma $\Phi(x_i)' \Phi(x_j)$. Si la dimensión de $\mathcal{H}$ es muy grande este cálculo podría ser computacionalmente costoso. Entonces si nosotros tenemos una *función de núcleo* tal que $K(x_i, x_j) = \Phi(x_i)' \Phi(x_j)$, podemos usar esto en lugar de $x_i'x_j$ en todas las veces que aparece en el problema de optimización, y nunca necesitamos saber explícitamente quién es Φ. Este producto en el ejemplo anterior sería

$$\Phi(x)' \Phi(y) = \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right) \left(y_1^2, \sqrt{2}y_1y_2, y_2^2\right)' = (x_1y_1 + x_2y_2)^2 = (x'y)^2 = K(x, y),$$

lo que muestra un ejemplo de cómo un producto interno basado en Φ puede convertirse en una función de dos entradas que requiere menos cuentas, y por ende, menos costo computacional. Asumimos entonces en el proceso de resolución del problema, que una función núcleo K puede ser encontrada.

Tenemos entonces $K(x,y) = \Phi(x)' \Phi(y)$, y el clasificador planteado antes queda escrito de la siguiente manera:

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i K(x_j, x) + b\right),$$

ecuación que soluciona el siguiente problema de optimización:

$$\begin{aligned} \text{máx} \quad & E_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \\ \text{sujeto a} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, n. \end{aligned}$$

Debido a esta estrategia del núcleo, el problema de optimización en el caso no–lineal tiene forma similar al caso lineal. Esto permite usar la maquinaria de programación cuadrática para el caso no–lineal. Para ver esto, la reformulación del problema en forma de programación cuadrática es:

$$\begin{aligned} \text{mín} \quad & E_D(\alpha) = \frac{1}{2}\alpha' K \alpha + c'\alpha \\ \text{sujeto a} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, n, \end{aligned}$$

donde $K = (K_{ij})$ con $K_{ij} = y_i y_j K(x_i, x_j)$ y $c = (-1, -1, \ldots, -1)$, vector de dimensión n. La única desventaja de esta formulación es que la matriz K no es necesariamente semidefinida positiva, lo cual significa que el problema puede no tener solución en el caso de no trabajar con holguras, por otro lado, si bien en el caso que el conjunto factible es acotado, va a tener solución, en general no es fácil encontrarla. Sin embargo, cuando K sí es semidefinida positiva, esta es la mejor forma.

El problema entonces se centra en cómo elegir una función núcleo K tal que la matriz K es definida positiva, y además, representa a una función Φ que transforma a los datos a un conjunto linealmente separable. Resulta que si K corresponde a un producto interno en algún espacio futuro $\mathcal{H}$, entonces, K es definida positiva. Basta entonces con determinar qué condiciones tiene que cumplir K para corresponderse a algún producto interno $K(x,y) = \Phi(x)' \Phi(y)$ para algún $\Phi : \mathbb{R}^n \to \mathcal{H}$. La respuesta a esto viene dada por las condiciones de Mercer.

Definición 1
Condiciones de Mercer Sea χ un dominio, y consideramos una función K simétrica, continua y a valores reales sobre $\chi \times \chi$. Luego, K se dice que realiza las condiciones de Mercer si, para toda función g a valores reales sobre χ,

$$\int g(x)^2 \cdot dx < \infty \Rightarrow \int K(x,y)\, g(x)\, g(y)\, dxdy \geq 0.$$

Teorema 2
Sea χ un dominio, y consideremos una función continua, simétrica y a valores reales K definida en $\chi \times \chi$. Ahora, sea $\mathcal{H}$ un espacio. Luego, existe una transformación $\Phi : \chi \to \mathcal{H}$ tal que

$$K(x,y) = \Phi(x)' \Phi(y),$$

si y sólo si K sastiface las condiciones de Mercer.

Para una función K dada, verificar la condición descripta podría no resultar una tarea fácil. Pero, en la práctica, existen muchas funciones que son válidas como núcleos, y afortunadamente, muchas de ellas tienen una buena perfomance en los datos del mundo real.

Describimos a continuación, algunas de estas funciones estudiadas que se pueden tomar como posibles núcleos (ver [7]).

(a) Lineal: es el que proviene del producto interno euclídeo, utilizado para modelar el caso lineal.

$$K(x, y) = \langle x, y \rangle.$$

(b) Polinomial: un mapeo polinomial es un método popular para el modelado no lineal.

$$K(x, y) = \langle x, y \rangle^d,$$

en su forma homogénea. Para evitar los problemas que pueden provenir del cero, suele ser usado en su versión no homogénea:

$$K(x, y) = (\langle x, y \rangle + c)^d.$$

En particular, cuando $d = 2$, se denomina cuadrática.

(c) Función de base Radial (RBF, por sus siglas en inglés): éstas han recibido especial atención, más comunmente con una Gaussiana de la forma:

$$K(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right).$$

El coeficiente $1/(2\sigma^2)$ suele ser reemplazado por γ con $\gamma > 0$. Técnicas clásicas utilizan funciones de base radial en algunos métodos para determinar centros de subconjuntos. Una característica de las SVMs es que la selección es implícita, cada vector soporte contribuye una función local gaussiana, centrada en el punto de dato.

(d) Función de base Exponencial Radial: una función de base radial es de la forma:

$$K(x, y) = \exp\left(-\frac{\|x - y\|}{2\sigma^2}\right).$$

Produce una función lineal a trozos la cual puede ser atractiva cuando las discontinuidades son aceptadas.

(e) Perceptron multicapa (MLP en sus siglas en inglés): con una sola capa oculta también tiene una representación de núcleo válida,

$$K(x, y) = \tanh(\gamma \langle x, y \rangle + r),$$

para ciertos valores de parámetros de escala ρ, y compensación ϕ. En nuestro caso los vectores soporte corresponden a la primer capa y los multiplicadores de Lagrange a los pesos.

(f) Serie de Fourier: una serie de Fourier puede ser considerada una expansión en un espacio de dimensión $2n+1$. El núcleo es definido en el intervalo $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$,

$$K(x,y) = \frac{\sin\left(N + \frac{1}{2}(x-y)\right)}{\sin\left(\frac{1}{2}(x-y)\right)}.$$

Sin embargo, este núcleo probablemente no es una buena elección ya que su capacidad de regularización es pobre.

(g) Splines: éstos son muy populares debido a su flexibilidad. Un spline finito de orden κ, con N nodos, localizado sobre τ_s esta dado por,

$$K(x,y) = \sum_{r=0}^{\kappa} x^r y^r + \sum_{s=1}^{N} (x-\tau_s)_+^{\kappa} (y-\tau_s)_+^{\kappa},$$

donde $(\cdot)_+$ hace referencia a la proyección en el ortante no negativo. También podríamos utilizar un spline infinito, y éste es definido en el intervale $[0,1)$ por

$$K(x,y) = \sum_{r=0}^{\kappa} x^r y^r + \int_0^1 (x-\tau_s)_+^{\kappa} (y-\tau_s)_+^{\kappa} \, d\tau.$$

2.4. SVM Multiclases

SVM Multiclases pretende asignar etiquetas a instancias, donde las etiquetas han sido elegidas a partir de un conjunto finito de elementos. Un enfoque para hacerlo es reducir el problema de multiclases en múltiples problemas de clasificacón binaria. Dentro de los métodos más comunes para dicha reducción se encuentra la contrucción de clasificadores binarios que distinguen:

- Entre todos los pares de clases: método conocido como OVO (one–versus–one) o AVA (all–versus–all). Si la cantidad de clases no es muy grande, y denominamos esta cantidad con K, el entrenamiento de $K(K-1)/2$ clasificaciones puede ser implementado. En esta estrategia se obtienen tantos subproblemas como pares de clases podemos obtener, cada uno de ellos es afrontado por un clasificador base independiente. Luego, la clasificación final se realiza combinando la salida de todos los clasificadores base mediante una estrategia de victorias máximas. Esta estrategia se basa en asignar un voto a la clase elegida por cada clasificador binario, luego para cada elemento se suman los votos asignados de cada clase y por último se elige la clase con mayor cantidad de votos.

- Entre una de las etiquetas y las restantes: método conocido como OVA (one–versus–all), en este caso se obtienen tantos subproblemas como clases. Para cada clase, se crea un

clasificador binario que distingue entre esa clase y el resto, la diferencia con los problemas binarios es que no se pide una salida de 0 o 1, sino, un número real, es decir, no se toma el signo a la funcion del clasificador. Luego, para predecir una clase se valúa cada uno de los clasificadores obtenidos y se toma el que retorne un valor mayor.

2.5. Entrenando al clasificador

2.5.1. ¿Qué núcleo usar?

Como ya vimos, si dada una función con ciertas características, ésta sastiface las condiciones de Mercer, entonces puede ser utilizada como núcleo. Mostramos también varios ejemplos de funciones que cumplen esto, y por ende, tenemos varias opciones a mano al momento de elegir el núcleo para entrenar nuestro clasificador. Sin embargo, en [9] se plantea que si bien los núcleos más usados son cuatro (lineal, RBF, MLP y polinomial), se recomienda elegir uno para probar primero, y luego se eligen el parámetro de penalización C y los parámetros del núcleo.

En general, el núcleo RBF es una razonable primera elección. Éste núcleo no lineal mapea las muestras a un espacio de dimensión más grande, y a diferencia del núcleo lineal puede ayudar cuando la relación entre las etiquetas de las clases y las características es no lineal, es decir, en el caso que los datos no son separables. Además, el núcleo lineal es un caso especial de RBF [11] ya que, se analizó experimentalmente, que el núcleo lineal, con un parámetro de penalización C' tiene la misma actuación que el núcleo RBF con algún parámetro (C, γ). De la misma forma, el núcleo MLP se comporta como RBF para ciertos parámetros [13].

La segunda razón es el número de hiper–parámetros, los cuales influencian el comportamiento del modelo seleccionado. El núcleo polinomial tiene más parámetros que el núcleo RBF.

Finalmente, el núcleo RBF tiene menos dificultades numéricas ya que toma valores en el rango $(0, 1]$, en contraste del núcleo polinomial el cual puede tomar valores que tienden a 0 o a ínf cuando el grado es alto. Además, se debe notar que el núcleo MLP no es válido, es decir, no representa el producto interno de dos vectores, bajo algunos parámetros [5].

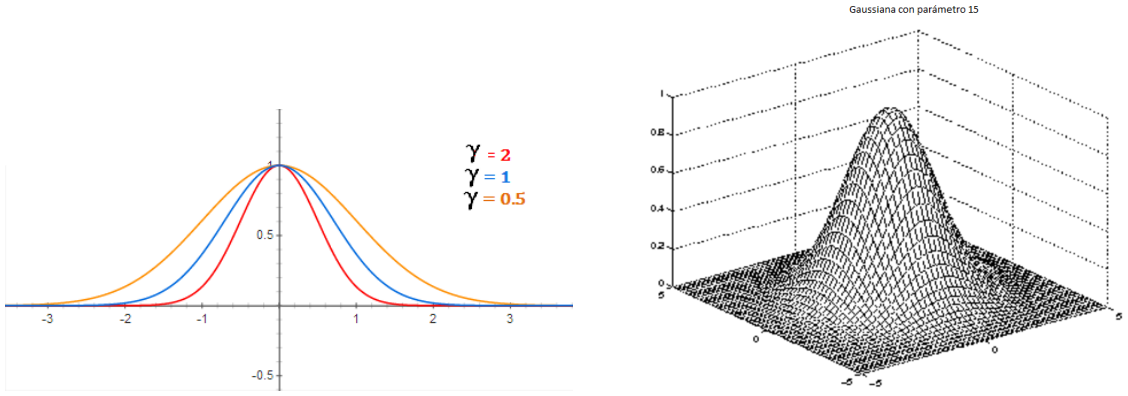
Hay algunas situaciones en las que el núcleo RBF no es adecuado, en particular cuando el número de características es muy grande. En este caso uno puede usar simplemente el núcleo lineal. Es decir, usar el núcleo lineal ya es suficientemente bueno, el mapeo no lineal, no mejorará la situación, y solo basta con estimar el parámetro de penalización C. Si bien dijimos que el núcleo RBF es al menos tan bueno como el lineal, la declaración es válida sólo luego de haber buscado los parámetros (C, γ).

En [9] se divide la discusión entre el núcleo RBF y lineal en tres casos: cuando el número de características es muy grande en relación al tamaño del conjunto de entrenamiento, cuando ambos números son grandes y cuando el número de elementos de entrenamiento es muy grande en relación al número de características. Se realizaron pruebas experimentales con conjuntos de entrenamiento de las tres clases y plantearon la hipótesis que en el primer caso uno podría no necesitar realizar un mapeo de los datos. En el segundo caso conviene usar, por la diferencia abismal en el tiempo que requiere, librerías que resuelven problemas lineales y no las de SVM. Y en el tercero, se mapea a un espacio más grande usando núcleos no lineales, pero si se quiere usar el núcleo lineal, debe usarse LIBLINEAR con la opción -2, también por el poco tiempo que requiere.

2.5.2. ¿Cómo se eligen los parámetros?

Debemos seleccionar los parámetros del núcleo y el parámetro de penalización C. Recordamos que este último surge del problema de clasificación con datos no linealmente separables, es la penalización a la suma de los errores por clasificar linealmente, es decir, a la suma de las variables de holgura, explicado en la Sección 2.2. Con respecto al núcleo, si tomamos el lineal, no tenemos un parámetro extra para calcular. Con respecto a los demás de los más usados, tenemos el parámetro γ en el núcleo RBF, los parámetros γ y d en el polinomial y γ y r en el MLP. Como planteamos en la subsección anterior, vamos a enfocarnos en el núcleo RBF. Es decir, en la selección del parámetro γ, ya que es una buena primera elección y ha sido usando en una gran cantidad de trabajos. Si se quiere estimar otro parámetro, el análisis es análogo.

La estimación de estos parámetros es un tema de investigación abierta y hay varios estudios y métodos al respecto. En [26] menciona que un gran número de experimentos ha demostrado que la propagación del parámetro σ (o respectivamente, γ) de la función de núcleo RBF afecta fuertementa en la actuación general de las SVMs. Para problemas de clasificación, el uso de un σ demasiado grande, o pequeño, conduce a una mala actuación del clasificador. Pensemos que cuando uno elige un núcleo, $K(x, y)$, lo que finalmente usa para clasificar son las n funciones $K(x_i, x)$, donde n es la cantidad de vectores de entrenamiento. En el caso del núcleo RBF, estas funciones son campanas gaussianas con distintos centros. Vemos en las figuras esta función en algún centro, saliendo dese $\mathbb{R}^2$ y $\mathbb{R}$ para entender como influye la variación de σ (ver Figura 6).



(a) (b)

Figura 6: (a) Función gaussiana variando γ, (b) Función gaussiana saliendo de $\mathbb{R}^2$.

Podemos ver que cuando $\sigma \to 0$ (y por lo tanto, $\gamma \to \infty$), el soporte de la función $K(x_i, x)$ se reduce a los valores cercanos de x_i, por lo que llegando al límite se produce $K(x_i, x_j) = \delta_{ij}$, transformandose el problema de optimización:

$$\max_{\alpha} \quad E_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

en:

$$\max_{\alpha} \quad E_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n} \alpha_i^2$$

en el que la solución óptima se encuentra cuando $\alpha_i = 1 \forall i$, y como en general se toma C grande, podemos suponer $C = 1$, y como de las condiciones KKT se obtiene que

$$0 < \alpha_i < C \Rightarrow y_i(w'x_i + b) = 1 \ \wedge \ \xi_i = 0,$$

tenemos que todos los vectores de entrenamiento son vectores soporte, y por ende están bien clasificados, sin embargo, para clasificar un nuevo punto el clasificador realiza

$$h(x) = sign\left(\sum_{j=1}^{n} \alpha_j y_j \Phi(x_j)'\Phi(x) + b\right) = sign(b)$$

si x no se encuentra suficientemente cerca de algún x_j, por lo tanto, los datos que no vemos al momento de entrenar podrían no ser reconocidos con claridad por las SVMs debido al sobre ajuste del modelo.

Por otro lado, cuando $\sigma \to \infty$ (y por lo tanto $\beta \to 0$), todo el conjunto de entrenamiento está incluido en el soporte de todas las funciones $K_j(x) = K(xj, x)$ y todo el conjunto puede ser considerado como un punto. Como resultado, las SVMs podrían no reconocer a algún nuevo punto de dato.

Por ende, las dos situaciones extremas deberían ser evitadas. Experimentalmente se estudió que el error de clasificación varía de grande a pequeño y luego aumenta nuevamente cuando $\sigma \to \infty$. Esta ley muestra la existencia de un apropiado σ el cual podría conducir al clasificador buscado (conclusión similar se sostiene con SVM regresión). Se estudió que cuando σ es mucho más chico que la menor de las distancias que existe entre los vectores del conjunto de entrenamiento tomados de a dos, o mucho más grande que la mayor de estas distancias, aparecen estas situaciones extremas mencionadas.

Vemos a continuación diferentes métodos que encontramos en la literatura para la elección de los parámetros que necesitamos determinar en la resolución de un problema de SVM. Dentro del comando *svmtrain* que ofrece Matlab para entrenar el clasificador, tenemos la opción de introducir que parámetros deseamos, tanto en el núcleo como el de penalización, y por default estos son $\sigma = 1$ en el caso del núcleo RBF, y $C = 1$ en el caso del parámetro de penalización.

(a) Método de discriminación lineal de Fisher (para la selección de σ).

En [26] se estudia la determinación de σ para problemas de clasificación y regresión. Para el primero, que es el de nuestro interés, el óptimo σ es calculado en base a la discriminación de Fisher.

La discriminación lineal de Fisher (FLD en sus siglas en inglés) aplicada en el tradicional reconocimiento de patrones, contrariamente a las SVMs, mapea los datos originales a una recta, y luego, los separa en la misma. El problema clave en FLD es elegir un vector

proyección ω apropiado, el cual puede separar los datos fácilmente. En [6] se provee una regla para seleccionar este vector proyección: para los datos sobre la proyección, la distancia entre los datos que pertenecen a distintas clases, la distancia debería ser la mayor posible, mientras que la distancia entre los datos de la misma clase debería ser la más chica posible.

Si ω es el vector proyección, y denominamos $P_\omega : \mathbb{R}^p \to \mathbb{R}$ a la función proyección en base al vector ω, la función discriminante de Fisher es definida como:

$$J_F(\omega) = \frac{|m_1 - m_2|^2}{s_1^2 + s_2^2},$$

donde m_i y s_i denotan la media y varianza de la clase i, donde $i = 1, 2$. Además:

$$\begin{aligned} m_i &= \frac{1}{n_i} \sum_{k=1}^{n_i} P_\omega(x_{ki}), \quad i = 1, 2, \\ s_i^2 &= \sum_{k=1}^{n_i} \left(P_\omega(x_{ki}) - m_i\right)^2, \quad i = 1, 2, \end{aligned}$$

donde x_{ki} denota los datos de la i-ésima clase, y n_i la cantidad de datos de la misma. Luego, el óptimo vector proyección se obtiene maximizando $J_F(\omega)$.

La idea análoga puede ser empleada para determinar σ en el núcleo RBF. Si tomamos N_1 y N_2 la cantidad de elementos de la clase con etiqueta $+1$ y -1 respectivamente, $\{x_{1i}, i = 1, \ldots, N_1\}$ y $\{x_{2i}, i = 1, \ldots, N_2\}$ los elementos de las clases, Φ la función que introduce los datos a un espacio de mayor dimensión y genera el núcleo $K(x_1, x_2) = \Phi(x_1)' \Phi(x_2)$, redefinimos la media de la siguiente manera:

$$\begin{aligned} m_1 &= \frac{1}{N_1} \sum_{i=1}^{N_1} \Phi(x_{1i}), \\ m_2 &= \frac{1}{N_2} \sum_{i=1}^{N_2} \Phi(x_{2i}), \end{aligned}$$

quedando la distancia entre las dos clases de la siguiente manera:

$$\begin{aligned} \|m_1 - m_2\|^2 &= (m_1 - m_2)'(m_1 - m_2) \\ &= \frac{1}{N_1^2} \sum_{i=1}^{N_1} \sum_{j=1}^{N_1} K(x_{1i}, x_{1j}) + \frac{1}{N_2^2} \sum_{i=1}^{N_2} \sum_{j=1}^{N_2} K(x_{2i}, x_{2j}) \\ &- \frac{2}{N_1 N_2} \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} K(x_{1i}, x_{2j}). \end{aligned}$$

Las varianzas de las dos clases están definidas por:

$$\begin{aligned} s_1^2 &= \sum_{i=1}^{N_1} \|\Phi(x_{1i}) - m_1\|^2 = N_1 - \frac{1}{N_1}\sum_{i=1}^{N_1}\sum_{j=1}^{N_1} K(x_{1i}, x_{1j}), \\ s_2^2 &= \sum_{j=1}^{N_2} \|\Phi(x_{2j}) - m_2\|^2 = N_2 - \frac{1}{N_2}\sum_{i=1}^{N_2}\sum_{j=1}^{N_2} K(x_{2i}, x_{2j}). \end{aligned}$$

Dada

$$f(\sigma) = \frac{s_1^2 + s_2^2}{\|m_1 - m_2\|^2},$$

el óptimo σ puede ser obtenido minimizando esta función.

