Máquinas de vector de soporte aplicadas a la clasificación del cáncer de mama

John Esteban Castro Ramírez, Martin Ospina Uribe, Miguel Ángel Chacon López
Análisis numérico, Departamento de Ciencias Matemáticas, Escuela de Ciencias Aplicadas e Ingeniería
Medellín, Colombia
jecastror@eafit.edu.co, mospinau1@eafit.edu.co, machaconl@eafit.edu.co

Resumen

El Machine Learning o aprendizaje automático es una disciplina de las ciencias de la computación y una rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas o algoritmos mediante los cuales las máquinas aprendan de manera automática. En este trabajo se hará uso del Machine Learning a través de las máquinas de vector soporte, el cual es un algoritmo de clasificación usando aprendizaje automático, de esta manera se clasifica el diagnóstico del cáncer de mama a partir de un conjunto de datos de un estudio de la Universidad de Wisconsin aplicado a 569 mujeres de acuerdo a ciertas características relevantes a la hora de determinar si una masa mamaria es benigna o maligna, de esta manera el algoritmo se entrena con el 80 % de los datos y valida con el 20 % en donde se obtienen unos buenos resultados en cuanto a medidas de precisión del modelo, sin embargo, se mejora esta precisión a través de una combinación convexa de funciones Kernel, determinando el parámetro α de la combinación convexa a través de simulación Montecarlo, concluyendo que la combinación convexa de funciones kernel en las máquinas de vector soporte puede arrojar mejores resultados en problemas de clasifación respecto a solo usar una función kernel.

Palabras clave: Machine Learning, clasificación, máquinas de vector soporte, cáncer de mama, Kernel, combinación convexa.

1. Introducción

El Machine Learning es una disciplina de la Inteligencia Artificial que a través de distintos algoritmos proporciona a los ordenadores la capacidad de identificar patrones en datos masivos para elaborar predicciones, particularmente los algoritmos de aprendizaje supervisado cuentan con un aprendizaje previo a partir de datos históricos [1]. De esta manera, las máquinas de vector soporte (SVM) son un algoritmo de aprendizaje supervisado que se utiliza en muchos problemas de clasificación y regresión, como por ejemplo en temas relacionados con medicina, reconocimiento de imágenes y voz y procesamiento de señales [2]. En el presente trabajo de investigación se muestra una aplicación de las máquinas de vector soporte dirigido a la clasificación del cáncer de mama tomando como conjunto de datos 569 muestras que contienen datos extraídos a partir de una aspiración con aguja fina, realizadas por un estudio de la Universidad de Wisconsin en Estados Unidos; de esta manera,

analizamos los resultados de la SVM usando distintas funciones Kernel como la lineal, polinómica y gaussiana, sin embargo, algunas veces se puede mejorar la precisión y los resultados obtenidos usando una combinación convexa de funciones kernel, por ende, se realizoadicha combinación para así mejorar la precisión y evitar posibles resultados erróneos de la predicción o clasificación.

2. Marco teórico

Las máquinas de soporte vectorial (SVM:*Support Vector Machines*), son un algoritmo de aprendizaje supervisado de clasificación o regresión basado en hiperplanos de margen máximo (definidos por los vectores de soporte) propuesto por Vladimir Vapnik. Este algoritmo básicamente consiste en que dado un conjunto de ejemplos de entrenamiento o muestras, podemos etiquetar la clase a la cuál pertenece cada uno de estos para entrenar la SVM que permita construir un modelo que prediga la clase a la cuál pertenecen otro conjunto de datos, el algoritmo busca un hiperplano que separé de forma óptima a los puntos correspondientes a una clase con los de la otra, es decir el hiperplano tiene la máxima distancia con los puntos que están más cerca de el mismo.

En términos más formales, debemos de considerar un conjunto de elementos representados por (x_i, y_i) que llamaremos ejemplos, cada uno con una serie de características que se almacenan en un vector x_i y una categoría que se almacena en y_i. Entonces, si podemos separar todos los ejemplos a través de un hiperplano de forma que cada ejemplo esté en el semiplano de la clase correcta, se dirá que el conjunto de ejemplos es separable. De esta manera, se derivan varios casos para la separación de los ejemplos o muestras, los cuales se presentan a continuación; sin embargo, en la vida real suelen ser más útiles los últimos correspondientes a ejemplos no separables linealmente.