(b) Búsqueda en grilla:

Estudiamos búsqueda en grilla y validación cruzada de un tutorial de Thorsten Joachims [10] como punto de partida, y de [9]. El objetivo de la búsqueda en grilla (grid–search) es identificar el par (C, σ), donde C es el parámetro de penalización y σ es el parámetro del núcleo RBF, tal que el clasificador con dichos parámetros prediga con la mayor exactitud posible los datos que no conocemos. Una estrategia común es separar los datos en dos partes, la cual una es considerada como no conocida (ésta será el conjunto de testeo). La precisión obtenida de la predicción en el conjunto que "no se conoce" refleja de una manera mucho más acertada la actuación del clasificador sobre un conjunto independiente de los datos. Una versión mejorada de este proceso es conocida como validación cruzada.

Con respecto a la estimación de los parámetros, se plantea que es necesario optimizar ambos de manera simultánea con C, ya que el óptimo C depende de los parámetros del núcleo. El método común es buscar combinaciones de parámetros a través de la búsqueda exhaustiva y seleccionar los parámetros del núcleo y costo a través de la típica validación cruzada. Un enfoque avanzado, por ejemplo minimizando la función planteada por el método de Fisher para encontrar σ, evita la búsqueda exhaustiva para mejorar la eficiencia de la búsqueda, sobre todo cuando se quieren optimizar más parámetros. En [3] se plantea un enfoque para un gran número de parámetros (más de 100), introduciendo un marco para estimar las posibles funciones de error utilizando métodos de descenso.

La grilla en la que se realiza la búsqueda es la formada por escalas pre-establecidas tanto para σ como para C. Una de las más usadas es la que utiliza

$$\sigma \in \left[2^{-15}, 2^{-13}, \ldots, 2^{3}\right]$$

y

$$C \in \left[2^{-5}, 2^{-3}, \ldots, 2^{15}\right].$$

Una variación que suele utilizarse para la elección del parámetro C, es escalar con los valores de las características, es decir, con el conjutno de entrenamiento. Por ejemplo, un

valor default típico para comenzar es el dado por:

$$C_{def} = \frac{1}{\sum_{i=1}^{N} K(x_i, x_i)},$$

donde N es la cantidad de vectores de entrenamiento y K la función de núcleo seleccionada. Luego, se realiza una búsqueda para C sobre una escala que depende de C_{def}, por ejemplo,

$$C \in \left[10^{-4} C_{def}, \ldots, 10^{4} C_{def}\right].$$

Luego, dentro de la grilla propuesta, se selecciona el par (C, σ) óptimo vía validación cruzada o vía aproximación dejar–uno–afuera. Para realizar una validación cruzada de m partes, se divide el conjunto de entrenamiento en m subconjuntos de igual tamaño. Secuencialmente, un subconjunto es testeado usando el clasificador entrenado con los restantes $m-1$ subconjuntos. Así, cada elemento del conjunto de entrenamiento es predecido una vez. Luego, la exactitud de la validación cruzada es el porcentaje de datos que son clasificados correctamente. El procedimiento de la validación cruzada puede evitar el sobreajuste del problema.

(c) Algoritmos genéticos:

Utilizamos el trabajo [12] y [21] para estudiar este tema. Un Algoritmo Genético (AG) es una técnica de búsqueda que utiliza como principio la evolución natural. En un AG una solución es representada por un cromosoma y cada elemento del cromosoma se denomina gen. El algoritmo toma un conjunto de cromosomas, al que llama población, y ésta evoluciona a través de un número de generaciones de la siguiente manera. Primero, se seleccionan dos soluciones de la población basadas en una cierta distribución de probabilidad, y estos serán los cromosomas padres. Luego, por medio de combinaciones de los padres, surge la descendencia, a través de la operación cruce. Luego una operación mutación modifica la descendencia con una baja probabilidad. La descendencia puede ser localmente mejorada por otro algoritmo o procedimiento heurístico. Una generación se completa al reemplazar cada miembro de la población por la descendencia.

Se corre un considerable número de generaciones hasta que algún criterio de parada es alcanzado. Finalmente, el AG devuelve como la mejor solución la población como solución del problema.

El proceso de AG para la optimización de los hiper–parámetros para SVMs es descripta de la siguiente forma:

(i) Inicialización: generar una población inicial aleatoria de n cromosomas. Estos tienen que ser puntos factibles del problema.

(ii) Evaluación de aptitud: evaluar la aptitud $f(x)$ para cada cromosoma.

(iii) Selección: seleccionar dos cromosomas padres acorde a sus aptitudes para la reproducción utilizando el método de la rueda de la ruleta. Éste se basa en dividir una rueda en tantas porciones como cromosomas haya, de forma tal que el área de cada

porción es proporcional a la aptitud de un cromosoma, y esta razón es calculada de la siguiente forma:

$$R_f = \frac{f(i)}{\sum_{i=1}^{n} f(i)} * 100\,\%,$$

donde $f(i)$ es la aptitud del i-ésimo cromosoma. Se ordenan las porciones por tamaño, de mayor a manor, se elige un número aleatorio del intervalo [0,1] y este paso devuelve el individuo situado en esta posición.

(IV) Cruce: formar una nueva descendencia a partir de los padres. Hay diferentes métodos para lelvar a cabo este cruce: cruce de un punto , de dos puntos, o cruce uniforme. Ambos se basan en crear hijos con parte de los padres, estas partes pueden haber sido separadas por un punto, dos, o seleccionada por una máscara, respectivamente.

(V) Mutación: modificar algunos genes de cada elemento de la nueva descendencia, generalmente uno solo, con una cierta probabilidad, por ejemplo, uniforme. Algunas opciones son: Incrementar o decrementar a un gen una pequeña cantidad generada aleatoriamente, o, multiplicar un gen por un valor aleatorio próximo a 1.

(VI) Siguiente generación: formar la siguiente generación de la pre-población.

(VII) Prueba: si el número de generaciones excede el valor inicializado como máximo, se detiene, y se retorna el mejor cromosoma de la población actual como solución.

(VIII) Ir al paso (II).

El algoritmo devuelve el mejor cromosoma, basado en las probabilidades asignadas para cada punto, y éste brindará los valores para crear el clasificador. Otra manera es que de la última generación se elige la mejor vía validación cruzada, como explicamos en el ítem anterior.

Notemos que, utilizando este algoritmo, no sólo elegimos los parámetros sino que también resolvemos el problema de optimización cuadrática. Pero si elegimos los parámetros como en el ítem anterior, esto debe hacerse de manera aislada y sobre esta parte trataremos en la subsección siguiente.

2.5.3. ¿Cómo se resuelve el problema de optimización?

Recordemos el problema que debemos resolver:

$$\begin{aligned} \text{mín} \quad & E_D(\alpha) = \frac{1}{2}\alpha' K \alpha + c'\alpha \\ \text{sujeto a} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, n, \end{aligned}$$

donde $K = (K_{ij})$ con $K_{ij} = y_i y_j K(x_i, x_j)$ y $c = (-1, -1, \ldots, -1)$, vector de dimensión n.

Éste, es un problema de optimización cuadrática, para el cual Matlab nos ofrece tres opciones de métodos para encontrar una solución: SMO (optimización secuencial minimal), LS (cuadrados mínimos) y QP (programación cuadrática). [16]

SMO es un método iterativo que surge para resolver el problema de optimización de SVM. Divide al problema en una serie de posibles sub-problemas más pequeños que pueden ser resueltos analíticamente. Por la restrincción de igualdad tenemos que como mínimo contamos con dos multiplicadores de Lagrange. La idea es ir tomando de a pares (α_i, α_j) y llevar el problema a encontrar un mínimo de función cuadrática de una dimensión. Se utilizan métodos eurísticos para elegir el par de multiplicadores y acelerar la convergencia. Esta elección es crítica para grandes conjuntos de datos donde hay $n(n-1)$ posibles opciones de α_i y α_j. [20]

LS resuelve un conjunto de ecuaciones lineales en lugar del problema cuadrático, que se obtiene a partir de construir la función lagrangiana de una reformulación del problema primal de SVM. En esta reformulación se modifica minimizar el error $\sum_{i=1}^{n} \xi_i$ por minimizar la suma de los errores cuadráticos, es decir, por $\sum_{i=1}^{n} \xi_i^2$ y se cambia ξ_i por $y_i - (w'\Phi(x_i) + b)$, quedando como resultado minimizar

$$\mu(\frac{1}{2}w'w) + \zeta \left(\frac{1}{2} \sum_{i=1}^{n} (y_i - (w'\Phi(x_i) + b))^2 \right),$$

donde μ y ζ son utilizados luego para proporcionar una interpretación bayesiana. [24]

Cuando llama al método QP, matlab recurre al código *quadprog*, y dentro de éste hay varios mecanismos que tratan de aprovechar la estructura cuadrática de la función objetivo para resolver el problema de optimización. Entre estos mecanismos *svmtrain* toma por defecto el método *active–set* (conjunto de restrincciones activas), pero también se puede optar por *interior–point–convex* (punto interior). Ambos son métodos iterativos.

El método de restricciones activas resuelve una sucesión de problemas con restricciones de igualdad. Si en un problema de optimización el conjunto de restricciones lineales está dado por $a_j'x \leq b_j, j = 1, \ldots, r$, en cada iteración k, se busca una dirección de descenso en el subespacio

$$S(y^k) = \{d | a_j'd = 0, j \in A(y^k)\},$$

donde $A(y) = \{j | a_j'y = b_j, j = 1, \ldots, r\}$. Este subespacio es el paralelo a la variedad de restrincciones activas. Luego, hay dos posibilidades: se encuentra una dirección d^k y se define $y^{k+1} = y^k + \alpha^k d^k$, donde el paso α^k es encontrado con alguna regla, o bien, no se encuentra una dirección de descenso $d \in S(y^k)$ porque y^k es estacionario sobre la variedad $y^k + S(x^k)$. En este último caso hay dos posibilidades: o x^k es estacionario en todo el conjunto factible, y en este caso el algoritmo se detiene, o bien una de las restricciones, $\bar{j}$, es relajada y se encuentra una dirección de descenso en el subespacio

$$\bar{S}(y^k) = \{d | a_j'd = 0, j \in A(y^k), j \neq \bar{j}\}.$$

Esta dirección d^k se calcula en base a una aproximación cuadrática de la función en cada iteración para una función en general, y en este caso que ya es cuadrática, en base a los mismos términos de la función. [1, p. 230 – 235]

Los métodos de puntos interiores, también llamado métodos de barrera, es una clase de algoritmos que resuelve problemas de optimiación lineal y no lineal convexos. Supongamos que el conjunto de restrincciones está dado por $g_j(x) \leq 0, j = 1, \ldots, n$, donde g_j son funciones

continuas a valores reales y X es un conjunto cerrado. El interior, relativo a X, del conjunto definido por las restrincciones de desigualdad es

$$S = \{x \in X | g_j(x) < 0, j = 1, \ldots, r\},$$

y sobre el interior de S se define una función $B(x)$ que se agrega a la función objetivo y se denomina función barrera. Esta función es continua y tiende a ∞ cuando las funciones restrincción g_j se aproximan a 0. Las dos funciones más usadas son:

$$B(x) = -\sum_{j=1}^{r} \text{lm}\{-g_j(x)\}, \text{logarítmica},$$

$$B(x) = -\sum_{j=1}^{r} \frac{1}{g_j(x)}, \text{inversa}.$$

Notar que ambas funciones barreras son convexas si las restrincciones son convexas. El método de barrera es introducido a través de una suseción de parámetros $\{\epsilon^k\}$ con

$$0 < \epsilon^{k+1} < \epsilon^k, k = 0, 1, \ldots, \epsilon^k \to 0.$$

Éste, consiste en encontrar

$$x^k = \text{argmin}_{x \in S}\{f(x) + \epsilon^k B(x)\}, k = 0, 1, \ldots$$

Como la función barrera es definida en el interior del conjunto S, las sucesivas iteraciones de algún método para esta minimización deberían ser puntos interiores. Si ahora $X = \mathbb{R}^n$, uno podría usar métodos de optimización sin restrincciones con un paso adecuado y se asegura que todas las iteraciones viven en S tomando un punto inicial en este conjunto. Además, $\epsilon^k B(x) \to 0 \ \forall x \in S$ cuando $\epsilon^k \to 0$. Luego, esta demostrado que todos los puntos límites de una secuencia $\{x^k\}$ generada por el método de barrera es un mínimo global del problema original. [1, p. 312 – 327]

Estos son sólo algunos métodos de resolución del problema cuadrático. En la literatura que revisamos no se menciona alguno en particular para resolver el problema de SVM, al haber mucha maquinaria desarrollada para este tipo de problemas simplemente se selecciona alguna. Nosotros tuvimos en cuenta sólo estas opciones que Matlab ofrece y entre ellas elegiremos alguna para utilizarla a lo largo de nuestro trabajo.

2.5.4. Algoritmos

En 1998 Gunn muestra en [7, p. 45–47] algunos algoritmos implementados en Matlab para la rutina de clasificación mediante SVMs, y en [7, p. 51] ofrece instrucciones y links para adquirir gratuitamente un Toolbox de SVMs para Matlab.

Para nuestro trabajo recurrimos a Matlab [16] para visualizar las herramientas disponibles para el entrenamiento y clasificación con SVMs. Los comandos que ofrece son dos: *svmtrain* y *svmclassify*.

En *svmtrain* ingresa como mínimo los datos de entrenamiento como filas de una matriz, la etiqueta de los vectores de entrenamiento como un vector columna. Luego, hay una serie de parámetros que tienen una determinación por defecto, pero se puede nombrarlos y de manera consecutiva especificarlos, entre estos tenemos el núcleo a utilizar (*'kernel_function'*), una vez elegido éste se puede elegir el parámetro (*'rbf_sigma', 'polyorder',* etc.), el método de resolución (*'method'*), el parámetro de penalización (*'boxconstraint'*), entre otros. Este algoritmo devuelve una estructura dentro de la que podemos encontrar los vectores soporte como filas en una matriz, un vector columna con los multiplicadores de Lagrange, un vector columna indicando los índices de los vectores soporte, entre otras salidas. *svmclassify* toma la estructura que devuelve *svmtrain* junto con un conjunto de vectores para clasificar que tienen que encontrarse como filas de una matriz y tienen que tener la misma dimensión que los vectores de entrenamiento. Utilizaremos permanentemente estos comandos a lo largo del trabajo, determinando algunos parámetros y utilizando otros por defecto.

3. Aplicación de SVM a pérdidas no técnicas de electricidad – Casos de Estudio

Las SVMs han sido elegidas y utilizadas por diversos autores para trabajar sobre problemas de clasificación de variada índole como mencionamos en la Sección 1.2. Dentro de esta variedad, nos concentramos en el análisis de las pérdidas no técnicas en una red de energía eléctrica. En este caso, el conjunto de entrenamiento es un conjunto de historiales de consumos mensuales obtenidos del medidor, cada uno de un hogar distinto, y luego de ser inspeccionado por un técnico de la distribuidora, se lo cataloga como consumidor fraudulento o confiable, teniendo así dos clases. Suele usarse en lugar de los datos puntuales, o junto con estos, promedios de ellos y/o índices que utilicen estos datos, u otros relacionados con los clientes, con el fin de generar una clasificación más eficaz. El propósito del modelo es preseleccionar clientes sospechosos para ser inspeccionados basados en comportamientos anormales de consumo.

Analizaremos los trabajos [18] y [19], realizados en 2008 y 2009 respectivamente. Estos artículos presentan una metodología para detectar consumidores fraudulentos utilizando SVMs. En base al primer trabajo, también analizamos el trabajo [9], donde se brinda una guía práctica para utilizar SVMs en casos aplicados en general.

3.1. Trabajo de Nagi et al. (2008) en Malasia

En el trabajo [18] se presenta un marco para identificar y detectar pérdidas no técnicas de electricidad (o NTL en sus siglas en inglés). Se desea determinar a clientes con consumos irregulares o anormales que pueden indicar actividades fraudulentas. Una combinación de métodos para extraer rasgos automáticamente con SVMs es utilizado para identificar consumidores fraudulentos. Los patrones de consumo de los clientes son extraídos utilizando data mining (definido esto como el proceso de extracción no trivial de información implícita, previamente desconocida y potencialmente útil) y técnicas estadísticas, creando así perfiles de clientes de ambas clases.

Se cuenta con datos de 265870 clientes de Kuala Lumpur durante un período de 25 meses, desde Junio de 2006 a Junio de 2008. Se realiza un filtro y selección de estos datos, utilizando técnicas estadísticas, con el fin de eliminar clientes repetidos, clientes que tengan consumo nulo durante todo el período, y clientes que no cuenten con 25 datos, es decir, consumidores nuevos que se adhirieron después del primer mes. Luego de este filtro quedan solo 186968 clientes, de los cuales 1171 fueron detectados como fraudulentos. Los pasos más importantes del preprocesamiento realizado fueron ilustrados por los autores en la Figura 7, de la cual explicaremos brevemente cada paso a continuación.

Se cuenta con datos de 25 meses por consumidor, y con esto se calculan 24 valores diarios promedio. Si hay M consumidores, se calcula

$$x_h^{(m)} = \frac{P_{h+1}}{D_{h+1} - D_h}, \quad m = 1, \ldots, M, \quad h = 1, \ldots, 24,$$

donde P_{h+1} representa el consumo mensual en el siguiente mes, y $D_{h+1} - D_h$ la diferencia de los días entre la lectura de los datos del siguiente mes con el corriente. A estos 24 valores se le agregó

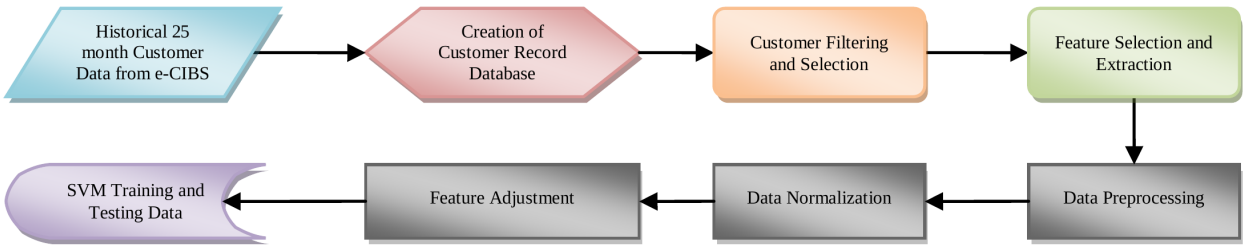


Figura 7: Diagrama de flujo para el pre-procesamiento de datos.

uno adicional a cada cliente el cual representa la confiabilidad del cliente, al que denominan CWR (Credit Worthiness Rating), ya que apunta a detectar aquéllos que intencionadamente evitan pagar sus cuentas y demoran en realizarlo. Los califican con un valor en un rango de 0.00 a 5.00, el cual lo obtienen promediando datos acerca del estado mensual de pago del cliente en 25 meses. Por ende, fueron seleccionados 25 rasgos para el clasificador: 24 valores de consumo promedio y 1 CWR.

Se realiza un preprocesamiento de los datos con la finalidad de eliminar el ruido o inconsistencias que puedan llegar a tener los mismos. Se utilizan técnicas estadísticas para convertir el conjunto de entrenamiento en un nuevo conjunto donde todos los datos estén completos (puede ocurrir que alguna lectura no haya podido ser realizada).

Luego los datos son normalizados para ser bien representados en el clasificador SVM. La manera en la que lo realizan es la siguiente:

$$NL = \frac{L - \text{mín}\,(L)}{\text{máx}\,(L) - \text{mín}\,(L)},$$

donde L representa el consumo de un cliente en un mes dado, y $\text{mín}\,(L)$ y $\text{máx}\,(L)$ representan el mínimo y el máximo de los valores del conjunto de 24 meses de consumo. Notemos que cuando $L = \text{mín}\,(L)$ entonces $NL = 0$, y cuando $L = \text{máx}\,(L)$ entonces $NL = 1$. Por ende cada vector de 24 consumos promedios es llevado a un nuevo vector con valores entre 0 y 1. Notemos también que no se normaliza el factor CWR.

Estos datos normalizados junto con las etiquetas de los consumidores son representados en una matriz que cuenta con tantas filas como cantidad de consumidores con los que se entrena el clasificador (en este caso 186968), y tantas columnas como cantidad de características hay (en este caso 25).

Una vez que tenemos los datos normalizados y completos se procede a entrenar el clasificador. Fueron utilizadas cuatro clases para representar cuatro diferentes perfiles de consumo.

- Clase 1 (Sospechoso Fraudulento Confirmado): en 1171 casos de fraude, los cuales fueron inspeccionados manualmente, se detectaron con claridad cambios abruptos en sus perfiles.
- Clase 2 (Sospechoso Fraudulento No Confirmado): son clientes que poseen cambios abruptos en sus perfiles como así también oscilaciones, pero no han sido inspeccionados.

- Clase 3 (Sospechoso Limpio Confirmado).
- Clase 4 (Sospechoso Limpio No Confirmado).

Sólo perfiles de 131 clientes fraudulentos, del total de 1171, fueron utilizados para entrenar al clasificador, ya que el resto de los perfiles no contaban con patrones anormales característicos de los casos de fraude. Probablemente estos clientes hayan realizado una conexión ilícita de electricidad antes de los dos años del período de estudio.

El clasificador tiene desbalanceadas las clases, y por esto es ponderado para equilibrar la relación entre las mismas. Los pesos fueron calculados dividiendo el número total de muestras del clasificador por las muestras de cada clase y multiplicado por 100. Notemos que al dividir por el número de muestras de la clase, mientras más grande es ésta, menos peso tendrá, y viceversa. Así, obtenemos cuatro clases con el mismo peso, para que sólo tenga influencia la información de cada una y no la cantidad de muestras de las mismas.

El trabajo propuesto para optimizar los parámetros de las SVMs es presentado en el trabajo con la Figura 8. Este gráfico muestra el procedimiento que se sigue para la elección de los parámetros γ y C utilizando búsqueda en grilla y validación cruzada, basado en el trabajo [9], explicado en la Sección 2.5.2.

En este caso fueron utilizadas secuencias crecientes y exponenciales de (C, γ), donde $C = [2^{-5}, 2^{-3}, \ldots, 2^{15}]$ y $\gamma = [2^{-15}, 2^{-13}, \ldots, 2^{3}]$, y para cada par (C, γ) se evalúa su efecto en el modelo, vía validación cruzada, calculando la exactitud del modelo. Con exactitud se refiere al porcentaje de etiquetas acertadas por el clasificador. Luego de elegir estos parámetros, se calcula otro índice para analizar la actuación del modelo que se denomina tasa de éxito. Este es el porcentaje de fraudes acertados, es decir, cuántos de los etiquetados como fraudulentos por el clasificador realmente cometieron fraude. Ambos índices son utilizados por Nagi et al. a lo largo de su trabajo.

Experimentalmente encontraron que los parámetros óptimos fueron $C = 1$ y $\gamma = 0.92$, es decir, dieron la exactitud y tasa de éxito más alta. La exactitud del modelo fue de 86.43 % y la tasa de éxito de 77.41 %.

Luego de ser realizado este trabajo con los datos de Kuala Lumpur, se realizó este procedimiento en otras tres ciudades del estado de Kelantan en Malasia, de las cuales también ya se tenían datos de inspecciones previas, evaluando la actuación del modelo desarrollado. Para cada base de datos, ya preprocesada, se realiza una búsqueda de los parámetros óptimos, buscando en cada par de ellos una exactitud del clasificador mayor al 70 %. Luego, una vez elegidos los parámetros, se construye el clasificador y se clasifica a los clientes, integrando a la base de datos de los mismos la información que devuelve el sistema, teniendo como resultado una lista de clientes sospechosos de fraude. Como resultado de este testeo piloto se obtuvo un promedio de exactitud de 72.60 %. Este proceso de predicción propuesto por los autores para detectar los fraudes en el consumo de electricidad es ilustrado en la Figura 9.

Los resultados obtenidos en el testeo piloto devuelven una tasa de éxito de 22 %, es decir, de la lista de sospechosos emitida por el modelo, solo el 22 % realmente fue etiquetado como fraudulento en las inspecciones previas, el otro 78 % fue designado por el sistema como fraudulento por tener un perfil de consumo similar a un caso de fraude, aunque en realidad según los autores, los cambios que se observan se deben a que la propiedad tiene dañado el medidor de

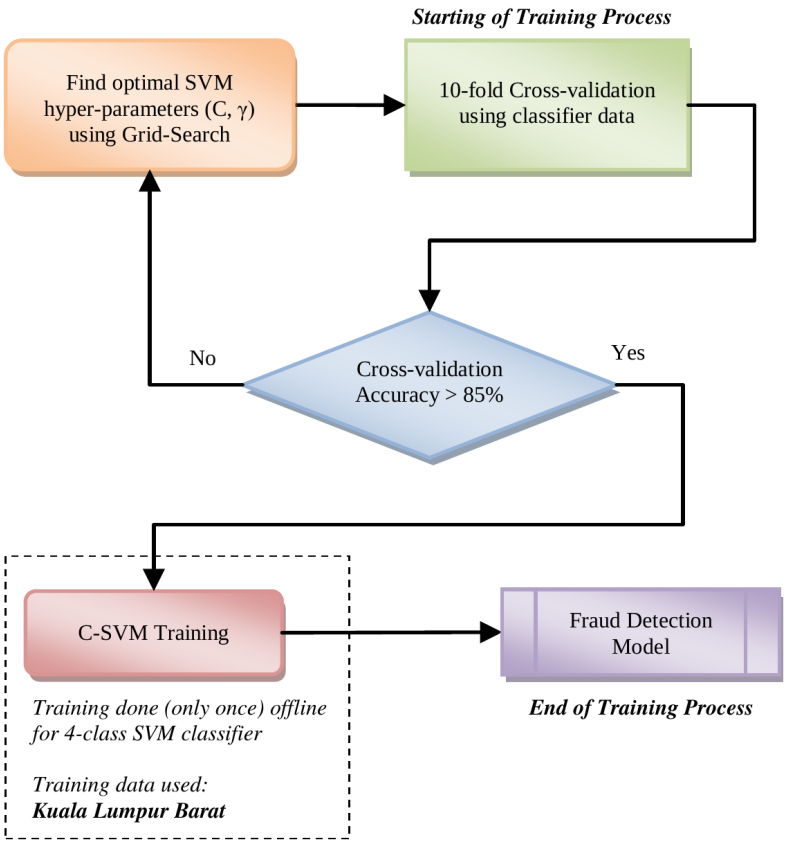


Figura 8: Diagrama de flujo para la optimización de parámetros de SVMs.

la misma, o está dañada la red eléctrica, o la propiedad tuvo un cambio de propietario, entre otras posibles razones.

Para evitar esto, inspeccionaron los perfiles de manera manual buscando características comunes que permitan distinguir estos casos de los casos de fraudes reales. Se utilizaron condiciones de filtro a partir de cotas para los valores de los dos últimos consumos promedios, del consumo promedio máximo y mínimo, y de la diferencia entre el máximo y mínimo. Estas cotas fueron establecidas manualmente, y si un consumidor no cumplía con algunas de estas condiciones, dejaba de ser contado como sospechoso. De esta forma, la tasa de éxito aumentó de 22 % a un aceptable 53 %.

Como conclusión los autores presentan una novedosa técnica para la detección de las pérdidas no técnicas utilizando SVMs, eligiendo sus parámetros con técnicas de validación cruzada, y con previo preprocesamiento de los datos. De esta manera logran obtener una tasa de éxito superior al 50 % y una exactitud superior al 70 %.

3.2. Trabajo de Nagi et al. (2009) en Malasia

Este trabajo fue realizado por Nagi et al. en el año 2009 [19].

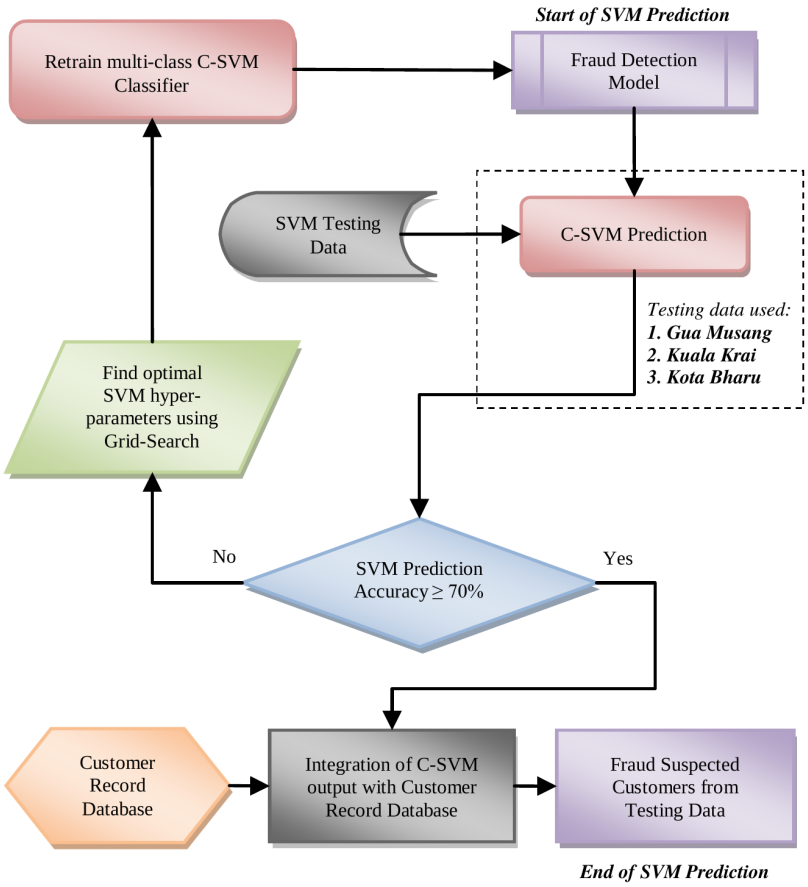


Figura 9: Diagrama de flujo del modelo de predicción para detectar fraudes en el consumo eléctrico.

Recordamos el problema de optimización del Lagrangiano dual (OLD) proveniente del planteo de SVMs. Buscamos la solución de:

$$\begin{aligned} \max_{\alpha} \quad & E_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \\ \text{sujeto a} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, n. \end{aligned}$$

Para crear el clasificador necesitamos encontrar un valor óptimo para el parámetro de penalización C, y para el parámetro del núcleo, que denominamos γ, para el caso del núcleo RBF, y de los multiplicadores de Lagrange $(\alpha_1, \ldots, \alpha_n)$. A este conjunto de parámetros se les denomina hiper–parámetros.

En el trabajo explicado en 3.1 se encuentran valores óptimos de los primeros dos parámetros a través de una búsqueda en grilla. Para cada par (C, γ) de esta grilla los multiplicadores de Lagrange son encontrados resolviendo el problema cuadrático resultante. Una vez fijados todos

los valores, se evalúa la actuación del clasificador vía validación cruzada, y en base a esto se eligen los mejores parámetros.

En cambio, en el trabajo que estamos describiendo [19], se trabaja de manera simultánea con todos los parámetros. Utilizando un algoritmo genético se obtienen 10 tiras

$$(\alpha_1, \ldots, \alpha_n, C, \gamma)\,,$$

las cuales se denominan cromosomas, y una de ellas brinda todos los valores óptimos. Se elige la mejor utilizando, nuevamente, validación cruzada. Estudiaremos entonces en este trabajo un enfoque híbrido entre el uso de un algoritmo genético y SVMs.

El trabajo propuesto para optimizar los hiper–parámetros es presentado en el paper con la Figura 10. Este diagrama grafica el proceso del algoritmo genético para obtener los resultados potenciales y a continuación el proceso de elección de alguno de estos resultados vía validación cruzada. Se basa en el trabajo [12], explicado en la Sección 2.5.2. La función que se utiliza para evaluar la aptitud de los cromosomas es la función objetivo del OLD.

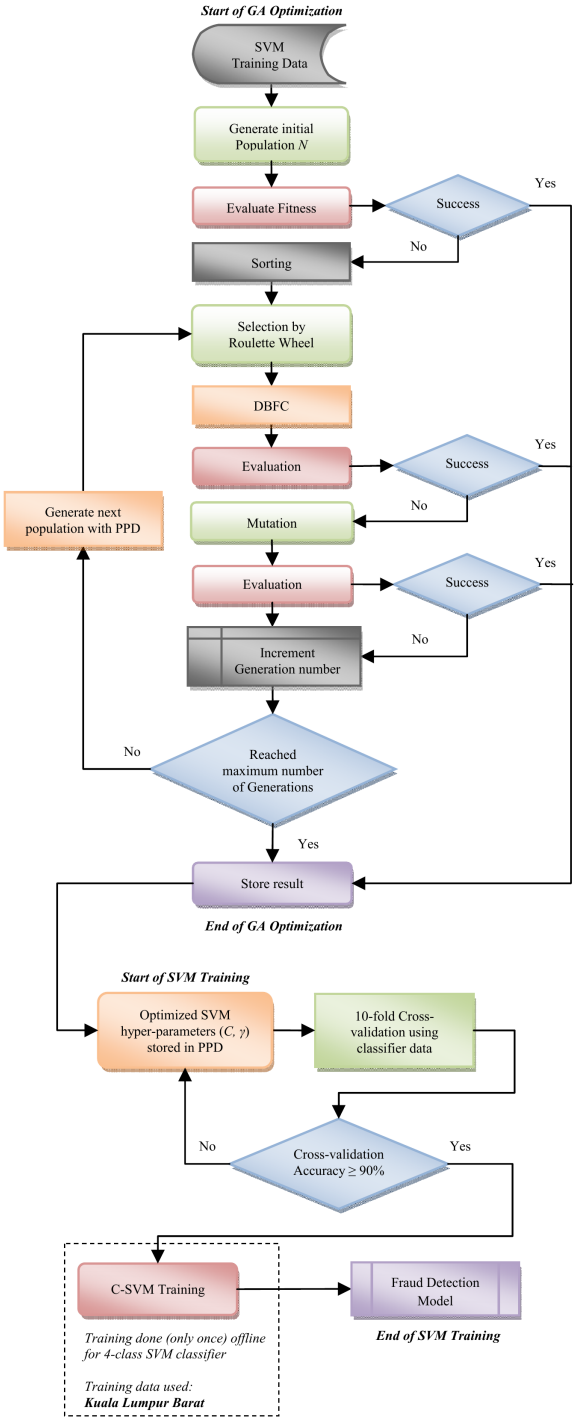


Figura 10: Diagrama de flujo de la combinación GA-SVM.

Si se usa el núcleo RBF, los hiper–parámetros del problema vimos que son: los multiplicadores de Lagrange $(\alpha_1, \ldots, \alpha_i)$, C y γ. Todos los parámetros son codificados en un cromosoma

X representados de la siguiente manera: $X = (\alpha_1, \ldots, \alpha_n, p_1, p_2)$, donde n es el número de características de entrenamiento, y p_1 y p_2 representan C y γ.

Para cada uno de los 10 pares obtenidos en la primer población, la actuación es medida entrenando con el 70 % del conjunto de entrenamiento y verificando con el restante 30 %. Se realizó este proceso con un número máximo de 500 generaciones, una población de tamaño 1000, un índice de cruce de 0.8 y un índice de mutación de 0.025. Estos parámetros necesarios para realizar el proceso de optimización fueron encontrados experimentalmente. Fueron los más adecuados para obtener la mayor exactitud y tasa de éxito. De todos los 10 pares de (C, γ), los hiper–parámetros seleccionados fueron: $C = 20.476$ y $\gamma = 0.2608$. Usando estos parámetros y realizando las validaciones cruzadas, la exactitud más grande encontrada fue de 92.58 %.

Se ve en la primera parte del diagrama de la Figura 10 la estructura de un algoritmo genético para encontrar una generación de la cual elegiremos el cromosoma óptimo. La manera de elegirlo es utilizando validación cruzada, y esto está representado en una segunda parte del esquema. Se busca en esta elección una exactitud mayor al 90 %, 20 % más de lo que se esperaba en el trabajo anterior.

Se evalúa el modelo con los mismos datos del trabajo anterior, es decir, de otras tres ciudades del mismo estado en el que sea encuentra Kuala Lumpur. El resultado de la evaluación en estas tres ciudades fue una exactitud promedio de 81.63 %, promedio mayor al encontrado en el primer trabajo que fue del 72.60 %. Luego se mejora la tasa de éxito de la misma forma que en el trabajo anterior, estableciendo manualmente condiciones de filtro para eliminar los clientes clasificados como fraudulentos y los que no lo son. Como las condiciones dependen del clasificador, éstas varían de un trabajo a otro. Sin embargo, el cálculo y la aplicación se realizan de manera análoga. De esta manera, se mejora la tasa de éxito de 37 % a 62 %.

Como conclusión los autores presentan una técnica híbrida entre el uso de algoritmos genéticos y SVMs para resolver el problema de clasificación, al igual que antes, con previo preprocesamiento de los datos. Logran obtener una tasa de éxito superior al 60 % y una exactitud superior al 80 % para la detección de las pérdidas no técnicas, mejorando los resultados obtenidos en el trabajo anterior.

3.3. Ideas para nuestro trabajo

Nosotros contamos con 106551 datos de 9330 consumidores distintos de la ciudad de Córdoba. Aproximadamente son 10 datos por consumidor, en algunos casos tenemos más, en otros menos. Y notamos que tenemos un pequeño porcentaje de consumidores fraudulentos en relación al total, por ende, en base a esto comenzamos a esbozar una línea para luego llegar a implementar el clasificador.

La idea principal que extrajimos de los trabajos expuesto es la estructura que se presenta para elaborar un clasificador: primero el tratamiento de los datos, luego la extracción de características, en la cual se incluye anexo de índices y reescrituras de los datos, luego la normalización de los datos, y del primer trabajo tomaremos la idea de búsqueda en grilla y análisis vía validación cruzada para la selección de los parámetros.

Tomaremos también la idea del cálculo de ítems para analizar la actuación del calsificador a lo largo de las modificaciones. De ambos trabajos utilizamos la idea de analizar de manera

manual los datos de los consumidores por separado para extraer características específicas que ayuden a distinguir mejor los conjuntos, y utilizamos esta ídea en la creación del ínice por caídas que explicaremos más adelante.

Tratamos entonces de utilizar la estructura de los trabajos propuestos para ciudades de Malasia, para junto con ella, aportes en general de toda la teoría planteada hasta el momento e ideas propias, poder contruir un clasificador binario que sirva como herramienta para detectar las pérdidas no técnicas de electricidad de la ciudad de Córdoba.

4. Detección de pérdidas en la ciudad de Córdoba

El objetivo de nuestro trabajo es estudiar un sistema inteligente para asistir al equipo de inspectores de la red de distribución eléctrica de Córdoba en el aumento de la eficiencia de sus operaciones para reducir las pérdidas no técnicas de energía en la ciudad.

El trabajo realizado se elaboró con Matlab, elaborando códigos de lectura, preprocesamiento y posprocesamiento de los datos, utilizando los algoritmos de clasificación ya implementados con algunas variantes. Al finalizar el análisis elaboramos un sólo código que unifica todos los pasos de la elaboración del clasificador. Explicamos el trabajo realizado en orden cronológico.

Con respecto a las modificaciones en los códigos nos referimos al uso del comando *svmtrain*. Este comando por defecto selecciona el método SMO, y el método que preferimos usar es el QP por aprovechar la estructura del problema y no realizar elecciones eurísticas en el proceso. Al llamar *quadprog*, éste tiene por defecto la opción *active–set*, y decidimos modificar el código de *svmtrain* para modificar el llamado de *quadprog* y elegir la opción *interior–point–convex*, el cual según nuestros experimentos y la misma literatura de matlab, se recomienda usar, ya que el problema de SVM, en la mayoría de los núcleos, es un problema convexo.

> *"En limitados experimentos, el algoritmo 'interior–point–convex' fue la mejor opción de 'quadprog' para 'svmtrain' tanto en velocidad como uso de la memoria"*. [16]

Además, como en los primeros experimentos nos encontramos con no poder resolver el problema de optimización por llegar al número máximo de iteraciones decidimos aumentar el máximo de iteraciones de 200 a 400. Modificamos entonces *svmtrain* y generamos una nueva función *svmtrainmodificado*. Se reemplazó la línea 342 de la siguiente manera:

```
qp_opts = optimset('Algorithm','active-set','display','off');
```

por

```
qp_opts =
    optimset('Algorithm','interior-point-convex','display','off','MaxIter',400);
```

En un principio intentamos trabajar con las cinco opciones de núcleos que nos ofrece Matlab: lineal, cuadrático, polinomial, RBF y MLP. Pero, con éste último, no pudimos resolver el problema con el método seleccionado ya que se genera un problema que no es convexo. Para intentar resolver esto volvimos al método *active–set*, pero el problema no convergía aún con el margen de 400 iteraciones. Intentamos entonces aumentar el máximo hasta 1000 iteraciones, y seguía sin converger, y es por ello que terminamos descartando esta opción de núcleo. Con el resto, pudimos trabajar de la manera mencionada.

4.1. Tratamiento de los datos

Como se mencionó anteriormente, contamos con 106551 datos de 9330 clientes. Estos datos fueron otorgados por la empresa encargada de la distribución de energía eléctrica de la ciudad de Córdoba al Dr. Fernández, y luego de un pretratamiento, lo recibimos en formato *csv*. Este archivo cuenta con 6 columnas, a saber:

- el número de consumidor,
- la fecha de control,
- el consumo marcado,
- un número que hace referencia al tipo de propiedad,
- una S o N que hace referencia si se registró o no una conexión ilícita, y
- un número de ruta.

Leemos este archivo con el comando *textscan* y a la quinta columna que no es numérica la trabajamos con el comando *cell2mat*.

Podemos notar que no tenemos la misma cantidad de datos para cada consumidor. Como necesitamos entrenar al clasificador con vectores de igual longitud analizamos primero cuál es la cantidad de datos que más se repite, con el fin de utilizar al subconjunto de consumidores que cuente con ésta cantidad para elaborar el clasificador. Para ello, primero abrimos el archivo con el comando *fopen*, lo leemos y luego los almacenamos en una *cell* con *textscan*.

Primero analizamos el número de clientes tomando la primer columna, la cual contiene los números identificadores de los clientes tantas veces como cantidad de datos tenga el cliente, y utilizando el comando *unique*, el cuál nos devuelve un nuevo vector pero que contiene sólo una vez cada número, así la longitud del vector equivale a la cantidad de consumidores, y este dato lo obtuvimos con el comando *length*.

Luego, con el objetivo de saber cuántos datos tiene cada cliente, realizamos un ciclo que involucra el comando *sum* y colocamos en el casillero del cliente k la de tantos 1's como veces aparece el número de cliente en los datos originales.