2.1. Máquinas de vector soporte para clasificación binaria de ejemplos separables linealmente.

Consideraremos primeramente los conjuntos de datos que pueden ser separados linealmente, este caso se da cuando dado un conjunto separable de ejemplos $S = \{(x_1, y_1), ..., (x_n, y_n)\}$, donde $x_i \in \mathbb{R}^d$ y $y_i \in \{-1, 1\}$ se puede definir un hiperplano de separación como una función lineal que es capaz de separar dicho conjunto sin error

$$D(x_i) = (w_1 x_1 + ... + w_d x_d) + b =< w, x > +b \tag{1}$$

en donde, $w \in \mathbb{R}^d$, b $\in \mathbb{R}$ y w es un vector que parte desde el origen y es perpendicular al hiperplano de separación. De esta manera, cualquier punto x puede ser descrito con respecto a w por medio de la proyección que será $x \cdot \frac{w}{||w||}$, entonces el ejemplo (x_i, y_i) pertenecerá a una clase positiva o negativa, es decir debe cumplir las siguientes restricciones:

$$< w, x_i > +b \geq 0, \quad \text{si } y_i = 1$$

$$< w, x_i > +b \leq 0, \quad \text{si } y_i = -1$$

O equivalentemente y de forma más sencilla:

$$y_i D(x_i) \geq 0 \qquad \text{i=1,...,n}$$

Sin embargo, existen infinitos hiperplanos que cumplen esta restricción, algunos de ellos se muestran en la siguiente figura:

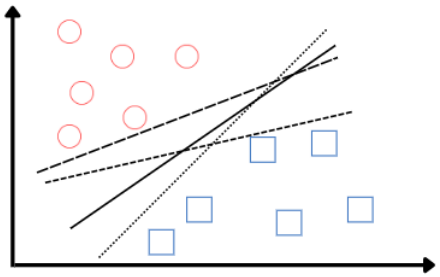

Figura 1: Algunos hiperplanos de separación que cumplen la restricción $y_i D(x_i) \geq 0, \quad$ i=1,...,n

Es allí, donde se debe de buscar un nuevo criterio para poder decidir o escoger el hiperplano de separación óptimo, para esto primero se debe definir el margen (τ) que es la mínima distancia entre el hiperplano de separación y el ejemplo más próximo a el de cualquiera de las 2 clases; Vapnik y Lerner propusieron en 1963, que el hiperplano de separación óptimo es aquel que maximiza el margen, de esta definición se obtiene la siguiente proposición.

Proposición: Un hiperplano de separación es óptimo si y sólo si equidista del ejemplo más cercano de cada clase.(Se puede demostrar fácilmente por reducción al absurdo)

A continuación se muestra un hiperplano de separación óptimo, en donde se ve que se cumple lo dicho en la proposición anterior.

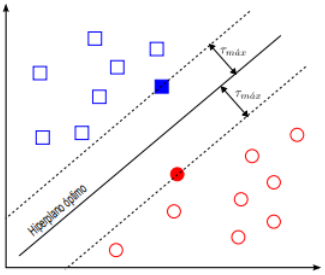

Figura 2: Hiperplano de separación óptimo

De esta manera, la distancia entre un hiperplano de separación y un determinado ejemplo x', viene dada por:

$$\frac{|D(x')|}{||w||_2} \tag{2}$$

En donde w es un vector perpendicular al hiperplano de separación óptimo, y debido a que cada ejemplo debe de cumplir la restricción de estar por fuera de los margenes, entonces se obtiene que:

$$\frac{y_i D(x_i)}{||w||_2} \geq \tau, \qquad i = 1, ..., n \tag{3}$$

De este modo, para los ejemplos que están en el margen o dicho de otra manera, que son los vectores soporte se cumplirá que:

$$\frac{y_i D(x_i)}{||w||_2} = \tau \tag{4}$$

De las ecuaciones anteriores, se deduce que encontrar el hiperplano de separación óptimo se reduce a hallar el w que maximiza a τ, para esto entonces se fija que $\tau||w||_2 = 1$ y la restricción dada en (3), se transforma en:

$$y_i D(x_i) \geq 1, \qquad i = 1, ..., n \tag{5}$$

El tamaño del margen está definido por:

$$(x_+ - x_-) \cdot \frac{w}{||w||} \tag{6}$$

En donde x_+ y x_- son los ejemplos o vectores soporte correspondientes a la muestra positiva y negativa, respectivamente.
Desarrollando un poco más esta ecuación, se obtiene que para el vector soporte de la muestra positiva x_+ de acuerdo a la ecuación (5):

$$\begin{aligned} y_i D(x_+) &= 1 \\ y_i(< w, x_+ > + b) &= 1 \\ (1)(< w, x_+ > + b) &= 1 \\ < w, x_+ > &= 1 - b \end{aligned} \tag{7}$$