Posteriormente, realizando un histograma, se concluye que más de 5800 usuarios tienen exactamente 12 datos, siendo éste entonces el número que fijamos. Así trataremos de entrenar el clasificador con un subconjunto de clientes que tengan exactamente 12 datos. A continuación, el primer código y el histograma resultante.

```
% limpiar
clear all;
close all;

% leer archivo de datos
fileID=fopen('concatenados2.csv');
C=textscan(fileID,'%f %s %f %f %s %f','Delimiter',',');
fclose(fileID);

% extraer datos importantes
nl=length(C{1});
clientes=unique(C{1});
nroclientes=length(clientes)
lecturasxcliente=zeros(nroclientes,1);
for k=1:nroclientes
```

```
    lecturasxcliente(k)=sum(C{1}==clientes(k));
end

figure(1);
hist(lecturasxcliente,25);
title('histograma de cantidad de lecturas por cliente');
grid on;
```

En la Figura 11 puede verse un histograma que muestra la cantidad de usuarios de acuerdo a la cantidad de lecturas:

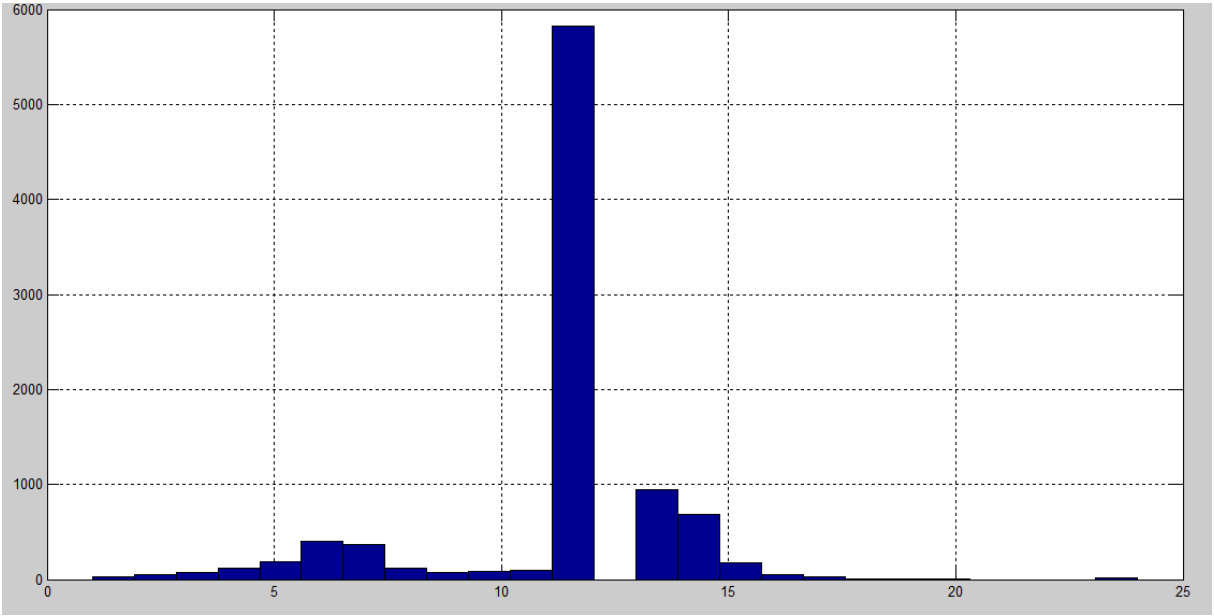


Figura 11: Histograma de cantidad de lecturas de datos por cliente.

Para corroborar esto, realizamos lo siguiente:

```
>> sum(lecturasxcliente==12)
```

lo cual nos dió como resultado 5823. Veamos cuántos de estos clientes realizaron alguna conexión ilícita de electricidad, y cuántas veces, de los 12 controles, fueron registrados como que sí robaban. Para esto realizamos:

```
fraudesalgunavez=cell2mat(C{5});
fraudes(k)=sum(C{1}==clientes( k ) & fraudesalgunavez == 'S');
```

Así, en el lugar k, tenemos cuántos controles resultaron clasificados con S. Es de interés chequear que el total de los controles correspondan a una misma variable, N o S, así esta variable no depende del control del mes, sino, del consumidor, y podemos entrenar nuestro clasificador. Para corroborar esto realizamos la siguiente cuenta:

```
>> fraudes./lecturasxcliente
```

Esto nos devuelve 0 si no se registró ninguna conexión ilícita, 1 si todos los controles fueron registrados con S, y un número entre 0 y 1 en caso contrario. Luego, contemos los casos no deseables:

```
>> sum( fraudes./lecturasxcliente > 0 & fraudes./lecturasxcliente < 1 )
```

Esto nos dio como resultado 0. Por lo tanto, ya tenemos que cada consumidor está asociado a una sola variable, S o N, según corresponda. Luego, ¿Cuántos de los 5823 clientes con 12 datos han robado? ¿Esta relación es coherente a la que se da en todos los datos, para poder quedarnos con este subconjunto sin obtener algo tan desviado? Esta pregunta la respondimos de la siguiente manera:

```
>> sum( lecturasxcliente == 12 & fraudes./lecturasxcliente > 0 )
ans =
     331

>> 331/5823
ans =
     0.0568

>> sum( fraudes./lecturasxcliente > 0 )
ans =
     701

>> 701/9330
ans =
     0.0751
```

Es decir, en el total de los datos tenemos que aproximadamente 7 % de los consumidores tienen conecciones ilícitas, y en nuestro subconjunto de clientes con 12 datos cometieron fraude aproximadamente el 5 %. Si bien no es exactamente la proporción, si tomamos los datos totales como la aproximación a la realidad, podemos asumir que nuestro subconjunto de entrenamiento no se aleja de la realidad,y que el clasificador nos dará resultados razonables.

Queremos armar una matriz A de dimensión 5823 tal que en cada fila se encuentre los datos de cada consumidor, teniendo en la primer columna el número de consumidor, en la segunda el número que indica el tipo de propiedad, y en las siguientes 12 los datos. Luego, X un vector de dimensión 5823×1 con las respectivas etiquetas. Para esto escribimos las siguientes líneas, en las que a partir de recorrer el vector *lecturasxcliente*, nos detenemos en los que tengan 12 lecturas y guardamos su nombre y sus consumos en la matriz A. Si robó alguna vez (que como vimos, equivale a haber robado en los 12 consumos), colocamos la etiqueta 1, y si no se registraron pérdidas no técnicas, colocamos la etiqueta -1:

```
n = sum( lecturasxcliente == 12 ) ;
```

```
% Quiero matriz A de entrenamiento, y etiquetas X.
A = zeros( n , 14 ) ;
X = zeros( n ,1 ) ;

consumos = C{ 3 } ;
nrotipodecliente = C{ 4 } ;
tipodecliente = unique( nrotipodecliente ) ;

m = 1 ;
for k = 1 : nroclientes
    if lecturasxcliente( k ) == 12
       A( m , 1 ) = clientes( k ) ;
       A( m , 2 ) = unique( nrotipodecliente( C{ 1 } == clientes( k ) ) ) ;
       A( m , 3 : 14 ) = consumos( C{ 1 } == clientes( k ) ) ;
       if robos( k ) > 0
          X( m ) = 1 ;
       else
          X( m ) = 0 ;
       end
       m = m + 1 ;
    else
       k = k + 1 ;
    end
end
```

4.2. Primera clasificación

Teniendo así la matriz con los consumos y el vector de las etiquetas, buscamos entrenar nuestro clasificador utilizando el comando de Matlab *svmclassify* y el comando modificado *svmtrainmodificado* como explicamos al comienzo de este capítulo.

Decidimos dividir nuestro conjunto de 5823 datos en dos, y entrenamos a lo largo de nuestro trabajo con los primeros 2700 consumidores con 12 datos y testeamos con los restantes. Antes, corroboramos que la proporción de etiquetas de cada clase se asemeje a la que se encuentra en el total del conjunto de consumidores con 12 datos. Para esto realizamos,

```
>> length( X )
ans =
     5823

>> sum( X ) / length( X )
ans =
     0.0568

>> sum( X ( 1 : 2700 ) ) / 2700
```

```
ans =
    0.0611
```

La diferencia es de 0.0043, menos del 1 %, por lo que comenzamos a trabajar con este subconjunto.

Entrenamos entonces nuestro clasificador con los primeros 2700 datos, con las opciones de funciones de núcleos que nos ofrece *svmtrain*, éstas son *linear*, *quadratic*, *polynomial*, *rbf* y *mlp* para hacer referencia al núcleo lineal, cuadrático, polinomial, de base gausiana radial y perceptron multicapa. El modo de usar estos comandos es el siguiente:

```
svmStruct = svmtrainmodificado( A( 1 : 2700 , 2 : 13 ) , X( 1 : 2700 ) ,
    'kernel_function' , 'linear' , 'method' , 'QP' ) ;

clasif = svmclassify( svmStruct , A( 2701 : end , 2 : 13 ) ) ;
```

En la primer línea se entrena el clasificador con los primeros 2700 datos, en la segunda se prueba el clasificador con los restantes 3123 consumidores que no se usaron, que se encuentran desde la fila 2701 hasta el final de la matriz A.

Analizaremos la eficacia del clasificador comparando las etiquedas obtenidas con las que tienen los clientes en realidad. Analizaremos algunos ítems que comenzaremos enumerando desde (IV) ya que en secciones siguientes utilizaremos los mismos ítems pero con tres más previos. Explicamos cada uno junto con los resultados obtenidos con el núcleo lineal para que la explicación sea más clara.

(IV) Fraudes según el clasificador: una manera de corroborar en primera instancia al clasificador será comparando la cantidad consumidores del conjunto de prueba que tienen etiqueta 1, con la cantidad de consumidores de este mismo conjunto pero que el clasificador le asigna etiqueta 1. Se puede ver que la cantidad de consumidores fraudulentos del conjunto de prueba es 166. Recordemos que este conjunto fue controlado ya por un inspector de la empresa que le asigno dicho rótulo. Si denominamos *prueba* al vector que contiene las etiquetas del conjunto de prueba, este cálculo es simplemente

```
>> sum(prueba)
```

Veamos ahora cuántos consumidores han sido etiquetados como fraudulentos por el clasificador, lo cual es equivalente a ver cuántos 1's hay en el vector *clasif*, vector que otorga la salida de *svmclassify*. A este cálculo lo denominamos "Fraudes según el clasificador" y lo utilizaremos también más adelante, en donde lo denotaremos con el número (IV). El resultado ideal de esta cuenta es 166 y se realiza

```
>> sum(clasif) =
ans =
    1202
```

y en este caso nos da como resultado 1202, lo que equivale a decir que el clasificador ha detectado de los 3123 consumidores del conjunto prueba a 1202 como sospechosos.

sensitivity or true positive rate (TPR)

(v) Porcentaje de fraudes acertados según el clasificador (o sensibilidad): veamos ahora qué porcentaje de los consumidores fraudulentos fueron detectados, ya que en esto se basa el objetivo final. Se lo suele denominar también como taza de verdaderos positivos. En general se calcula:

$$\frac{\sum verdaderos\ positivos}{\sum positivos}$$

Para el trabajo lo denotaremos con el número (V) y lo calculamos de la siguiente manera:

```
>> sum( ( clasif == prueba ) - ( ( clasif + prueba ) == zeros( n - 2700 , 1 )
   ) ) ) / sum( prueba )
```

ya que la idea es contar sobre el total de etiquetas 1 de los que tengo de prueba, a cuántos el clasificador le dio efectivamente la etiqueta 1. Para esto, de todas las compatibilidades entre los vectores restamos cuando ambos son ceros, lo que equivale a ver cuándo la suma de ambos vectores es cero. Para ver el porcentaje sobre el total de consumidores prueba fraudulentos, dividimos por la cantidad de ellos, que como ya dijimos, equivale a *sum(prueba)*. El valor ideal de esta cuenta es 1, que equivale a haber encontrado el 100 % de los consumidores que generan pérdidas no técnicas.

En los trabajos de Nagi este índice es multiplicado por 100 y se denomina tasa de éxito.

En nuestro caso esta cuenta da 0.6807229, que equivale a haber encontrado el 68 % de los consumidores que cometen fraude.

(vi) Exactitud(o precisión): en general, es el porcentaje total de aciertos. Se calcula:

$$\frac{\sum verdaderos\ positivos + \sum verdaderos\ negativos}{\sum Total\ de\ la\ población} = \frac{TP + TF}{P + N}.$$

Por ende, va a ser el porcentaje de aciertos tanto en los fraudes como en los no fraudulentos que tiene el clasificador. El resultado ideal es 1 y la cuenta es similar a la anterior, con la diferencia que da un porcentaje sobre el total del conjunto de prueba que en nuestro caso es 3123. Queda de la siguiente manera:

```
>> ( sum( clasif == prueba ) ) / length( prueba )
```

En nuestro caso este cálculo da como resultado 0.634360, lo que significa que el 63 % de las etiquetas del clasificador fueron asignadas de manera correcta.

En la Tabla 1 se muestran los resultados del clasificador con respecto a cada núcleo.

Para tratar de mejorar estos resultados vamos a agregarle características a los datos, a las que llamamos índices, y tienen como objetivo ayudar a separar mejor los datos. La idea es que estos índices aproximen a las etiquetas, para que el clasificador mejore. Pero antes nos preguntamos si esto tiene esto sentido. Si agregáramos una característica adicional a los datos podría ocurrir que los datos queden separados por un hiperplano. Esto equivale a implementar un criterio de clasificación.

Kernel	(IV)	(V)	(VI)
linear	1202	0.0940	0.6344
quadratic	701	0.1227	0.7775
polynomial	622	0.1206	0.7957
rbf	692	0.1069	0.7727

Tabla 1: Primeros resultados.

4.3. Extracción de características 1: agregando índices

4.3.1. Análisis del anexo de índices

Agregaremos una columna adicional de etiquetas, simulando que encontramos un índice que aproxima la etiqueta a un 100 % con la esperanza de ayudar al clasificador a separar los datos. El código que realiza esta parte se llama *aproximaretiqueta1*. Las líneas del mismo que se encargan de agregar esta columna de datos y entrenar nuevamente son las siguientes:

```
% Agregamos un índice
n = sum( lecturasxcliente == 12 ) ;
A = horzcat( A , X ) ;

% Clasificador
svmStruct = svmtrainmodificado(A( 1 : 2700 , 3 : 15 ) , X( 1 : 2700 ) ,
    'kernel_function' , 'mlp' , 'method' , 'QP' ) ;
clasif = svmclassify( svmStruct ,A( 2701 : end , 3 : 15 ) ) ;
```

Ejecutamos este código con todas las opciones de núcleos que nos ofrece Matlab, y en base a esto completamos la Tabla 2 con los ítems explicados antes y que nos permiten analizar la actuación del clasificador:

Kernel	(IV)	(V)	(VI)
linear	166	1.0000	1.0000
quadratic	167	0.9940	0.9997
polynomial	164	0.9880	0.9994
rbf	145	0.8735	0.9933

Tabla 2: Aproximación de una etiqueta.

Esto nos dice que si logramos realizar un índice que aproxime al 100 % las etiquetas, nuestro

clasificador funcionaria de manera exacta con el núcleo lineal, y casi perfectamente con los otros núcleos. Esto nos motiva a buscar un índice que aproxime de la mejor manera a las etiquetas, y este trabajo se realizará en las subsecciones siguiente.

Ahora simulamos que aproximamos dos índices con exactitud 100 % y los resultados se muestran en la Tabla 3.

Kernel	(IV)	(V)	(VI)
linear	166	1.0000	1.0000
quadratic	167	1.0000	0.9997
polynomial	164	0.9880	0.9994
rbf	143	0.8614	0.9926

Tabla 3: Aproximación de dos etiquetas.

Como podemos observar, los resultados son casi idénticos o un poco peor que con una etiqueta, y es por eso que en el trabajo agregaremos sólo una, y buscaremos la mejor entre tres opciones.

En las próximas subsecciones explicaremos el marco de estudio del análisis de los índices, y analizaremos con detalle tres casos distintos, los compararemos y elegiremos uno para unirlo a nuestra base de datos y utilizarlo en los siguientes pasos.

4.3.2. Propuesta de marco de estudio – Criterios de selección

Basados en una de las ideas descriptas en [18] y [19], y anexando algunos experimentos propios, estudiamos un método automático de extracción de características de los datos históricos de consumo con una combinación de SVMs para identificar clientes fraudulentos. Estas características fueron cuantificadas en diferentes índices. Dentro de estos índices, estudiamos también el uso de la información de los perfiles de consumo de los clientes para exponer comportamientos anormales, pues se sabe que tienen gran relación con las actividades de pérdidas no técnicas. El objetivo final es generar una lista de potenciales sospechosos de fraude en el sitio de inspección basados en comportamientos significativos que surgen debido a irregularidades en el consumo.

En las siguientes secciones expondremos la idea de la elaboración de cada índice, los códigos con los que se ejecutaron, algunas imágenes representativas, resultados al implementarlos con cada núcleo. Luego compararemos el clasificador sin índices y agregando cada uno de éstos, centrándonos en un objetivo: disminuir la cantidad de usuarios sospechosos pero tratando de incluir a la mayor parte de los 166. A diferencia de los artículos mencionados, hemos buscado llevar las características a resultados determinísticos, dando 1 o 0, según dicha característica indique que el consumidor se asemeja más a un consumidor típico fraudulento o no, bajo la idea de la subsección anterior que una buena aproximación de las etiquetas brinda como resultado un buen clasificador.

Para cada índice vamos a notar el resultado de algunas cuentas que nos ayudarán a discernir cómo mejora o no al clasificador y luego poder elegir el o los mejores. Las cuentas están hechas en base a calcular los índices de manera independiente y guardar la información en la columna 15 de la matriz A. Se calculan tres ítems y luego se elabora el clasificador, ya que aunque el índice no haya aproximado de la mejor manera a las etiquetas, puede influir de manera positiva en el resultado final. Se entrena entonces al clasificador con 13 datos: los 12 de los consumos más el nuevo índice. Se analizan los resultados, calculando items propios y los dos utilizados en los dos artículos mencionados.

(I) Fraudes según el índice: es similar a Fraudes según el clasificador descripta anteriormente en el ítem (IV). La idea es que cada índice se asemeje a las etiquetas, y al clasificar un nuevo vector como no tenemos la etiqueta, realizamos una primera clasificación y ayudamos así a que la aproximación final sea más acertada. Pero, como nuestro vector X es de etiquetas, veamos si realmente las estamos aproximando. En teoría cuando ambos deberían ser 1 y 0 en los mismos lugares, por ende, la suma de estas "nuevas etiquetas" debería ser 331, ya que vemos qué tan bien aproxima tanto para el conjunto de entrenamiento como para el conjunto de prueba. Este cálculo lo realizamos de la siguiente manera:

```
>> sum( A( : , 15 ) )
```

(II) Etiquetas mal aproximadas por el índice: bajo el mismo criterio del ítem (I), la suma de los valores absolutos de los elementos de la diferencia entre los valores generados por el índice y el vector de etiquetas X debería ser cero. Esta cuenta es:

```
>> sum( abs( A( : , 15 ) - X ) )
```

(III) Porcentaje de fraudes acertados según el índice: otra medida que utilizamos para comparar los índices es el porcentaje de fraudes acertados, ya que si bien nos interesa cuán bien aproxima las etiquetas en general, el foco está en no dejar afuera a los consumidores fraudulentos. Esto lo calculamos de la siguiente manera:

```
>> ( sum( ( ( A( : , 15 ) == X ) - ( ( A( : , 15 ) + X ) == zeros( n , 1 ) ) )
   ) ) / sum( X )
```

Similar al punto (V) descripto anteriormente como "Porcentaje de fraudes según el clasificador". El valor ideal de esta cuenta es 1.

El resultado de todos estos ítems se imprime en pantalla al correr cada uno de los códigos que se encargan de calcular los índices y entrenar luego el clasificador con dicho índice.

Un ítem que tuvimos en cuenta, aunque no lo calculamos explícitamente en todos los casos, fue la cantidad de fraudes acertados, que no es más que el ítem (IV) por el (V), pero utilizaremos varias veces esta cantidad para comparar las listas de sospechosos propuesta por el clasificador.

Vamos a fijar criterios para determinar cuál clasificador nos conviene más. Por un lado, quisiéramos encontrar un listado de 166 personas sospechosas y que sean las correctas. Para

esto buscaremos disminuir la cantidad de usuarios sospechosos pero tratando de incluir a la mayor parte de los 166. Dados dos clasificadores que devuelven dos listados distintos, ¿cuál escogeremos?. El clasificador 1 da un listado de a sospechosos de los cuales b son ciertos (el índice de fraudes acertados sería b/a), el clasificador 2 da un listado de c sospechosos de los cuales d son acertados (el índice de fraudes acertados sería d/c), y supongamos que c es más grande que a. Podemos tener dos casos:

- $d \leq b$: en este caso claramente nos quedaremos con el primer clasificador, ya que aumentamos la cantidad de sospechosos sin ningún beneficio extra, ya que los aciertos no mejoran o empeoran.

- $d > b$: en este caso puede o no convenir. Decidimos que si la cantidad de sospechosos nuevos trae consigo una cantidad de aciertos nuevos mayor que los que podría haber obtenido si los elegíamos de manera aleatoria, entonces hay chances de que el segundo clasificador sea mejor. Es decir, si $(d-b)\,/\,(c-a) > 166/3123 = 0.0532$, entonces, nos quedamos con el segundo clasificador. Aunque, a veces la mejora no es muy "significativa", para esto utilizaremos también el criterio de Nagi.

El criterio utilizado por Nagi, que es buscar una tasa de éxito y exactitud alta, que en nuestro caso son los índices (V) y (VI). El resultado ideal es tener ambos índices iguales a 1. Y, en el caso que obtuviéramos un resultado ideal, vamos a haber acertado el 100 % de las etiquetas, y por ende, obtendremos un listado de 166 personas sospechosas y acertadas, y cumpliríamos nuestro objetivo.

Si tenemos dos clasificadores y uno tiene ambos índices más altos que los del otro, elegiremos el primero. Si un número es más alto pero el otro es más bajo, descartaremos uno utilizando el criterio anterior. Decidimos un orden para utilizar los criterios ya que encontramos casos en los que ante una misma decisión, los criterios toman elecciones distintas. De todas formas los criterios no son tan opuestos, ya que se puede ver que si dos clasificadores tienen un índice igual, y el otro distinto, el segundo criterio diría que hay que elegir el que gane en el índice distinto, y haciendo cuentas, se logra ver que el primer criterio también elegiría el mismo.

Basado en este esquema de decisión, que utilizaremos tanto en esta sección como en las siguientes, continuamos con la presentación de tres índices distinto y el cálculo de los ítems mencionados, posteriormente elegiremos el mejor.

4.3.3. Proximidad al perfil promedio

El primer índice propuesto se relaciona con la proximidad del perfil del consumidor con el perfil de un usuario fraudulento promedio y del no fraudulento.

Denominamos perfil al conjunto ordenado de los consumos bimestrales consecutivos en un determinado periodo de tiempo. En nuestros datos, el periodo de tiempo es el de los años 2011 y 2012.

El perfil promedio de cada clase estará dado por el promedio de los perfiles de los consumidores de cada clase. Así, basados en los consumidores con 12 datos, para la clase j el perfil de dicha clase será un vector fila de longitud 12 donde en el lugar i irá el promedio de los elementos de la columna $i+2$ de la matriz A que pertenezcan a consumidores de la clase j (recordemos

que en la primer columna tenemos el número de usuario, y en la segunda el tipo de usario). Es decir, de la columna $i+2$ tomaremos el de la fila k sólo si la etiqueta del consumidor k es j, que es equivalente a que $X(k) = j$, donde el vector X es el de las etiquetas. Para realizar el promedio, notemos que la cantidad de clientes fraudulentos coincide con $sum(X)$, y la cantidad de clientes no fraudulentos con $n - sum(X)$. El cálculo de estos perfiles promedio lo realizamos dentro del código *leerdatos*, y las líneas que lo calculan son las siguientes:

```
perfilroban = zeros( 1 , 12 ) ;
for i = 1 : 12
    Ai = A( : , i + 2 ) ;
    perfilroban( i ) = ( sum( Ai( X == 1 ) ) ) / sum( X ) ;
end

perfilnoroban = zeros( 1 , 12 ) ;
for i = 1 : 12
    Ai = A( : , i + 2 ) ;
    perfilnoroban( i ) = ( sum( Ai( X == 0 ) ) ) / ( n - sum( X ) ) ;
end
```

Graficamos ambos perfiles utilizando el comando *plot*, enfrentando el vector $Z = [1:12]$ con cada perfil, y obtenemos la Figura 12.

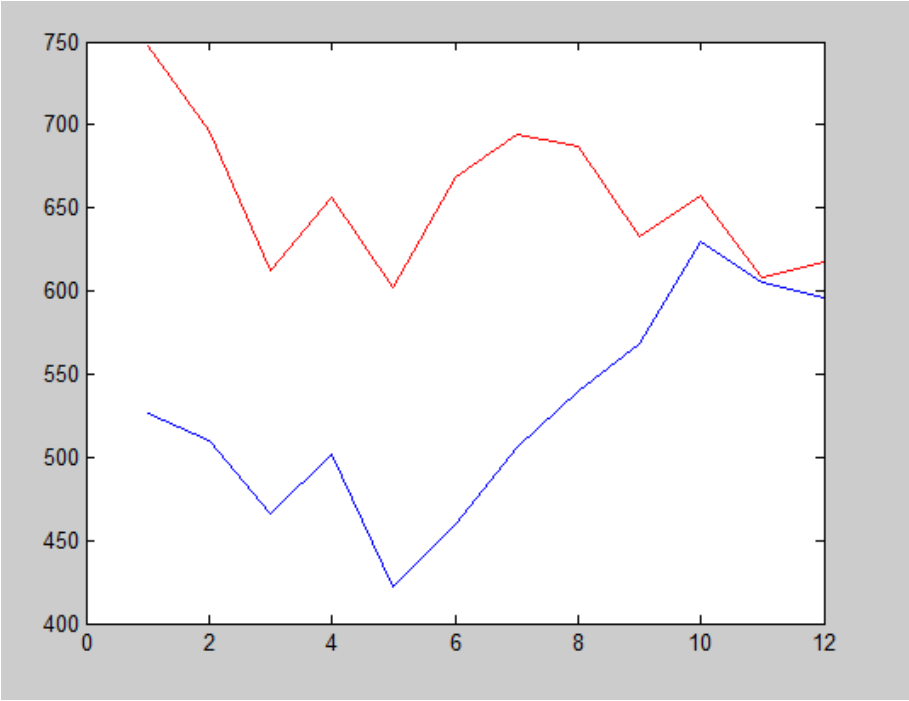


Figura 12: Rojo: perfil de la clase 0 (consumidor estándar). Azul: perfil de la clase 1 (consumidor fraudulento).

Para ver que la estructura de los perfiles promedios se conserva como estructura general de los datos, tomamos alternadamente los consumidores, realizamos en base a ellos los perfiles promedio, y vemos así que tomar la estructura del promedio va a ser tomar información que realmente represente a los datos de manera general. Graficamos los tres perfiles calculados juntos para ver visualmente esto (ver Figura 13).

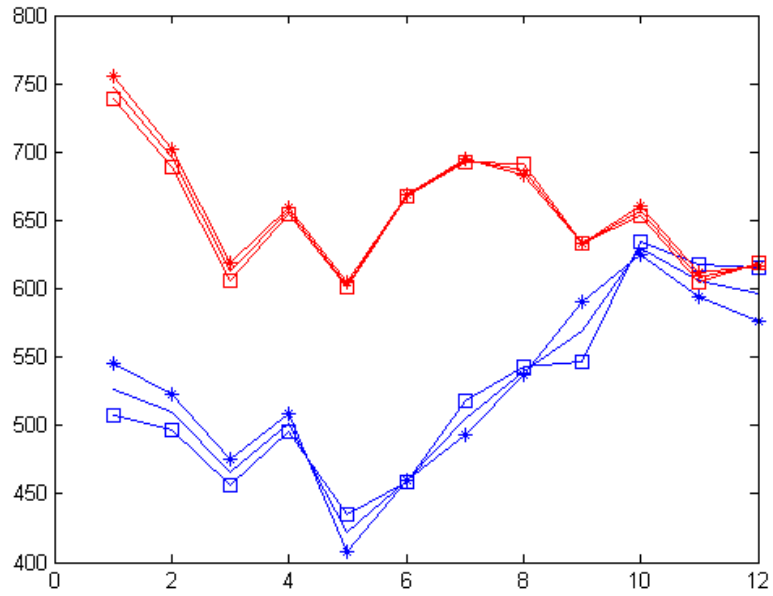


Figura 13: Perfiles promedio de todos los consumidores, de los pares, y de los impares. Rojo: perfil de la clase 0 (consumidor estándar). Azul: perfil de la clase 1 (consumidor fraudulento).

Calculamos para cada consumidor la distancia de su perfil con los perfiles promedio calculados. La distancia que utilizaremos es la euclídea, ya que nos interesa la suma de las diferencias de cada componente del perfil del usuario con el promedio, y elevamos al cuadrado ya que la diferencia positiva tendrá el mismo peso que la negativa. Podríamos considerar también el valor absoluto de estas diferencias, utilizando así la norma 1 en lugar de la 2.

Calculamos cuál es el menor valor de ambos cálculos, y agregamos una columna en la matriz A al lado de los consumos de cada consumidor. Colocamos un 1 si se realiza la menor distancia con el vector *perfilroban*, y 0 en caso contrario.

```
l = input( 'ingrese la norma que desea utilizar (1 o 2): ' ) ;

n = sum( lecturasxcliente == 12 ) ;
A = [ A , zeros( n , 1 ) ] ;
for k = 1 : n
    if norm( A( k , 3 : 14 ) - perfilroban , l ) <= norm( A( k , 3 : 14 ) -
       perfilnoroban , l )
```

```
      A( k , 14 ) = 1 ;
    else
      A( k , 14 ) = 0 ;
    end
end
```

Luego implementaremos este dato para entrenar al clasificador, entrenando ahora con 13 datos en lugar de con 12.

El cálculo de este primer índice, el anexo de esto en la matriz A y la nueva clasificación con sus respectovos resultados, se realiza en el código *conunindice*, el cual tiene la opción de ingresar ambas normas.

Los resultados de estos cálculos con este primer índice, utilizando norma 1 y 2, y con los diferentes núcleos, están expresados en las Tablas 4 y 5.

	Ideal	1° índice con $\|\cdot\|_1$	1° índice con $\|\cdot\|_2$
(I)	331	4095	3990
(II)	0	3894	3811
(III)	1	0.8036	0.7704

Tabla 4: Aproximación de las etiquetas del 1° índice.

	Con $\|\cdot\|_1$			Con $\|\cdot\|_2$		
Kernel	(IV)	(V)	(VI)	(IV)	(V)	(VI)
Ideal	166	1	1	166	1	1
linear	1133	0.0900	0.0910	1122	0.0936	0.6548
quadratic	834	0.1060	0.7362	841	0.1050	0.7339
polynomial	650	0.1060	0.7829	642	0.1140	0.7880
rbf	683	0.1020	0.7730	671	0.1040	0.7768

Tabla 5: Clasificador con el 1° índice.

Tomaremos estas tablas de resultados junto con las de las próximas subsecciones y en base a ellas elegiremos el índice a agregar en la sección 4.3.6.

4.3.4. Proximidad al perfil promedio del mismo tipo

La idea de este índice es similar a la anterior, con la diferencia que al notar que los niveles de consumo varían de manera significativa, supusimos que depende (aunque no exclusivamente) del tipo de propiedad que sea. Por ello, elaboramos nuevos perfiles promedio, pero ahora uno por cada tipo de propiead. Utilizamos aquí la información guardada en la columna 2 de la matriz A, y las cuentas que siguen son análogas al índice anterior. Estos perfiles también se encuentran en el código *leerdatos*, y las líneas que lo realizan son las siguientes:

```
perfilesroban = [ ] ;
for j = 1 : 12
    Aj = A( : , j + 2 ) ;
    for i = 1 : length( tipodecliente )
        l = Aj( X == 1 & A( : , 2 ) == tipodecliente( i ) ) ;
        perfilesroban( i , j ) = sum( l ) / length( l ) ;
    end
end

perfilesnoroban = [ ] ;
for j = 1 : 12
    Aj = A( : , j + 2 ) ;
    for i = 1 : length( tipodecliente )
        l = Aj( X == 0 & A( : , 2 ) == tipodecliente( i ) ) ;
        perfilesnoroban( i , j ) = sum( l ) / length( l ) ;
    end
end
```

Contamos con 12 tipos de propiedades, pero mostramos a modo ilustrativo los perfiles promedio de 2 de ellas en la Figura 14, para ver cómo influye este dato en los niveles de consumo de los clientes.

Como podemos ver, que si bien algunos patrones de los perfiles se respetan, en promedio los consumos de las propiedades del tipo 140 se encuentran en un rango entre 0–500 KWh bimestrales, mientras, que los del tipo 144, en un rango entre 500–1100 KWh, lo cual puede influir en que la distancia con un perfil promedio general no resulte una característica significativa para el consumidor.

Nuevamente, calculamos la distancia del perfil de cada consumidor con el promedio, pero ahora con el de su mismo tipo. Realizamos esto con la norma 1 y 2, y con todos los núcleos. El código que realiza esto se denomina *conunindice2*. Volcamos la información obtenida en las Tablas 6 y 7.

De la misma forma que el índice anterior, tomaremos en cuenta los datos de estas tablas para elegir el mejor índice en la sección 4.3.6.

4.3.5. Cambios notables

Para elaborar este índice tratamos de buscar alguna regularidad en los perfiles de consumidores fraudulentos que lo lleven a diferenciarse de los otros. Idea similar a la desarrollada por

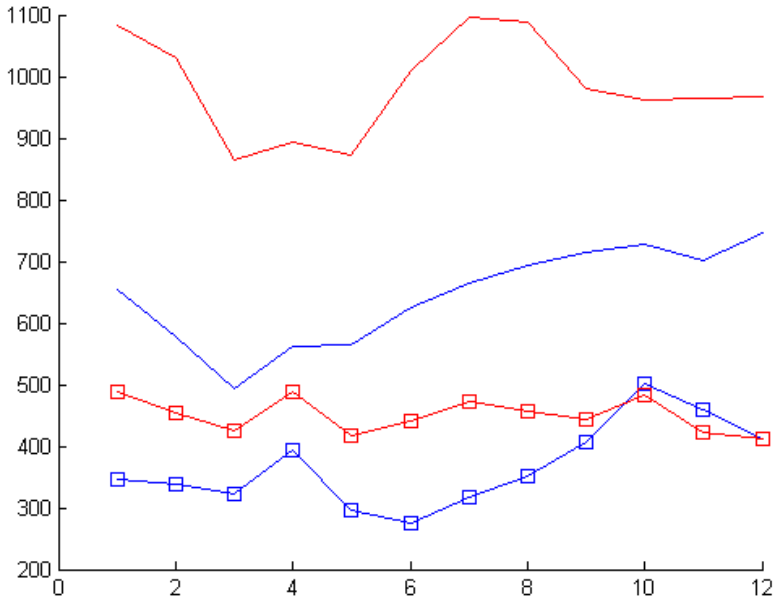


Figura 14: Perfiles promedio de dos tipos de propiedades distintas. Líneas: perfil promedio de la propiedad tipo 144. Líneas y cuadrados: perfil promedio de la propiedad tipo 140. Rojo: consumidor estándar. Azul: consumidor fraudulento.

Nagi, cuando clasifica en cuatro clases y llama a la clase 1 como "fraudes sospechosos confirmados" e incluye en este grupo a los perfiles en los cuales aparecen claramente cambios abruptos. Si bien da una idea gráfica de esto, no se menciona cÓmo cuantifica estos cambios abruptos. En los dos artículos estudiados presenta los siguientes ejemplos (ver Figuras 15 y 16).

Las imágenes de la izquierda son extraídas de [18] y las de la derecha son extraídas de [19].

En base a esta idea, analizamos nuestros perfiles para poder destacar alguna característica y cuantificarla para elaborar un nuevo índice.

Para poder visualizar los perfiles en general y extraer ideas realizamos dos videos experimentales en el que se muestran de manera secuencial los gráficos de los perfiles fraudulentos por un lado, y no fraudulentos por otro. Tratamos de ver de manera superficial si estos cambios abruptos de Nagi son realmente una característica presente en los perfiles de consumidores fraudulentos, y ausentes en los consumidores estándares, o qué otra característica diferencia a ambos conjuntos. Los códigos que realizan estas secuencias son *expperfilroba* y *expperfilnoroban*. Ponemos a continuación las líneas del primero:

```
% expperfilroban

w = 1 : 12 ;
for k = 1 : n
    if X( k ) == 1
```

	Ideal	2° índice con $\|\cdot\|_1$	2° índice con $\|\cdot\|_2$
(I)	331	2523	2594
(II)	0	2688	2743
(III)	1	0.25 08	0.2749

Tabla 6: Aproximación de las etiquetas del 2° índice.

	Con $\|\cdot\|_1$			Con $\|\cdot\|_1$		
Kernel	(IV)	(V)	(VI)	(IV)	(V)	(VI)
Ideal	166	1	1			
linear	1592	0.0770	0.5152	1562	0.0750	0.5261
quadratic	847	0.1050	0.7326	824	0.1080	0.7400
polynomial	630	0.1060	0.7880	623	0.1080	0.7903
rbf	660	0.1030	0.7791	659	0.1030	0.7794

Tabla 7: Clasificador con el 2° índice.

```
      plot( w , A( k , 2 : 13 ) , 'red' ) ;
      pause( 4 ) ;
      k = k + 1 ;
   else
      k = k + 1 ;
   end
end
```

El segundo código se diferencia al igualar el vector de las etiquetas a 0 en lugar de a 1.

Al correr los códigos y analizar las imágenes se pueden deducir comportamientos típicos de ambas clases y a raíz de esto distinguimos cuatro subconjuntos de consumidores, dos en cada clase: los que responden a estos comportamientos "típicos" y los que no. Relacionamos de manera directa estos ejemplos con las cuatro clases planteadas por Nagi. Exponemos a continuación un comentario de cada clase, y la relación, diferencia y extensión con las clases propuestas por Nagi.

Recordemos que en [18] se clasifica en estas cuatro clases, por lo cual, si bien podemos pensarlas como subconjuntos de dos conjuntos grandes: "fraudulentos" y "no fraudulentos", su definición tiene que ver meramente con las características del perfil. En cambio, nosotros contamos en nuestro conjunto de entrenamiento con dos clases que subdividimos en 4 de manera teórica y superficial, no práctica, permitiendo así ejemplos que no respondan a una subclase

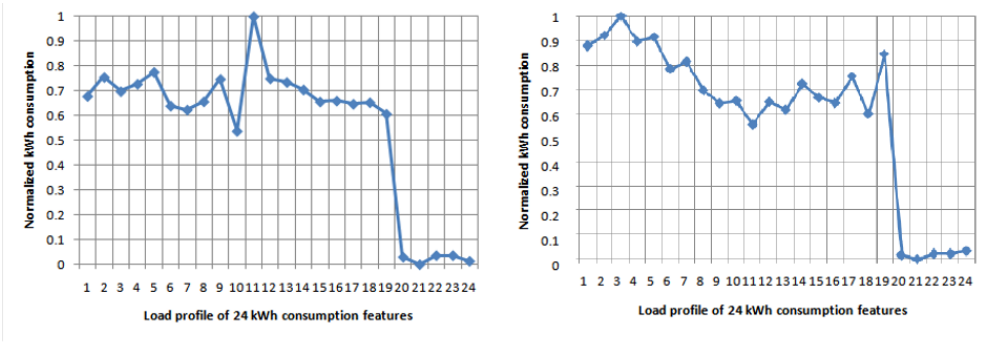


Figura 15: Perfil de consumo de un tipico consumidor fraudulento tomado sobre un periodo de dos años.

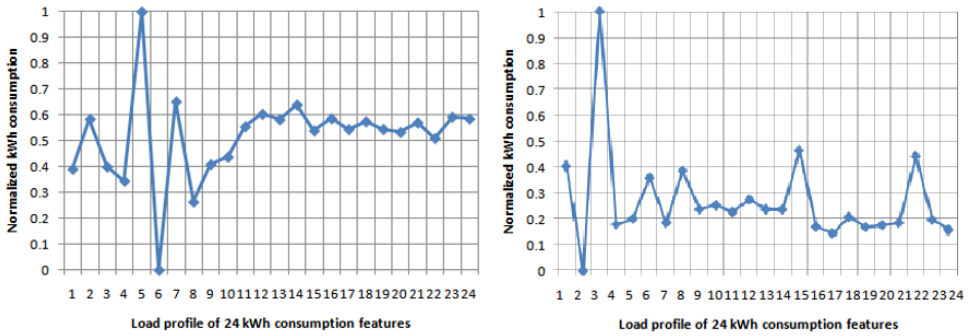


Figura 16: Perfil de consumo de un tipico consumidor no fraudulento tomado sobre un periodo de dos años.

concreta, y además, el uso de la etiqueta en las definiciones de las subclases, hecho que no estaría permitido si se deseara clasificar en estos cuatro grupos.

A continuación del comentario de cada clase expondremos algunos gráficos de perfiles de ejemplos de cada una de ellas obtenidos del código anterior. Recordemos para esto que los gráficos son las representaciones de consumos bimestrales de KWh por un cliente en un período de dos años.

- Consumidor fraudulento típico: incluimos en esta clase a aquellos clientes etiquetados como fraudulentos que presentan caídas o aumentos rotundos en sus perfiles que tienden a conservarse por un período de tiempo. Planteamos esta característica como un comportamiento en el consumo típico de un consumidor fraudulento, bajo la hipótesis de que en el mes del cambio abrupto realizaron una conexión ilegal de energía eléctrica, o bien fueron detectados como fraudulentos obligando esta situación a reconectarse al sistema eléctrico estándar. Denominamos a estos dos casos de tipo 1 y de tipo 2. Podemos encontrar también, casos de: (a) un aumento considerable del consumo seguido de un descenso también considerable casi de manera consecutiva, que viene de la mano de haber sido detectado

como fraudulento pero haber realizado luego una nueva conexión ilícita; y (b) un descenso considerable, y luego de manera consecutiva o no, un ascenso notable, que representa el haber realizado una conexión ilícita por un período acotado de tiempo incluído en el período de dos años de control. A estos comportamientos los denominaremos casos de tipo mixto. Relacionamos este subconjunto con la clase 1 planteada por Nagi como "Fraude sospechoso confirmado", en la cual incluye a los perfiles con cambios abruptos pero tiene en cuenta sólo a los de tipo 1. Ver Figura 17.