Y de manera análoga para el vector soporte de la muestra negativa x_- se obtiene que $- < w, x_- >= 1 + b$, entonces, reemplazando estos resultados en la ecuación (6), se tiene que:

$$\begin{aligned}(x_+ - x_-)\frac{w}{||w||} &= \frac{1}{||w||}(< w, x_+ > - < w, x_- >) \\ &= \frac{1}{||w||}((1-b) + (1+b)) \\ &= \frac{2}{||w||}\end{aligned} \tag{8}$$

De esta manera, como se quiere maximizar dicho margen para encontrar el hiperplano de separación óptimo, se plantea el problema de optimización asociado para este problema.

Se debe maximizar $\frac{2}{||w||}$, y en los problemas de optimización las constantes no importan mucho entonces se puede pasar al problema equivalente que es maximizar $\frac{1}{||w||}$ e igualmente se puede pasar este problema a minimizar $||w||$ y finalmente se puede transformar el problema en minimizar $\frac{1}{2}||w||^2$, así, el problema de optimización queda dado de la siguiente manera.

$$\begin{gathered}\text{minimizar } \tfrac{1}{2}||w||^2 \\ \text{sujeto a: } y_i(< w, x_i > +b) - 1 = 0\text{; para los vectores soporte}\end{gathered}$$

Debido a que este es un problema de optimización no lineal, se hará uso de los multiplicadores de Lagrange. Primeramente, se sabe que se puede pasar al problema dual si la función a optimizar y las restricciones son convexas, el problema de optimización planteado cumple estas condiciones, por lo que se pasa al problema dual y en primer lugar, se debe de construir la función lagrangiana, que tiene como objetivo eliminar las restricciones

$$L(w, \lambda) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} \lambda_i(y_i(< w, x_i > +b) - 1); i = 1, ..., n \tag{9}$$

donde $\lambda_i \geq 0$ y son los denominados multiplicadores de Lagrange. Finalmente el problema se reduce a minimizar la función lagrangiana (9). La ventaja que se tiene es que debido al $||w||^2$, se tiene una función cuadrática y como estos problemas siempre tienen una forma de parábola, se tendrá un único mínimo global. Para resolver este problema, entonces primero se deriva la función lagrangiana

respecto a w y se iguala a 0, obteniendo así lo siguiente

$$\begin{aligned} \frac{\partial L(w,\lambda)}{\partial w} = w - \sum_{i=1}^{n} \lambda_i y_i x_i = 0 \\ w = \sum_{i=1}^{n} \lambda_i y_i x_i \end{aligned} \tag{10}$$

Y ahora se deriva respecto a todo lo que varíe, entonces se deriva respecto al sesgo (b)

$$\frac{\partial L(w,\lambda)}{\partial b} = \sum_{i=1}^{n} \lambda_i y_i = 0 \tag{11}$$

Ahora, se utilizan ambos resultados $w = \sum_{i=1}^{n} \lambda_i y_i x_i$ y $\sum_{i=1}^{n} \lambda_i y_i = 0$, se reemplazan en la función Lagrangiana (9), obteniendo de esta manera lo siguiente:

$$\begin{aligned} L(w,\lambda) &= \frac{1}{2}||w||^2 - \sum_{i=1}^{n} \lambda_i (y_i(< w, x_i > + b) - 1) \\ &= \frac{1}{2}(\sum_{i=1}^{n} \lambda_i y_i x_i) \cdot (\sum_{j=1}^{n} \lambda_j y_j x_j) - \sum_{i=1}^{n} \lambda_i [y_i(\sum_{j=1}^{n} \lambda_j y_j x_j + b) - 1] \\ &= \frac{1}{2}(\sum_{i=1}^{n} \lambda_i y_i x_i) \cdot (\sum_{j=1}^{n} \lambda_j y_j x_j) - \sum_{i=1}^{n} (\lambda_i y_i x_i \cdot \sum_{j=1}^{n} \lambda_j y_j x_j) - b \sum_{j=1}^{n} \lambda_j y_j + \sum_{j=1}^{n} \lambda_j \\ &= \frac{1}{2}(\sum_{i=1}^{n} \lambda_i y_i x_i) \cdot (\sum_{j=1}^{n} \lambda_j y_j x_j) - \sum_{i=1}^{n} (\lambda_i y_i x_i \cdot \sum_{j=1}^{n} \lambda_j y_j x_j) + \sum_{j=1}^{n} \lambda_j \\ &= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j x_i \cdot x_j - \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j x_i \cdot x_j + \sum_{j=1}^{n} \lambda_j \\ &= \sum_{j=1}^{n} \lambda_j - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j x_i \cdot x_j \end{aligned} \tag{12}$$

Así, el problema de optimización final será min $\sum_{j=1}^{n} \lambda_j - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j x_i \cdot x_j$, para esto suele ser más sencillo plantear el problema dual ya que será más sencillo de solucionar y representará

un menor coste computacional, de esta manera el problema dual será:

$$\begin{aligned} max \sum_{j=1}^{n} \lambda_j - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j x_i \cdot x_j \\ Sujeto\ a: \\ \sum_{i=1}^{n} \lambda_i y_i = 0 \\ \lambda_i \geq 0; i = 1, ..., n \end{aligned} \tag{13}$$

Este problema puede ser resuelto mediante técnicas de programación cuadrática, y luego de obtener su solución, para obtener la solución del primal simplemente se sustituye en la ecuación

$$D(x) = \sum_{i=1}^{n} \lambda_i^* y_i < x, x_i > + b^* \tag{14}$$

Además a partir de las condiciones de KKT (*Karush-Kuhn-Tucker*), también se obtiene que $\lambda_i[1 - y_i(< w, x_i > + b)] = 0$, i=1,...,n. Entonces, si en determinado ejemplo $\lambda_i > 0$, se cumplirá que

$$y_i(< w^*, x_i > + b^*) = 1 \tag{15}$$

Por tanto, solo aquellos ejemplos que tengan asociado un $\lambda_i > 0$ serán los vectores soporte y los únicos que intervienen en la construcción del hiperplano de separación óptimo.

De la ecuación (15) se puede despejar b^*, obteniendo que:

$$b^* = y_{VS} - < w^*, x_{VS} > \tag{16}$$

En donde, (x_{VS}, y_{VS}) es la tupla que satisface la igualdad anterior, es decir la tupla asociada a los vectores soporte. Sin embargo, en la práctica suele ser más robusto obtener b^* promediando los vectores soporte

$$b^* = \frac{1}{N_{VS}} \sum_{i \in VS} (y_i - < w^*, x_i >) \tag{17}$$

En donde, N_{VS} es la cardinalidad del conjunto formado por los vectores soporte.

2.2. Máquinas de vector soporte para clasificación binaria de ejemplos cuasi-separables linealmente

En la vida real, los problemas normalmente poseen ejemplos alejados o ruidosos y por tanto, no ser separables linealmente. Así, un ejemplo es no separable si no cumple con la condición siguiente:

$$y_i(< w, x_i > +b) \geq 1, \quad i = 1, ..., n \tag{18}$$

De este modo, se pueden dar dos casos, que el ejemplo caiga dentro del margen de la clase correcta de acuerdo con el hiperplano de separación, o que caiga en la otra clase. Por tanto, para abordar este problema se debe añadir a la condición (18) variables de holgura δ_i, i=1,...,n que permitan cuantificar el número de ejemplos no separables que se está dispuesto a admitir, de esta manera la nueva condición es:

$$y_i(< w, x_i > +b) \geq 1 - \delta_i, \quad i = 1, ..., n \tag{19}$$

δ_i representa la desviación del caso separable desde el borden del margen que corresponde a la clase y_1 para el ejemplo (x_i, y_i). Así, se tienen los siguientes casos para las variables de holgura.

- $\delta_i = 0$, el ejemplo es separable.
- $0 < \delta_i \leq 1$, ejemplos no separables y mal clasificados.