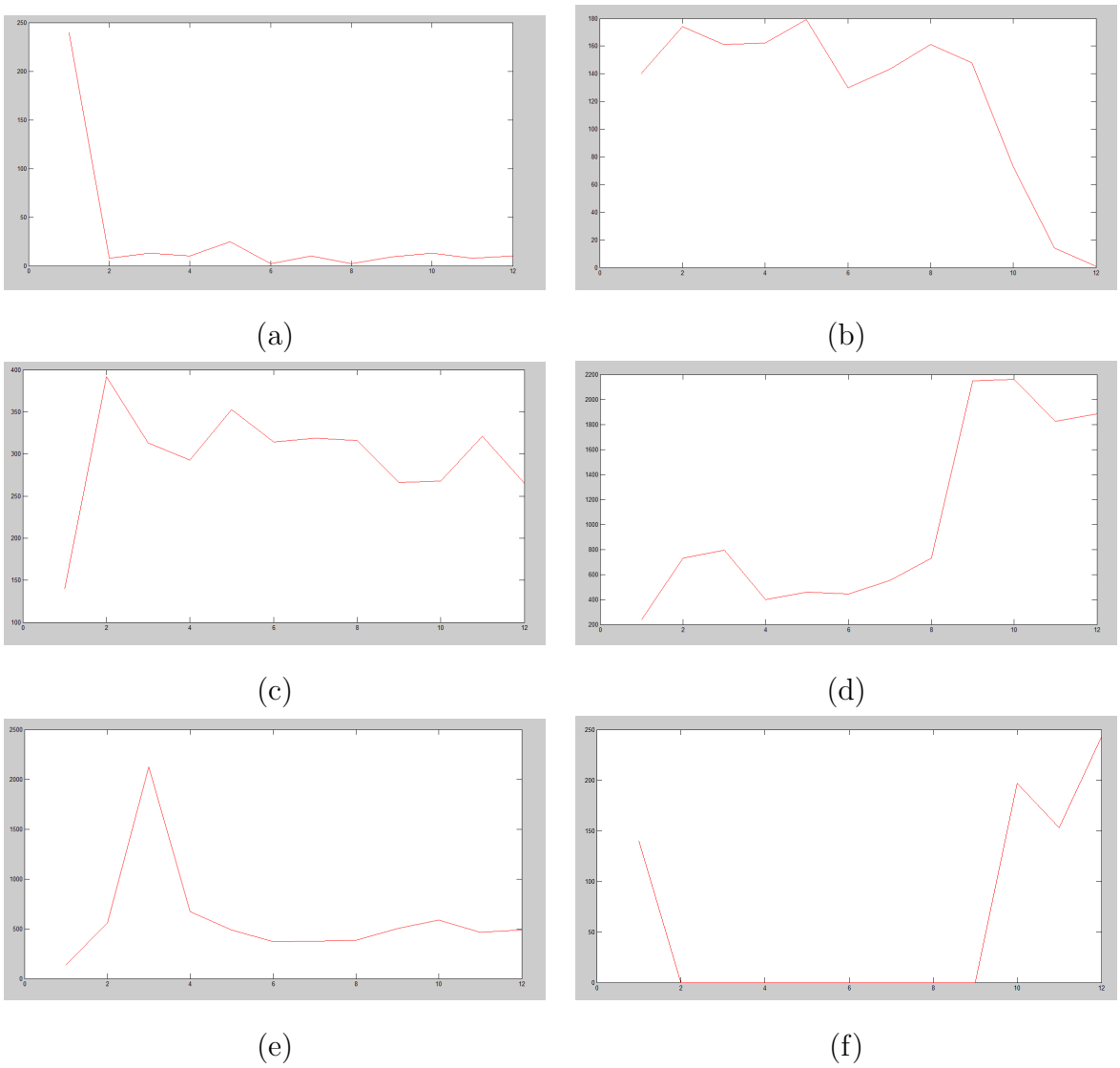
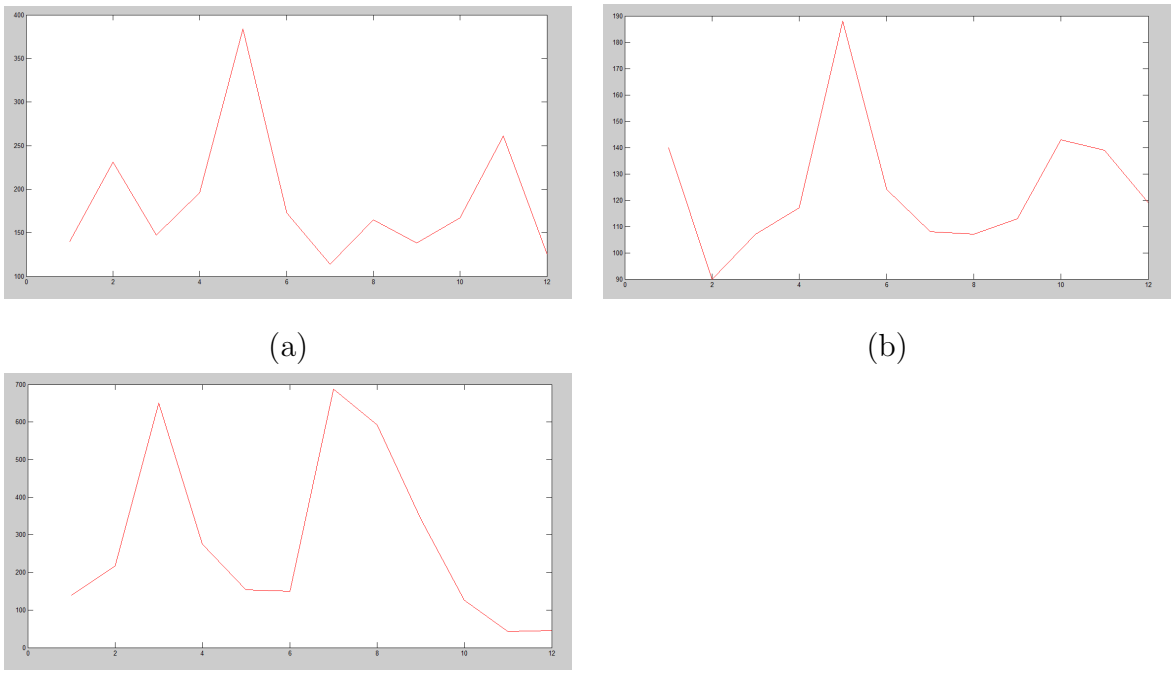


Figura 17: (a) y (b) consumidor fraudulento típico tipo 1, (c) y (d) consumidor fraudulento típico tipo 2, (e) y (f) consumidor fraudulento típico tipo mezcla.

- Consumidor fraudulento escondido: incluimos en esta clase a aquellos clientes etiquetados como fraudulentos pero que no presentan el comportamiento característico mencionado

antes, sino más bien un comportamiento similar al del no fraudulento típico, que explicaremos en el ítem siguiente, siendo una mezcla o teniendo ausente el primer comportamiento. Esto se puede relacionar con la clase 2 planteada por Nagi como "Fraude sospechoso no confirmado" en la que incluye a aquellos consumidores cuyos perfiles contienen cambios abruptos pero también presentan oscilaciones, siendo estas una característica de un consumidor no fraudulento. Una explicación hipotética para este comportamiento, también planteada por Nagi, es el haber realizado una conexión ilícita antes del periodo de tiempo en el cual tenemos los datos. Ver Figura 18.

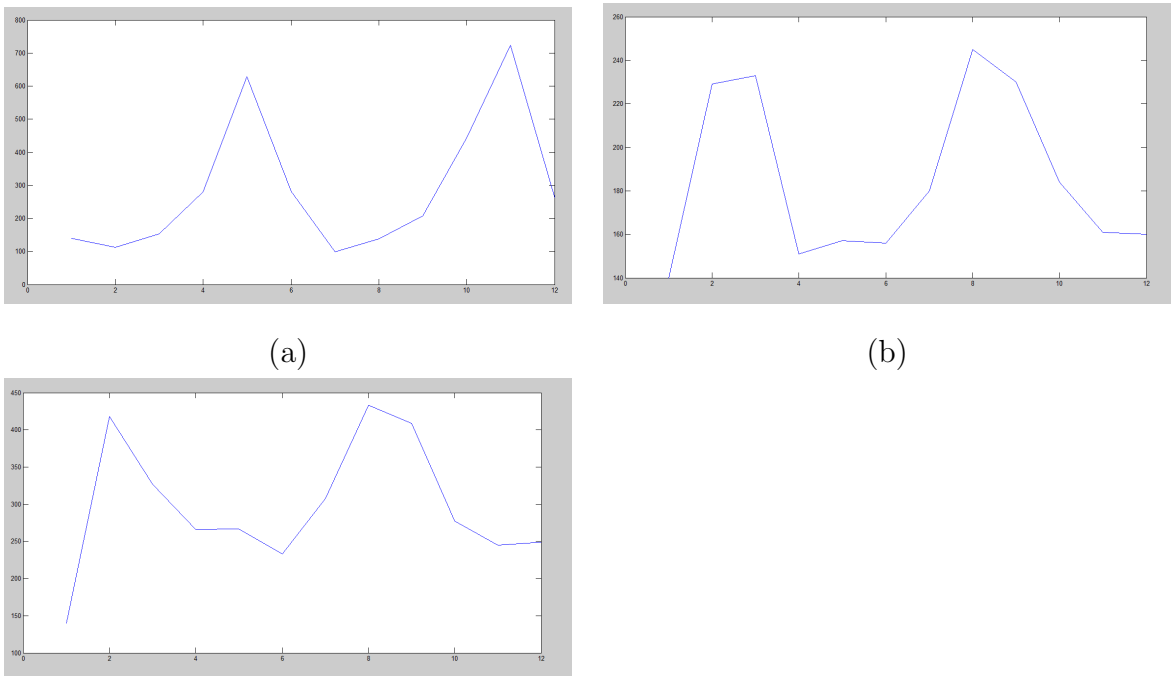


(a) (b)

(c)

Figura 18: Consumidor fraudulento escondido.

- Consumidor no fraudulento típico: incluimos en esta clase a aquellos clientes etiquetados como no fraudulentos que tienen en su perfil un comportamiento oscilatorio pero con dos picos pronunciados, los cuales distan de aproximadamente seis consumos lo que equivale a un año calendario. Planteamos esta característica como un comportamiento en el consumo típico de un consumidor no fraudulento, bajo la hipótesis de que los picos de consumo son producto del cambio de necesidades generado por el cambio de estaciones climáticas. Se relaciona con la clase 3 planteada por Nagi como "Limpio sospechoso confirmado", con la diferencia que habla simplemente de oscilaciones sin remarcar picos ya que en Malasia las temperaturas durante el año no varían mucho a lo largo del año. Ver Figura 19.

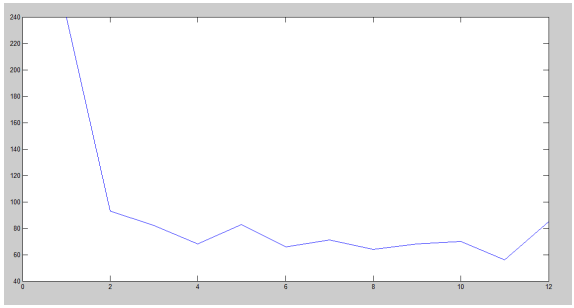


(a) (b)

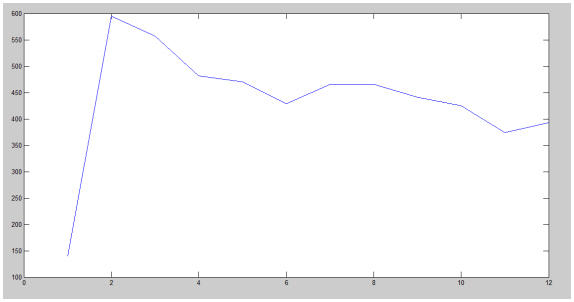
(c)

Figura 19: Consumidor no fraudulento típico.

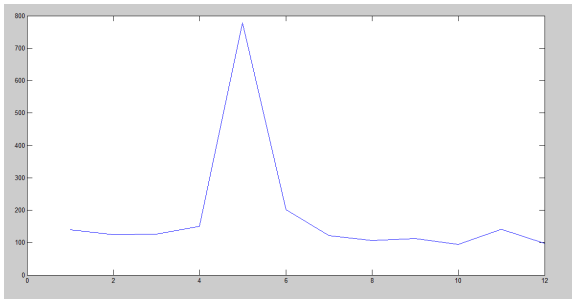
- Consumidor no fraudulento escondido: incluimos en esta clase, de manera análoga que la clase de consumidor fraudulento escondido, a aquellos clientes que poseen casi un comportamiento opuesto al esperado, presentando características típicas de la otra clase, cualquiera sea el tipo. La similitud y diferencia con la clase 4 planteada por Nagi como "Limpio sospechoso no confirmado" es idéntica a la relación explicada del segundo ítem con la clase 2. Una explicación hipotética de la presencia de cambios abruptos en consumidores no fraudulentos puede ser la incorporación de artefactos eléctricos que permanecieron constantes para el aumento rotundo, o el no uso de la propiedad por vacaciones, finalizaciones de contratos o venta. Ver Figura 20.



(a)



(b)



(c)

Figura 20: (a) consumidor no fraudulento escondido tipo 1, (b) consumidor no fraudulento escondido tipo 2, (c) consumidor no fraudulento escondido mezcla.

La idea es tratar de cuantificar estas características para poder crear un nuevo índice. Vamos a etiquetar como sospechoso de fraude a aquellos consumidores que:

- Tengan una caída abrupta que se conserve en el tiempo, donde caída le llamamos a una diferencia negativa, abrupta a que el valor absoluto de esta caída sea al menos dos veces más en relación al resto de las caídas, y que se conserven se refiere a que no vuelva a ascender, donde ascender es tener una diferencia positiva, por lo que pediremos que sea al menos el doble en valor absoluto que los ascensos. Notemos que por como lo estamos definiendo, si hay una caída que cumple estas características, esta tiene que ser única. Esto detectaría a los consumidores fraudulentos típicos de tipo 1.

- Tengan un ascenso abrupto que se conserve en el tiempo. Esto detectaría a los consumidores fraudulentos típicos de tipo 2.

- Tengan una caída y un ascenso, ambos abruptos, que se conserven en el tiempo en relación a las restantes diferencias. Hay dos órdenes posibles y dos opciones de acuerdo a si son consecutivas o no. Tenemos entonces cuatro casos de los cuales el único que no desearíamos es el tener primero un descenso y de manera continuada un ascenso, ya que sería común encontrar esto en propiedades no fraudulentas en épocas de vacaciones, pero en un principio, permitimos todos los casos. Este ítem detectaría a los consumidores de tipo mezcla.

Calculamos las diferencias entre los consumos y trabajamos con ellas. Esto tiene relación con los valores promedios tomados por Nagi en sus trabajos, con la diferencia que ellos calculan estas diferencias y las dividen por las diferencias de las fechas de dichos consumos. En nuestro trabajo, si bien tenemos las fechas, para estas cuentas tomamos que los datos fueron tomados con iguales diferencias de tiempo, y equivale a trabajar sólo con las diferencias sin dividir.

Consideramos la opción de no tener en cuenta ni la primera diferencia ni la última, ya que los cambios abruptos en los extremos suponemos que pueden llevar a confusión, porque podría ser un pico que no visualizamos. Probamos además cambiando el factor que relaciona el valor absoluto del cambio abrupto con el resto, tomamos: 2, 2.5 y 3. Ponemos a continuación las líneas de código que matematiza los ítems anteriores seguida de una tabla que vuelca los resultados encontrados para estas seis diferentes opciones recién mencionadas, con los cuatro núcleos con los que venimos trabajando: lineal, cuadrático, polinomial y RBF. Este cálculo, el clasificador con este índice y los cálculos utilizados para el análisis, se encuentran en *conunindice3*.

```
l = input( 'ingrese 1 si desea tener en cuenta los extremos, 0 sino: ' ) ;
c = input( 'ingrese el escalar para marcar caídas abruptas: ' ) ;

A = A( : , 1 : 14 ) ;
for k = 1 : n
    v = A( k , (5 - l ) : ( 13 + l ) ) - A( k , ( 4 - l ) : ( 12 + l ) ) ;
    m = max( v ) ;
    w = min( v ) ;
    vm = v ;
    vm( v == m ) = 0 ;
    vw = v ;
    vw( v == w ) = 0 ;
    if ( sum( v == m ) == 1 & abs( m ) >= c * abs( vm ) ) | ( sum( v == w ) == 1 &
        abs( w ) >= c * abs( vw ) ) ;
       A( k , 15 ) = 1 ;
       vw( v == m ) = 0 ;
    elseif sum( v == m ) == 1 & sum( v == w ) == 1 & abs( m ) >= c * abs( vw ) &
        abs( w ) >= c * abs( vw ) ;
       A( k , 15 ) = 1 ;
    else
       A( k , 15 ) = 0 ;
    end
end
```

Tenemos entonces 6 posibles clasificadores con este índice, colocamos en la Tabla 8 los resultados de los 6 posibles índices.

		$c = 2$	$c = 2.5$	$c = 3$
Sin extremos	(I)	1933	1084	657
	(II)	1992	1251	890
	(III)	0.4109	0.2477	0.1480
Con extremos	(I)	1623	844	493
	(II)	1714	1045	752
	(III)	0.3625	0.1964	0.1088

Tabla 8: ítems (I), (II) y (III) para el 3° índice con sus 6 variaciones.

A continuación presentamos las tablas correspondientes a cada uno de los núcleos, con las 6 opciones cada uno.

- Núcleo lineal: Tabla 9.

NÚCLEO LINEAL		$c = 2$	$c = 2.5$	$c = 3$
Sin extremos	(IV)	1155	1061	1194
	(V)	0.1000	0.1010	0.0950
	(VI)	0.6513	0.6756	0.6375
Con extremos	(IV)	1024	998	1182
	(V)	0.1030	0.1070	0.0950
	(VI)	0.6862	0.6958	0.6401

Tabla 9: Clasificador con el 3° índice con núcleo lineal.

- Núcleo cuadrático: Tabla 10.

NÚCLEO CUADRÁTICO		$c = 2$	$c = 2.5$	$c = 3$
Sin extremos	(IV)	689	700	699
	(V)	0.1200	0.1210	0.1200
	(VI)	0.7794	0.7771	0.7768
Con extremos	(IV)	649	667	702
	(V)	0.1190	0.1210	0.1270
	(VI)	0.7883	0.7851	0.7791

Tabla 10: Clasificador con el 3° índice con núcleo cuadrático.

- Núcleo polinomial: Tabla 11.

NÚCLEO POLINOMIAL		$c = 2$	$c = 2.5$	$c = 3$
Sin extremos	(IV)	626	587	580
	(V)	0.1170	0.1070	0.1170
	(VI)	0.7931	0.7992	0.8047
Con extremos	(IV)	626	592	615
	(V)	0.1120	0.1250	0.1200
	(VI)	0.7912	0.8047	0.7973

Tabla 11: Clasificador con el 3° índice con núcleo polinomial.

- Nucleo RBF: Tabla 12.

Núcleo RBF		$c = 2$	$c = 2.5$	$c = 3$
Sin extremos	(IV)	605	624	665
	(V)	0.1270	0.1090	0.1100
	(VI)	0.8024	0.7906	0.7807
Con extremos	(IV)	590	590	671
	(V)	0.1240	0.1170	0.1160
	(VI)	0.8047	0.8021	0.7819

Tabla 12: Clasificador con el 3° índice con núcleo RBF.

Tomamos ahora los datos de estas tablas junto con las de las anteriores para discutir, en base a los criterios establecidos, la elección de un índice en la subsección siguiente.

4.3.6. Análisis, elección y conclusión

Para elegir el mejor clasificador, tomaremos los datos de todos los cuadros, y primero analizaremos qué pasó con el núcleo RBF. Una vez elegido el mejor con éste núcleo compararemos el mejor en general, ya que, según como vimos con [9], es una buena primera elección. Entonces hay muchas posibilidades de que con el primer análisis encontremos un buen clasificador, entonces después solo resta descartar los otros. En el siguiente análisis trabajaremos a los listados como fracciones, ya que tomaremos *fraudes acertados / cantidad de sospechosos de fraudes*, que corresponde con:

$$\frac{(\text{IV}) \cdot (\text{V})}{(\text{VI})}$$

(a) El primer clasificador con el núcleo RBF da una exactitud de 0.7727, índice de fraudes acertados de 0.1070 y un listado de 74/692 sospechosos.

(b) En el primer índice es fácil descartar entre la norma 1 y 2, ya que la segunda opción tiene más alto ambos índices. Luego, tenemos que elegir entre éste, que tiene una exactitud de 0.7768 y un índice de fraudes acertados de 0.1040. Como no se puede decidir por los índices, siguiendo el criterio planteado calculamos:

$$\frac{74 - 70}{692 - 671} = \frac{4}{21} \approx 0.19 > 0.0530,$$

por lo tanto, nos quedamos con el clasificador sin agregar índice, el primer clasificador.

(c) En el segundo índice también descarto el cálculo con norma 1, quedándonos con el realizado con norma 2 que devuelve una exactitud de 0.7794, un índice de fraudes acertados de

0,1030 y un listado de 68/659. Comparamos éste con el primer clasificador, y realizamos las cuentas siguiendo el criterio planteado:

$$\frac{74-68}{692-659} = \frac{6}{33} \approx 0.1818 > 0.0530,$$

por lo tanto, nos quedamos con el clasificador sin índices.

(d) En el tercer índice tengo 6 posibilidades, pero los dos clasificadores con $c = 2$ tienen los dos ítems, (v) y (vi), más altos que los otros cuatro clasificadores, por ende, tomamos $c = 2$ y resta elegir si tenemos en cuenta los extremos o no. Sin extremos tenemos una exactitud de 0.8024, un índice de fraudes acertados de 0.1270 y un listado de 77/605. Con extremos tenemos una exactitud de 0.8047, un índice de fraudes acertados de 0.1240 y un listado de 73/590. Por ende, realizamos las cuentas del criterio planteado para decidir:

$$\frac{77-73}{605-590} = \frac{4}{15} \approx 0.2600 > 0.0530,$$

por lo tanto, tomamos $c = 2$ sin extremos. Comparamos éste con el primer clasificador, y como ambos ítems son más altos me quedo con ésta opción.

Por ende, analizando sólo con el núcleo RBF, obtuvimos como mejor índice al 3° tomando $c = 2$, y sin tener en cuenta los extremos, obteniendo una exactitud de 0,8024, un índice de fraudes acertados de 0,1270 y un listado de 77/605. Comparamos ahora con el resto de los núcleos y vemos si alguno supera esto.

Comparando con los datos del primer clasificador, del clasificador con el primer índice, con el segundo y con el tercero con núcleo lineal y cuadrático no hay ningún clasificador que supere los resultados con el índice elegido hasta el momento.

Comparando con el núcleo polinomial, el que tiene mejor actuación es tomando $c = 2.5$ con extremos, que tiene una exactitud de 0,8047, un índice de fraudes acertados de 0.1250 y un listado de 74/592. Comparamos éste con el clasificador elegido hasta el momento, realizando el procedimiento explicado. Tenemos:

$$\frac{77-74}{605-592} = \frac{3}{13} \approx 0.2310 > 0.05,$$

por ende nos quedamos con el índice que ya habíamos elegido.

Como conclusión de este análisis tomamos el 3° índice con $c = 2$ y sin extremos y obtenemos un clasificador:

- exactitud: 0.8024
- índice de fraudes elegidos: 0.1270
- listado: $\frac{77}{605}$

4.4. Extracción de características 2: reescribiendo los datos

En la subsección anterior entrenamos al clasificador con los perfiles de consumos de cada cliente, más un índice extra calculado en base a estos datos. En esta subsección presentamos otras posibilidades para entrenar el clasificador con los mismos datos, es decir, cambiar el perfil de consumo por una reescritura de estos datos con la motivación de que las características que queremos representar de ambas clases estén mejor representadas, y así ayudar a que el clasificador etiquete de manera más eficiente los datos invisibles.

Presentamos tres nuevas maneras de ingresar los datos, y entrenaremos al clasificador con cada una de ellas y un índice: primero con el índice calculado anteriormente, y luego, con un nuevo índice utilizando también la idea de cambios notables y el núcleo RBF que ya tuvieron éxito, pero calculado en base a estos nuevos datos. Para esta segunda forma utilizamos de nuevo todas las opciones de c, con y sin los extremos de los datos.

Plasmamos en las siguientes tres subsecciones dos tablas con los resultados obtenidos de las diferentes formas, y a continuación el análisis para la elección o descarte de la modalidad en relación al clasificador que ya tenemos. Luego, enunciaremos la conclusión que obtenemos de estos tres análisis.

4.4.1. Entrenando con las diferencias de consumos

En este caso extraeremos 11 características por consumidor más el índice calculado en la subsección anterior. Tomando como punto de partida el éxito del índice de cambios notables, entrenaremos con las diferencias de consumos de un bimestre a otro, y así serán relacionados dos consumidores que hayan realizadodo cambios similares en sus consumos, independientemente si dichos consumos se encuentren en rangos muy distintos de valores. Por ejemplo, si dos consumidores tienen perfiles de consumos constantes, aunque una constante sea mucho más grande que la otra, para este clasificador, serán tomados ambos de la misma forma y muy probablemente los clasifique igual.

El cálculo de estas diferencias ya fue calculado en el momento de generar el índice de cambios notables. Es por esto, que el código en general es muy similar. Este código calcula estas diferencias, utiliza el índice elegido, entrena con los 12 datos y verifica en el conjunto de testeo. Calculamos además los parámetros que nos permiten comparar un clasificador con otro. El código se llama *entrenandocondiferencias*. Completamos la Tabla 13 con la información obtenida del mismo.

Ahora modificamos el código, lo llamaremos *entrenandocondiferencias2* y en éste dejamos fijo el núcleo RBF y calculamos el índice con sus variaciones a partir de los datos ingresados. Los resultados se muestran en la Tabla 14.

Recordemos que en la subsección anterior obtuvimos un clasificador con 0.8024 de exactitud, 0.1270 de índice de fraude acertados y un listado de 77/605. Como podemos ver en la Tabla 13, quedan descartadas todas las opciones que no quedan descartadas por tener ambos items más bajos son con el núcleo polinomial y RBF. Analizamos estos casos con el criterio explicado:

- Núcleo polinomial: cuenta con un listado de 56/560. Luego,

$$\frac{77-56}{605-560} = \frac{21}{45} \approx 0.4640 > 0.0530,$$

Kernel	(IV)	(V)	(VI)
linear	997	0.0953	0.6884
quadratic	742	0.1038	0.7586
polynomial	560	0.1000	0.8034
rbf	305	0.0918	0.8671

Tabla 13: Segundo clasificador: entrenando con diferencias de consumos y el índice calculado.

Núcleo RBF		$c = 2$	$c = 2.5$	$c = 3$
Sin extremos	(IV)	265	274	255
	(V)	0.0868	0.1058	0.1020
	(VI)	0.8767	0.8777	0.8818
Con extremos	(IV)	255	289	271
	(V)	0.1137	0.1142	0.1070
	(VI)	0.8838	0.8754	0.8786

Tabla 14: Segundo clasificador: entrenando con diferencias de consumos y nuevo índice.

por ende, rechazamos esta opción.

- Núcleo RBF: cuenta con un listado de 28/305. Luego,

$$\frac{77-28}{605-305} = \frac{49}{300} = 0.1630 > 0.0530,$$

por ende, rechazamos esta opción.

De la Tabla 14 podemos ver que la opción con extremos y $c = 2$ (que tiene un listado de 29/255) supera a todas las otras opciones salvo a la opción con extremos y $c = 2.5$ (que tiene un listado de 33/289). Elegimos entre estas dos opciones la segunda, ya que realizamos:

$$\frac{33-29}{289-255} = \frac{4}{34} \approx 0.1176 > 0.0530,$$

y recordamos que esto nos dice que conviene el listado con más cantidad de sospechosos. Comparamos esta opción con la que venimos trabajando, y conviene seguir con esta última, ya que:

$$\frac{77-33}{605-289} = \frac{44}{316} \approx 0.1392 > 0.053.$$

Por lo tanto rechazamos esta opción de entrenamiento y seguimos con el clasificador que encontramos en la subsección anterior.

4.4.2. Entrenando con los consumos promedios diarios

En este caso, tomaremos la idea utilizada en [18] y [19]. En estos trabajos, no entrenan con los consumos de cada mes sino con el consumo promedio diario, es decir, con el valor de gasto promedio diario de electricidad de cada mes. Debido a que las mediciones no se realizan en un mismo día de cada mes de manera exacta, quizás en una medición se toma en cuenta lo consumido en 5 semanas, y en la otra, en 3 semanas. El clasificador podría interpretar que acá existe una caída de consumo, cuando muy probablemente el consumo se haya mantenido estable, fomentando así error en la clasificación del programa.

Dados dos pares de datos de fecha de medición y consumo del mes hasta dicha fecha, de dos meses consecutivos, el consumo promedio diario es calculado como el cociente del segundo mes por la cantidad de días transcurridos entre una medición y otra, aunque no aclara bien cómo se calculan estos días a partir de dos fechas.

Nosotros tenemos consumos bimestrales, y dadas dos fechas con días D_1 y D_2 de sus respectivos meses, esta cantidad de días la calculamos como $61 + D_2 - D_1$, ya que 8 de los 12 bimestres tienen 61 días.

Para esto requeriremos el dato de las fechas de cada medición, las cuales se encuentran guardadas en la columna $C\{2\}$. Se guardarán sólo los días en un vector columna F. El archivo que calcula estos consumos promedios, le anexa el índice elegido, entrena con estos 12 datos al clasificador resolviendo el problema de optimización con el método QP, clasifica el conjunto de testeo y devuelve los valores de los parámetros que utilizamos para evaluar a los clasificadores. La rutina tiene el nombre de *consumosdiarios*, y las líneas del mismo que calculan los consumos promedios son las siguientes:

```
global A n clientes nroclientes nl ;
P = [ ] ;

F = ones( nl , 1 ) ;
m = 1 ;

for k = 1 : nroclientes
    if lecturasxcliente( k ) == 12
       P( m , 1 ) = A( m , 1 ) ;
       S = F( C{ 1 } == clientes( k ) ) ;
       S = 61 * ones( 11 , 1 ) + S( 2 : 12 ) - S( 1 : 11 ) ;
       P( m , 2 : 12 ) = A( m , 4 : 14 ) ./ S' ;
       m = m + 1 ;
    else
       k = k + 1 ;
    end
end
```

El resto del código es similar a lo trabajado hasta el momento. Los datos obtenidos del mismo están en la Tabla 15.

Kernel	(IV)	(V)	(VI)
linear	1123	0.0944	0.6551
quadratic	667	0.1214	0.7851
polynomial	611	0.1097	0.7941
rbf	583	0.1184	0.8044

Tabla 15: Segundo clasificador: entrenando con consumos de promedios diarios y primer índice.

Ahora modificamos el código, lo llamaremos *consumosdiarios2* y en éste dejamos fijo el núcleo RBF y calculamos el índice con sus variaciones a partir de los datos ingresados. Los resultados están en la Tabla 16.

Núcleo RBF		$c = 2$	$c = 2.5$	$c = 3$
Sin extremos	(IV)	651	597	583
	(V)	0.1091	0.1072	0.1115
	(VI)	0.7839	0.7967	0.8018
Con extremos	(IV)	574	583	613
	(V)	0.1185	0.1115	0.1191
	(VI)	0.8066	0.8018	0.7973

Tabla 16: Segundo clasificador: entrenando con consumos de promedios diarios y nuevo índice.

Como podemos ver en la Tabla 15 el único caso que no queda descartado de manera directa es con el núcleo RBF. Este caso ofrece un listado de 69/583, el cual compararemos con el clasificador que tenemos seleccionado hasta el momento y nos quedamos con este último por tener un listado más grande de sospechosos, ya que

$$\frac{77-69}{605-583} = \frac{8}{22} \approx 0.3630 > 0.0530.$$

Analizamos ahora la Tabla 16. Notamos que el clasificador con extremo y $c = 2$ supera a todos los otros casos excepto al clasificador con extremos y $c = 2.5$. El listado del primero es de 68/574 y el del segundo es de 73/613, comparamos ambos listados y elegimos el segundo ya

que

$$\frac{73-68}{613-574} = \frac{5}{39} = 0.1282 > 0.0530.$$

Pero, entre éste y el listado que tenemos hasta el momento (77/605) elegimos este último ya que en menos sospechosos tiene más aciertos. En conclusión, rechazamos este modo de entrenar al clasificador.

4.4.3. Entrenando con diferencias de consumos promedios diarios

En este caso mezclamos los dos anteriores buscando obtener un mejor clasificador, que contenga los beneficios de ambos. Extraeremos 10 datos por consumidor, y agregaremos el índice elegido. Los 10 datos serían las diferencias de los consumos promedios diarios de cada bimestre. Es decir, calculamos primero los 11 consumos promedios diarios por bimestre al igual de antes, y luego calculamos las diferencias de ellos. Por ende, mezclamos las cuentas de ambos. Este código, con la resolución del problema de optimización de SVM y el cálculo de los parámetros que nos permiten analizar el rendimiento del clasificador se encuentran en el código *entrenandocondiferenciaspromedios*, y ponemos a continuación, las líneas que calculan estas diferencias de los promedios.

```
global A n clientes nroclientes nl ;

P = [ ] ;

F = ones( nl , 1 ) ;
m = 1 ;

for k = 1 : nroclientes
    if lecturasxcliente( k ) == 12
       P( m , 1 ) = A( m , 1 ) ;
       S = F( C{ 1 } == clientes( k ) ) ;
       S = 61 * ones( 11 , 1 ) + S( 2 : 12 ) - S( 1 : 11 ) ;
       P( m , 2 : 12 ) = A( m , 4 : 14 ) ./ S' ; %calculo los consumos promedios
       P( m , 2 : 11 ) = P( m , 3 : 12 ) - P( m , 2 : 11 ) ; %calculo las
          diferencias y las guardo en los primeros 10 lugares
       m = m + 1 ;
    else
       k = k + 1 ;
    end
end

P = P( : , 1 : 11 ) %Borro la última columna
```

El resto del código es similar a lo que se viene trabajando. Ponemos en la Tabla 17 los resultados obtenidos con los distintos núcleos:

Ahora modificamos el código, lo llamaremos *entrenandocondiferenciaspromedios2* y en éste

Kernel	(IV)	(V)	(VI)
linear	969	0.1073	0.7032
quadratic	661	0.1120	0.7826
polynomial	488	0.1086	0.8245
rbf	260	0.1000	0.8802

Tabla 17: Segundo clasificador: entrenando con diferencias de consumos de promedios diarios.

dejamos fijo el núcleo RBF y calculamos el índice con sus variaciones a partir de los datos ingresados. Presentamos los resultados en la Tabla 18.

Núcleo RBF		$c = 2$	$c = 2.5$	$c = 3$
Sin extremos	(IV)	284	292	288
	(V)	0.1021	0.1096	0.1076
	(VI)	0.8745	0.8738	0.8745
Con extremos	(IV)	278	285	279
	(V)	0.1187	0.1088	0.1111
	(VI)	0.8790	0.8754	0.8774

Tabla 18: Segundo clasificador: entrenando con diferencias de consumos y nuevo índice.

Observamos la Tabla 17 y las opciones que no quedan descartadas en primera instancia son con el núcleo polinomial y RBF. Analizamos estos casos:

- Núcleo polinomial: ofrece un listado de 53/488 el cual rechazamos ya que

$$\frac{77-53}{605-488} = \frac{24}{117} \approx 0.2050 > 0.0530.$$

- Núcleo RBF: ofrece un listado de 26/260 el cual rechazamos ya que

$$\frac{77-26}{605-260} = \frac{51}{345} \approx 0.0840 > 0.053.$$

Analizamos luego la Tabla 18, y notamos que la opción con extremos y $c = 2$, que tiene un listado de 33/278, supera a todas las opciones de manera directa, excepto a la opción con

extremos y $c = 3$ pero que tiene un listado de 31/279, por lo cual también queda desartada, al tener más sospechosos y menos aciertos. Comparamos entonces la primera opción con la que venimos trabajando, y nos quedamos con esta última ya que

$$\frac{77-33}{605-278} = \frac{44}{327},$$

y esto nos dice que nos quedemos con el listado más grande, por ende, rechazamos esta manera de entrenar.

4.4.4. Conclusión de entrenamiento

Concluimos que bajo los criterios seleccionados para elegir o rechazar un clasificador ante otro conviene entrenar con los perfiles de consumos y el índice elegido anteriormente, es decir, seguimos con el clasificador elegido hasta la subsección anterior.

4.5. Normalización

Como vimos en los trabajos de Nagi, los datos utilizados para el entrenamiento son normalizados. Cada vector de datos de cada cliente, sin tener en cuenta el índice de credibilidad que se le agrega, es llevado a un vector con valores entre 0 y 1.

En [9] se explica que escalar antes de aplicar SVMs es muy importante. La principal ventaja del escalado es evitar darle atributos a rangos de números más grandes, evitar que dominen a los rangos más pequeños. Además, otra ventaja es evitar las dificultades numéricas durante los cálculos. Esto se debe a que, como vimos en la Sección 1, los núcleos usualmente dependen de un producto interno de los vectores de características de los usuarios. Por lo tanto, si los valores de estas características son muy grandes podrían causar problemas numéricos. Se recomienda en el trabajo estudiado que se escale linealmente cada característica al rango $[-1, +1]$ o $[0, 1]$.

En este último trabajo se aclara también que debe usarse el mismo método para escalar el conjunto de entrenamiento que el de testeo. Por ejemplo, si la primer característica del conjunto de entrenamiento es llevado de $[-10, +10]$ a $[-1, +1]$, y supongamos que la primer característica del conjunto de testeo está en el rango $[11, +8]$, ésta tiene que ser llevada a $[-1.1, +0.8]$.

Notar que tenemos dos posibilidades para normalizar los datos: normalizar por cliente, como en los trabajos de Nagi, y normalizar por atributo, como en el trabajo citado, y que en éste último tenemos que tener un tratamiento adicional con los datos del conjunto de testeo antes de utilizar el clasificador, hay que tener cuidado de utilizar la función lineal por característica ya encontrada, y no usar nuevas. En el caso de normalizar por cliente esto no hace falta, ya que se crea una función lineal por cliente.

Vamos a tomar el conjunto de entrenamiento seleccionado y lo normalizaremos de ambas maneras. Luego entrenamos al clasificador, resolvemos el problema de optimización con QP, y calculamos los parámetros que nos permiten analizar el rendimiento de los clasificadores. Los códigos que realizan esto se denominan *normalizacionxcliente* y *normalizacionxcaracteristica*, son modificaciones de *entrenandocondiferenciaspromedios*, ya que agregamos en éste la normalización de los datos, sin el índice, y seguimos igual. Colocamos a continuación los fragmentos agregados en cada caso. Elegimos en ambos casos escalar linealmente al rango $[0, 1]$.

```
% Normaliza por cliente
m = min( P' )' ; % vector columna con el mínimo de cada fila
M = max( P' )' ; % vector columna con el máximo de cada fila

P = ( P - (m * ones( 1 , n ) ) ) ./ ( ( M - m ) * ones( 1 , n ) ) ;

% Normaliza por característica
m = min( P( 1 : 2700 , : ) ) ; % vector fila con el mínimo de cada columna
M = max( P( 1 : 2700 , : ) ) ; % vector fila con el máximo de cada columna

P = ( P - ( ones( n , 1 ) * m ) ) ./ ( ones( n , 1 ) * ( M - m ) ) ;
```

Ejecutamos ambos con todos los núcleos y colocamos la información obtenida en las Tablas 19 y 20.

Kernel	(IV)	(V)	(VI)
linear	1155	0.1004	0.6513
quadratic	689	0.1205	0.7794
polynomial	626	0.1166	0.7931
rbf	605	0.1270	0.8024

Tabla 19: Segundo clasificador: entrenando con normalización de datos por característica.

Kernel	(IV)	(V)	(VI)
linear	1288	0.8385	0.6036
quadratic	951	0.0967	0.7012
polynomial	411	0.0998	0.8415
rbf	66	0.2272	0.9353

Tabla 20: Segundo clasificador: entrenando con normalización de datos por cliente.

Podemos notar que el resultado obtenido normalizando por cliente y utilizando el núcleo RBF supera ampliamente tanto las otras opciones como el clasificador que teniamos hasta el momento.

4.5.1. Conclusión

Notamos que normalizando los datos de los perfiles de consumo de pasamos a tener

- Exactitud: de 0.8024 a 0.9353.
- Índice de fraudes acertados: de 0.1270 a 0.227.

4.6. Optimización de parámetros utilizando búsqueda en grilla y validación cruzada

En la sección anterior decidimos cómo entrenar el clasificador a partir de los datos que nos fueron brindados. Extrajimos características que ayuden a separar mejor los datos y las agregamos a ellos en forma de índice. Reescribimos los datos tomando diferencias de consumos promedios diarios con la idea de extraer de manera más certera la información. Luego normalizamos para equiparar los niveles de consumos. Pero estuvimos usando los parámetros de penalización y de núcleo que están incorporados en Matlab por default, estos son, $C = 1$ y $\sigma = 1$ para el núcleo RBF.

En esta sección elegiremos estos parámetros a través de una búsqueda en grilla, y para seleccionar los mejores evaluaremos su actuación en el modelo por medio del uso de la validación cruzada, al igual que lo usado en [18] y explicado en la Sección 2.5.3. En el trabajo de Nagi se realizaba una búsqueda con γ en lugar de σ, pero el razonamiento es análogo (recordemos que $\gamma = 1/(2\sigma^2)$). Lo que buscamos con la validación cruzada es primero encontrar una alta exactitud, al igual que lo realizado en los trabajos de Nagi, y segundo, buscamos un mejor índice de fraudes acertados.

Como vamos a entrenar 2700 datos seguimos los consejos de [9]. Primero realizamos una búsqueda en una grilla gruesa con un subconjunto de entrenamiento, en este caso, elegimos los primeros 1000. Para cada par (C, σ) calculamos la exactitud (o índice de fraudes acertados) de este par realizando una validación cruzada en el conjunto de entrenamiento. Dividimos entonces en 10 subconjuntos de 100 elementos, y clasificamos cada subconjunto en base a los otros 9 restantes. En base a esto, se elige el mejor par de parámetros, o el primero que supera un determinado valor establecido.

Luego realizamos una búsqueda en una grilla más fina, a partir del par seleccionado, y realizamos el mismo procedimiento hasta obtener el par final.

Calculamos la exactitud e índice de fraudes acertados vía validación cruzada con los valores por defecto, ya que el objetivo es mejorar la actuación de estos parámetros. El código que realiza esto se denomina *validacioncruzadadefault*, y el resultado obtenido fue:

- Exactitud: 0.8167.
- Índice de fraudes acertados: 0.0896.

Es por esto que decidimos que los criterios de parada serán:

- En la grilla gruesa con 1000 elementos: pediremos una exactitud mayor a 0.85 y un índice de fraudes acertados mayor a 0.1.
- En la grilla fina con 2700 elementos: pediremos una exactitud mayor a 0.90 y un índice de fraudes acertados mayor a 0.15.

Los valores de C se recorrerán de mayor a menor, con el objetivo de evitar acotar de más el conjunto factible del problema de optimización, recordando que este valor determinará la cota superior en la restrincción de caja que del problema dual.