Por tanto, la suma de todas las variables de holgura permite medir el coste asociado al número de ejemplos no separables, cuanto mayor sea el valor de la suma, mayor será el número de ejemplos no separables. Así, no basta con maximizar el margen como se hacía en el caso de ejemplos separables linealmente, y por tanto, la nueva función objetivo deberá incluir estos errores o variables de holgura.

$$f(w, \delta) = \frac{1}{2}||w||^2 + C\sum_{i=1}^{n} \delta_i \tag{20}$$

Donde C es denominado *parámetro de regularización* y su valor es elegido por el usuario, y permite regular el compromiso entre el grado de sobreajuste del clasificador lineal y la proporción del número de ejemplos mal clasificados. Finalmente, el nuevo problema de optimización consiste en encontrar el hiperplano definido por w y b que minimiza (20) con las restricciones dadas en (19), es decir

$$\begin{aligned} & minimizar \quad \frac{1}{2} < w, w > +C\sum_{i=1}^{n} \delta_i \\ s.a \quad & y_i(< w, x_i > +b) + \delta_i - 1 \geq 0 \\ & \delta_i \geq 0, \quad i = 1, ..., n \end{aligned} \tag{21}$$

Al igual que en el caso de ejemplos separables linealmente, se usa la forma dual del problema para resolverlo y el hiperplano se separación óptima se obtendrá de una forma análoga a la anterior.

2.3. Máquinas de vector soporte para clasificación binaria de ejemplos no separables linealmente

Como se ha mostrado anteriormente, el hiperplano separador es un buen clasificador cuando el conjunto de datos es separable o cuasi-separable linealmente, sin embargo, este hiperplano no es directamente aplicable cuando el conjunto de datos a clasificar no es separable linealmente. Para esta tarea, se hace uso de un conjunto de funciones base, no lineales, para definir espacios transformados de alta dimensionalidad, y se describirá como encontrar los hiperplanos de separación óptimos en dichos espacios. A este nuevo espacio se le denominara espacio de características, $\mathcal{F}$, con el fin de diferenciarlo del espacio original donde se encuentra el conjunto de datos de entrada.

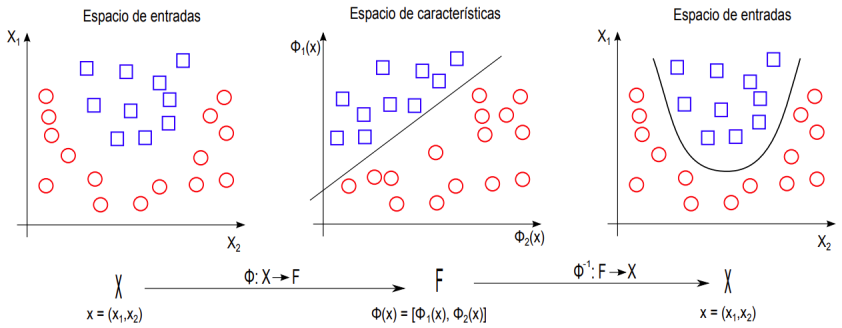

Figura 3: Búsqueda de una función de decisión no lineal en el espacio del conjunto de datos original (espacio de entradas), se puede transformar en una búsqueda de una función de decisión lineal (hiperplano) en un nuevo espacio transformado (espacio de características), y este hiperplano se transforma en una función de decisión no lineal en el espacio original.

Sea $\Phi : X \to \mathcal{F}$ una función de transformación que le da a cada vector de entrada un punto en el espacio de características $\mathcal{F}$ donde $\Phi(x) = [\phi_1(x), ..., \phi_m(x)]$ y $\exists \phi_i(x), i = 1, ..., m$, tal que $\phi_i(x)$ es una función no lineal. El objetivo es construir un hiperplano de separación lineal en este nuevo espacio, y luego la frontera de decisión lineal obtenida en el espacio de características se transformará en una frontera de decisión lineal para el espacio original de entradas.
En este contexto, la función de decisión (1) en el espacio de características está dada por:

$$D(x) = (w_1\phi(x) + ... + w_m\phi(x)) =< w, \Phi(x) > \tag{22}$$

y, su forma dual, la función de decisión se obtiene transformando la expresión de la frontera de decisión (14) en

$$D(x) = \sum_{i=1}^{n} \alpha_i^* y_i < \Phi(x), \Phi(x_i) > \tag{23}$$

A continuación, se ve como calcular el producto escalar $< \Phi(x), \Phi(x_i) >$ haciendo uso de lo que se

denomina *función kernel*.

2.3.1. Función kernel

Por definición, una función kernel es una función K: X × X → ℝ que asigna a cada par de elementos del espacio de entrada 𝔽, X, un valor real correspondiente al producto escalar de las imágenes de dichos elementos en un nuevo espacio 𝔽′, es decir:

$$\begin{aligned} K(x,x') &=< \Phi(x), \Phi(x_i) > \\ &= \Phi_1(x)\Phi_1(x_i) + ... + \Phi_m(x)\Phi_m(x_i) \end{aligned} \tag{24}$$

Por tanto, se puede reemplazar en la ecuación (23), obteniendo la expresión:

$$D(x) = \sum_{i=1}^{n} \alpha_i^* y_i K(x,x') \tag{25}$$

Así, el problema dual de la ecuación (21) que se podrá resolver encontrando los parámetros $\alpha_i, i = 1, ..., n$ es:

$$\begin{aligned} max: & \sum_{i=1}^{n} \alpha_i - \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \\ s.a: & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i, \ i = 1, \ ... \ , n \end{aligned} \tag{26}$$

Como ejemplo, se tiene lo siguiente. Suponga que se tienen elementos de un espacio 𝔽 de dimensión 2. Y el conjunto de funciones base a tener en cuenta es de cardinalidad n, por tanto, la dimensión del espacio 𝔽′ es n.Estas funciones no son necesariamente lineales, y 𝔽′ no tiene tampoco la misma dimensión del espacio de entrada. Entonces, el hiperplano separador será una frontera lineal en el espacio F', pero tendrá otra forma en F, esto dependerá de cómo sean las funciones base.

Note que, dependiendo de cómo sean las funciones base, se tiene un aumento en la dimensionalidad del espacio F', por ejemplo, si se quiere que F' sean los polinomios de grado menor o igual a 3, se tiene una base de tamaño 10 (1, x_1, x_2, x_1x_2, x_1^2, x_2^2, $x_1^2x_2$, $x_1x_2^2$, x_1^3, x_2^3).

Llevando esto al límite, **¿qué hacer con los espacios de entrada de dimensión infinita?** El matemático polaco Nachman Aronszajn, creo el siguiente teorema, el cual es importante para tener en cuenta las funciones Kernel y simplificar el problema que se tiene:

Teorema:

Para cualquier función K: $\mathbb{X} \times \mathbb{X} -> \mathbb{R}$, que sea simétrica y semidefinida positiva se tiene que existe un espacio de Hilbert y una función $\phi(x) : \mathbb{X} -> \mathbb{R}$ tales que:

$$K(x, x') =< \phi(x), \phi(x') >, \ \forall x, x' \in \mathbb{X}$$

Lo importante de este teorema, son las consecuencias que tiene al ser aplicado. Esta afirmación quiere decir que para resolver el problema dual, no será necesario tener en cuenta la dimensión del espacio de entradas $\mathbb{F}$, la dimensión del espacio de características $\mathbb{F}'$, y tampoco las coordenadas de los ejemplos de las categorías a separar, sino que solo es necesario conocer la forma del Kernel como tal, sin importar que esté sobre un espacio de dimensión infinita.
A modo de ejemplo, se tienen las siguientes funciones Kernel:

1. **Kernel lineal:** $K_L(x, x') =< x, x' >$
2. **Kernel polinómico:** $K_p(x, x') = [\gamma < x, x' > +\tau]^p, \ \gamma > 0$
3. **Kernel Gaussiano:** $K_G(x, x') = e^{-\gamma ||x - x'||}, \ \gamma > 0$
4. **Kernel sigmoidal:** $K_S(x, x') = tanh(\gamma < x, x' > +\tau)$