Luego, para cada caso, un par (C, σ) como candidato. Luego, entrenaremos a nuestro clasificador con cada uno de estos pares encontrados reemplazando los parámetros de default, y calculamos cuántos fraudes hay según el clasificador, cuál es el porcentaje de fraudes acertados y cuál es la exactitud obtenida, para analizar en base a esto las mejoras que se obtienen al cambiar los parámetros, y elegir así que par seleccionamos, ya sea uno de los encontrados, o el propuesto por defecto.

Realizaremos a su vez, dos tipo de búsqueda en grilla:

(a) Con las grillas utilizada por Nagi en [18]: éstas son

$$C = \left[2^{-5}, 2^{-3}, \ldots, 2^{15}\right],$$

y cambiando γ por σ, tenemos,

$$\sigma = \left[2^{-15}, 2^{-13}, \ldots, 2^{3}\right].$$

Utilizamos ésta como grilla gruesa, y una vez elegido el primer par (C, σ), realizamos una grilla más fina. Si el par seleccionado es $\left(2^a, 2^b\right)$, la segunda grilla será:

$$C = \left[2^{a-2}, 2^{a-2+0.25}, \ldots, 2^{a}, \ldots, 2^{a+2-0.25}, 2^{a+2}\right],$$

y

$$\sigma = \left[2^{b-2}, 2^{b-2+0.25}, \ldots, 2^{b}, \ldots, 2^{b+2-0.25}, 2^{b+2}\right].$$

La idea de esta segunda grilla es propuesta en [9].

(b) Cambiamos en este caso la escala que se utiliza para C, seguimos los consejos de [10] y utilizamos información del conjunto de entrenamiento y del núcleo que usamos para elegir en donde buscar C. Calculamos

$$C_{def} = \frac{1}{\sum_{i=1}^{N} K\left(x_i, x_i\right)},$$

donde N es la cantidad de vectores de entrenamiento y K la función de núcleo seleccionada, y tomamos la escala para C de la siguiente manera:

$$C = \left[10^{-4} C_{def}, \ldots, 10^{4} C_{def}\right],$$

para la grilla gruesa, y para la fina seguimos con la idea anterior, si el parámetro C seleccionado es $C = 10^a$, escalando ahora los exponentes cada 0.25 desde $a - 1$ hasta $a + 1$, obteniendo para la segunda etapa la siguiente escala:

$$C = \left[10^{a-1} C_{def}, 10^{a-1+0.25} C_{def}, \ldots, 10^{a+1-0.25} C_{def}, 10^{a+1} C_{def}\right].$$

La escala para σ se mantiene igual que en el ítem anterior.

Luego para cada tipo de grilla realizamos dos códigos, uno que busca una exactitud alta y el otro, que busca un índice de fraudes acertados alto. Colocamos las líneas de los códigos, resultados obtenidos y actuación de dichos resultados en el entrenamiento de los datos en las siguientes cuatro subsecciones. Luego, concluiremos qué par nos brindó mas eficacia a la hora de elaborar nuestro clasificador.

4.6.1. Búsqueda en grilla independiente

Llamamos grilla independiente a la que no depende del conjunto de entrenamiento, es la explicada en el ítem (a). Utilizamos esta grilla para buscar tanto exactitud como índice de fraudes acertados alto, y colocamos a continuación las líneas de código utilizadas, los resultados obtenidos de las grillas y la actuación de los parámetros obtenidos al momento de implementarlos en el modelo.

(a) Buscamos exactitud alta:

El código que realiza la búsqueda en una grilla independiente buscando una alta exactitud se llama *gridsearch1* y lo colocamos a continuación. En las próximas secciones sólo explicitaremos las modificaciones en base a éste.

```
% Grilla gruesa con los primeros 1000.
global P ;
bool = 1 ;
Sigma = -15 : 2 : 3 ;
Sigma = 2.^Sigma ;
C = -5 : 2 : 15 ;
C = 2.^C ;
E = [ ] ;
% options=statset('Maxiter',100000); %Por default es 15000
for i = length( C ) : - 1 : 1
    if bool == 1
       for j = 1 : length( Sigma ) ; % Me muevo por todos los pares (C,gamma)
           y = zeros( 100 , 1 ) ;
           %k=1;
           svmStruct = svmtrainmodificado( P( 101 : 1000 , 2 : end ) , X( 101 :
               1000 ) , 'kernel_function' , 'rbf' , 'rbf_sigma' , Sigma( j ) ,
               'method' , 'QP' , 'boxconstraint' , C( i ) ) ;
           y( 1 : 100 ) = svmclassify( svmStruct , P( 1 : 100 , 2 : end ) ) ;

           for k = 2 : 9 % 10-validación cruzada
               Pk = [ P( 1 : ( k - 1 ) * 100 , 2 : end ) ; P( k * 100 + 1 :
                   1000 , 2 : end ) ] ;
               xk = [ X( 1 : ( k - 1 ) * 100 ) ; X( k * 100 + 1 : 1000 ) ] ;
               svmStruct = svmtrainmodificado( Pk , xk , 'kernel_function' ,
                   'rbf' , 'rbf_sigma' , Sigma( j ) , 'method' , 'QP' ,
                   'boxconstraint' , C( i ) ) ;
               y( ( k - 1 ) * 100 + 1 : k * 100 ) = svmclassify( svmStruct , P(
                   ( k - 1 ) * 100 + 1 : k * 100 , 2 : end ) ) ;
           end

           %k=10
           svmStruct = svmtrainmodificado( P ( 1 : 900 , 2 : end ) , X( 1 : 900
               ) , 'kernel_function' , 'rbf' , 'rbf_sigma' , Sigma( j ) ,
               'method' , 'QP' , 'boxconstraint' , C( i ) ) ;
```

```
            y(901:1000) = svmclassify( svmStruct , P( 901 : 1000 , 2 : end ) ) ;

            E(i,j) = sum( y == X( 1 : 1000 ) ) / 1000 ; % exactitud con este C
                y gamma

            if E( i , j ) > 0.85 ; %Si la exactitud cruzada es mas del 85%, me
                quedo con ese C y gamma
               exactitud = E( i , j ) ;
               C = C( i ) ;
               Sigma = Sigma( j ) ;
               bool = 0 ;
               break ;
            end
         end
      else
         break
      end
end

exactitud = max( max( E ) ) ;
Sigma = Sigma( max( E ) == exactitud ) ; % Me quedo con C y Gamma que dan
   mayor exactitud
Sigma = Sigma( end ) ; % Si hay muchos que me dan un porcentaje alto, me
   quedo con el mas grande
C = C( max( E' ) == exactitud ) ;
C = C( end ) ;

% Grilla más fina a partir del C y Gamma encontrado

C2 = log2( C ) ;
C2 = ( C2 - 2 ) : 0.25 : ( C2 + 2 ) ;
C2 = 2.^C2 ;
Sigma2 = log2( Sigma ) ;
Sigma2 = ( Sigma2 - 2 ) : 0.25 : ( Sigma2 + 2 ) ;
Sigma2 = 2.^Sigma2 ;

E2 = [ ] ;
bool = 1 ;
for i = length( C2 ) : -1 : 1
   if bool == 1
      for j = 1 : length( Sigma2 ) % Me muevo por todos los pares (C,gamma)
         y = zeros( 2700 , 1 ) ;
         %k=1
         svmStruct = svmtrainmodificado( P( 271 : 2700 , 2 : end ) , X( 271 :
            2700 ) , 'kernel_function' , 'rbf' , 'rbf_sigma' , Sigma2( j ) ,
            'method' , 'QP' , 'boxconstraint' , C2( i ) ) ;
```

```
            y( 1 : 270 ) = svmclassify( svmStruct , P( 1 : 270 , 2 : end ) ) ;

            for k = 2 : 9 % 10-validación cruzada
                Pk = [ P( 1 : ( k - 1 ) * 270 , 2 : end ) ; P( k * 270 + 1 :
                    1000 , 2 : end ) ] ;
                xk = [ X( 1 : ( k - 1 ) * 270 ) ; X( k * 270 + 1 : 1000 ) ] ;

                svmStruct = svmtrainmodificado( Pk , xk , 'kernel_function' ,
                    'rbf' , 'rbf_sigma' , Sigma2( j ) , 'method' , 'QP' ,
                    'boxconstraint' , C2( i ) ) ;
                y( ( k - 1 ) * 270 + 1 : k * 270 ) = svmclassify( svmStruct , P(
                    ( k - 1 ) * 270 + 1 : k * 270 , 2 : end ) ) ;
            end

            %k=10
            svmStruct = svmtrainmodificado( P( 1 : 2430 , 2 : end ) , X( 1 :
                2430 ) , 'kernel_function' , 'rbf' , 'rbf_sigma' , Sigma2( j ) ,
                'method' , 'QP' , 'boxconstraint' , C2( i ) ) ;
            y( 2431 : 2700 ) = svmclassify( svmStruct , P( 2431 : 2700 , 2 : end
                ) ) ;

            E2( i , j ) = sum( y == X( 1 : 2700 ) ) / 2700 ; % exactitud con
                este C y gamma

            if E2( i , j ) > 0.90 % Si la exactitud cruzada es mas del 85%, me
                quedo con ese C y gamma
               exactitud2 = E2( i , j ) ;
               C2 = C2( i ) ;
               Sigma2 = Sigma2( j ) ;
               bool = 0 ;
               break ;
            end
         end
      else
         break ;
      end
   end

   exactitud2 = max( max( E2 ) ) ;
   Sigma2 = Sigma2( max( E2 ) == exactitud2 ) ; % Me quedo con C y Gamma que dan
       mayor exactitud
   Sigma2 = Sigma2( end ) ;
   C2 = C2( max( E2' ) == exactitud2 ) ;
   C2 = C2( end ) ;
```

Realizando esto obtenemos que en el grilla gruesa una exactitud de 0.9300 en el par

$(C, \sigma) = (2^{15}, 2^{-15})$, y luego en la grilla más fina, una exactitud de 0.9389 en el par $(C, sigma) = (2^{17}, 2^{-17})$.

Cambiamos entonces en el entrenamiento los valores de default por el par encontrado: $(C, \sigma) = (2^{17}, 2^{-17})$, modificando de la siguiente forma:

```
svmStruct = svmtrainmodificado( P( 1 : 2700 , 2 : 14 ) , X( 1 : 2700 ) ,
    'kernel_function' , 'rbf' , 'rbf_sigma' , 2^(-17) , 'method' , 'QP' ,
    'boxconstraint' , 2^(17) ) ;
```

y el resultado obtenido fue el siguiente:

```
roban según el clasificador: 0.000000e+00
indices de fraudes acertados: NaN
exactitud: 9.468460e-01
```

si bien, logramos una exactitud alta el clasificador etiqueta a todos como no fraudulento (por ende el índice de fraudes acertados da 0/0), y esto no aporta a nuestro trabajo ya que no nos ofrece un listado de sospechosos, por ende, descartamos esta opción.

(b) Buscamos índice de fraudes acertados alto:

La diferencia es que buscamos un índice de fraudes acertados alto en lugar de una exactitud alta, por ende, cambiamos del código anterior estos cálculos:

```
E( i , j ) = sum( y == X( 1 : 1000 ) ) / 1000 ;
E2( i , j ) = sum( y == X( 1 : 2700 ) ) / 2700 ;
```

por estos:

```
E( i , j ) = ( sum( y == X( 1 : 1000 ) ) - sum( ( y + X( 1 : 1000 ) ) ==
    zeros( 1000 , 1 ) ) ) / sum( y ) ;
E2( i , j ) = ( sum( y == X( 1 : 2700 ) ) - sum( ( y + X( 1 : 2700 ) ) ==
    zeros( 2700 , 1 ) ) ) / sum( y ) ;
```

Los resultados obtenidos fue un acierto de 0.2000 con el par $(C, \sigma) = (2^9, 2^1)$ en la grilla gruesa, y finalmente en la grilla más fina un acierto de 0.1923 con el par $(C, \sigma) = (2^{11}, 2^0)$.

Entrenamos al clasificador con este último par y obtenemos los siguientes resutlados:

```
roban según el clasificador: 43
indice de fraudes acertados: 0.2325581
exactitud: 0.9394813
```

Lo cual brinda un listado de 10/43 y mejora el resultado que teniamos antes pasando a tener una exactitud 0.9353 a 0.9395, y un índice de fraudes acertados de 0.2273 a 0.2326.

4.6.2. Búsqueda en grilla que depende del conjunto de entrenamiento

Utilizamos en este caso las grillas descriptas en el ítem (b) del comienzo de esta sección. En este caso la escala de C depende del conjunto de entrenamiento. Utilizamos esta grilla para calcular exactitud e índice de fraudes acertados alto, y colocamos a continuación las líneas de código, los resultados obtenidos y como influyen estos resultados en el modelo que tenemos.

(a) Buscando exactitud alta:

En este caso lo que se modifica en el código es la grilla que se utiliza para la búsqueda del parámetro C, ya que se utiliza el C_{def} que mencionamos antes, que se determina en base al conjunto de entrenamiento y al núcleo elegido, en este caso RBF.

Se agrega por lo tanto en el código anterior la definición de C:

```
Cdef=0;
for i = 1 : 2700
    Cdef = Cdef + rbf_kernel( P( i , 2 : end ) , P( i , 2 : end ) ) ;
end
```

y se reemplaza esta grilla:

```
C = -5 : 2 : 15 ;
C = 2.^C ;
```

por esta otra:

```
c = -4 : 1 : 4 ;
c = 10.^c ;
C = ( 1 / Cdef ) * c ;
```

El código que realiza esto se denomina *gridsearch2* y se obtuvo como resultado una exactitud de 0.9300 en el par $(C, \sigma) = \left(\frac{1}{10^4 C_{def}}, 2^{-15}\right)$ de la grilla gruesa, y una exactitud de 0.9398 en el par $(C, \sigma) = \left(\frac{1}{10^6 C_{def}}, 2^{-17}\right)$ en la grilla más fina.

Entrenamos al clasificador con este par, y los resultados obtenidos son los siguiente:

```
roban según el clasificador: 0
indice de fraudes acertados: NaN
exactitud: 0.9468460
```

idéntico que el caso de la búsqueda en grilla independiente buscando exactitud alta. Por ende, descartamos este caso.

(b) Buscando índice de fraudes acertados alto:

En este caso combinamos la grilla descripta en el ítem anterior con los cambios para buscar índice de fraudes acertados alto en lugar de exactitud.

En este caso encontramos un índice de aciertos de 0.1667 en el par $(C, \sigma) = \left(\frac{1}{10^4 C_{def}}, 2^1\right)$ en la grilla gruesa, y un índice de 0.1923 en el par $(C, \sigma) = \left(\frac{1}{10^6 C_{def}}, 2^0\right)$ en la grilla más fina. Entrenamos al clasificador con éste último par y obtuvimos:

```
roban según el clasificador: 43
indices de fraudes acertados: 0.2325581
exactitud: 0.9394813
```

Estos resultados son idénticos a los que ya obtuvimos, por lo tanto entrenaremos con $\sigma = 1$ y tenemos como opciones a $C = 2^{11} = 2048$ o $C = \frac{1}{C_{def} * 10^6} = 370.3704$, y me quedo con el más alto que es el primero.

4.6.3. Conclusión

Notamos que dejando de utilizar el par de parámetros por defecto de $(C, \sigma) = (1, 1)$ pasamos a utilizar el encontrado por la búsqueda en grilla que es $(C, \sigma) = (2^{11}, 1)$, obtenemos las siguiente mejora:

- Exactitud: de 0.9353 a 0.9394813.
- Índice de fraudes acertados: de 0.227 a 0.2325581.

4.7. Conclusión final

Finalmente optamos por entrenar con los perfiles de consumo normalizados por cliente, agregando un índice que depende de las caídas que se producen en dicho perfil el cual penaliza a aquellos clientes que tienen una caída que es mas del doble de una caída estándar (c=2), sin tener en cuenta ni la primer ni la última diferencia de consumo. Optamos por el núcleo RBF y utilizaremos los parámetros $(C, \sigma) = (2^{11}, 1)$. Con estas elecciones logramos un clasificador que al ponerlo en prueba con el conjunto de testeo obtuvimos una exactitud de 0.9394813 y un índice de fraudes acertados de 0.2325581. Como guardamos los números de consumidores en la primera columna de la matriz A, el conjunto de testeo es del consumidos 2701 en adelante, y el clasificador etiqueta con 1 a los fraudulentos, el listado de sospechoso es simplemente:

```
A(2701:end,:)(clasif==1)
```

dando por finalizado este trabajo.

5. Trabajo a futuro

Nos queda como trabajo a futuro generalizar el clasificador a todos los consumidores, no solo a los que tienen 12 datos. La idea para realizar esto es corresponder los 2 años que tenemos como datos con los números enteros de 0 a 730, un entero por cada día, y elegimos 12 fechas fijas que disten entre ellas en 61 días. Luego, como contamos con las fechas de los consumos, graficamos el historial de datos de cada cliente. Para finalizar interpolamos estos datos y tomamos los 12 valores de consumo de las fechas prefijadas. Así, cada consumidor tendría 12 datos históricos. El procedimiento siguiente sería análogo al realizado.

Por otro lado, se sabe que la proporción de las clases que devuelve el clasificador es muy similar a la proporción que existe al momento de entrenar. Es por ello que, se podría averiguar cuánto se estima que es el porcentaje de consumidores fraudalentos sobre el total, y entrenar las clases con esta proporción. En nuestro trabajo entrenamos con 166 consumidores fraudulentos, y la idea sería completar el conjunto de entrenamiento con un subconjunto de consumidores limpios aleatorios de tal manera que genere la proporción deseada. Luego, se sigue con el procedimiento planteado.

Queda abierta también la posibilidad de buscar nuevos índices y nuevos modos de ingresar los datos de consumo, en busca de la mejora de resultados, como así también, la posibilidad de determinar nuevamente los parámetros a través de otros métodos, ya sea con el método de Fisher, con la utilización de un algoritmo genético, o con alguna otra propuesta.

Referencias

[1] Dimitri P Bertsekas. Nonlinear programming. 1999.

[2] Cristina García Cambronero and Irene Gómez Moreno. Algoritmos de aprendizaje: knn & kmeans. *Intelgencia en Redes de Comunicación, Universidad Carlos III de Madrid*, 2006.

[3] Olivier Chapelle, Vladimir Vapnik, Olivier Bousquet, and Sayan Mukherjee. Choosing multiple parameters for support vector machines. *Machine learning*, 46(1-3):131–159, 2002.

[4] Bertrand Clarke, Ernest Fokoue, and Hao Helen Zhang. *Principles and theory for data mining and machine learning*. Springer Science & Business Media, 2009.

[5] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[6] Keinosuke Fukunaga. *Introduction to statistical pattern recognition*. Academic press, 2013.

[7] Steve R Gunn et al. Support vector machines for classification and regression. *ISIS technical report*, 14, 1998.

[8] Guodong Guo, Stan Z Li, and Kap Luk Chan. Support vector machines for face recognition. *Image and Vision computing*, 19(9):631–638, 2001.

[9] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification. 2003.

[10] Thorsten Joachims. Support vector and kernel methods. *SIGIR-Tutorial, Cornell University Computer Science Department*, 2003.

[11] S Sathiya Keerthi and Chih-Jen Lin. Asymptotic behaviors of support vector machines with gaussian kernel. *Neural computation*, 15(7):1667–1689, 2003.

[12] Yung-Keun Kwon, Byung-Ro Moon, and Sung-Deok Hong. Critical heat flux function approximation using genetic algorithms. *Nuclear Science, IEEE Transactions on*, 52(2):535–545, 2005.

[13] Hsuan-Tien Lin and Chih-Jen Lin. A study on sigmoid kernels for svm and the training of non-psd kernels by smo-type methods. *submitted to Neural Computation*, pages 1–32, 2003.

[14] David G Luenberger and Yinyu Ye. *Linear and nonlinear programming*, volume 116. Springer Science & Business Media, 2008.

[15] José Mario Martınez and Sandra Augusta Santos. Métodos computacionais de otimização. *Colóquio Brasileiro de Matemática, Apostilas*, 20, 1995.

[16] Matlab. `http://www.mathworks.com/`.

[17] Tom M Mitchell et al. Machine learning. wcb, 1997.

[18] J Nagi, AM Mohammad, KS Yap, SK Tiong, and SK Ahmed. Non-technical loss analysis for detection of electricity theft using support vector machines. In *Power and Energy Conference, 2008. PECon 2008. IEEE 2nd International*, pages 907–912. IEEE, 2008.

[19] J Nagi, KS Yap, SK Tiong, SK Ahmed, and AM Mohammad. Detection of abnormalities and electricity theft using genetic support vector machines. In *TENCON 2008-2008 IEEE Region 10 Conference*, pages 1–6. IEEE, 2009.

[20] John Platt et al. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998.

[21] Marcos Gestal Pose. Introducción a los algoritmos genéticos. *Departamento de Tecnologías de la Información y las Comunicaciones Universidad de Coruña*, 2000.

[22] Herbert A Simon. Why should machines learn? In *Machine learning*, pages 25–37. Springer, 1983.

[23] Alex Smola and Vladimir Vapnik. Support vector regression machines. *Advances in neural information processing systems*, 9:155–161, 1997.

[24] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[25] Vladimir Naumovich Vapnik and Vlamimir Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.

[26] Wenjian Wang, Zongben Xu, Weizhen Lu, and Xiaoyun Zhang. Determination of the spread parameter in the gaussian kernel for classification and regression. *Neurocomputing*, 55(3):643–663, 2003.