3. Máquinas de vector soporte para la clasificación del cáncer de mama

3.1. Recolección de los datos

Se uso una base de datos obtenida de [6], la cual está relacionada con el diagnóstico del cáncer de mama en un estudio de la Universidad de Wisconsin en Estados Unidos. Esta base de datos surgió de una investigación del Dr. Wolberg para el diagnóstico del cáncer de mama únicamente en una aspiración con aguja fina, de esta aspiración se extraen ciertas características que son de relevancia para determinar o clasificar si la masa cancerígena es benigna o maligna; el proceso para la recolección básicamente consiste en que se toma una aspiración con aguja fina de la masa mamaria y luego se monta a una platina de muestras del microscopio en donde se retiñe para resaltar los núcleos celulares, para luego aislarlos con ayuda del software Xcyt (Un software usado para el diagnóstico y pronóstico citológico a distancia para el cáncer de mama) y una vez que estos núcleos han sido aislados, el programa calcula los valores para las 10 características que se pueden extraer de cada núcleo (radio, perímetro, área, textura, lisura, concavidad, puntos cóncavos, simetría, compacidad y dimensión fractal); se calculan la media, error estándar y valores extremos de estas características de los núcleos, obteniendo así 30 características nucleares para cada muestra. A continuación se muestra el resultado de una aspiración con aguja fina y su resultado al procesarla en el software Xcyt.

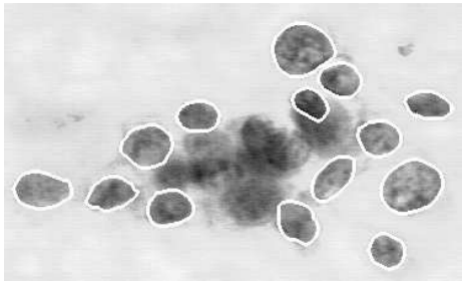

Figura 4: Muestra de aspiración con aguja fina capturada en microscopio y procesada en Xcyt

Además se incluye el ID y el diagnóstico del paciente (masa benigna o maligna), por lo que la base de datos consiste de 569 filas (muestras)×32 columnas (características).

De esta manera, el objetivo es desarrollar una máquina de vector soporte que a partir del 80 % de los datos de este conjunto elegidos de forma aleatoria se pueda entrenar con el fin de clasificar si un tumor es maligno o benigno basado en las características mencionadas anteriormente, obtenidas a partir de la aspiración con aguja fina. Luego, se testea con el 20 % restante de los datos obteniendo ciertas medidas respecto a la precisión de la SVM.

Para simplificar el procedimiento, primero se hizo un tratamiento a los datos para disminuir su dimensionalidad a través de análisis de componentes principales. De esta manera, se proyectan los datos en las direcciones de las dos máximas variabilidades, correspondientes a los valores extremos de la dimensión fractal y la simetría, las cuales explican el 99.82 % de la variabilidad de los datos iniciales.

3.2. Resultados

A continuación se presenta el desarrollo, resultados y complejidad que trae consigo la máquina de vector soporte usando distintas funciones kernel, para luego determinar qué tipo de función Kernel permite obtener mejores resultados en cuánto a medidas de efectividad y complejidad computacional.

Para visualizar los resultados de la máquina de vector soporte, se hicieron 200 repeticiones del entrenamiento de esta, y en cada una, como fue mencionado anteriormente, se entrenó con el 80 % de los datos, y el restante 20 % fue usado como prueba para verificar la efectividad del algoritmo. En adición, con el fin de obtener buenos resultados, el 80 % de los datos seleccionados para el entrenamiento son seleccionados aleatoriamente.

Para medir el desempeño del modelo, se calculan cuatro métricas distintas, la proporción de predicciones correctas (accuracy), la precisión, la sensibilidad o recall y el f1 score, las cuales se

calculan de la siguiente manera:

$$\begin{aligned}
\text{Accuracy} &= \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \\
\text{Precisión} &= \frac{\text{TP}}{\text{TP} + \text{FP}} \\
\text{Sensibilidad} &= \frac{\text{TP}}{\text{TP} + \text{FN}} \\
\text{F1 score} &= \frac{2(\text{Precisión} * \text{Sensibilidad})}{\text{Precisión} + \text{Sensibilidad}}
\end{aligned}$$

Donde

- **TP**: Verdaderos positivos, el modelo predice correctamente positivo
- **FP**: Falsos positivos, el modelo predice incorrectamente positivo
- **TN**: Verdaderos negativos, el modelo predice correctamente negativo
- **FN**: Falsos negativos, el modelo predice incorrectamente negativo

Luego, cada métrica mide algo distinto sobre la prueba del modelo. La proporción de predicciones correctas indica que porcentaje de los datos de prueba fueron predecidos correctamente, la precisión mide que porcentaje de los predecidos como positivos son realmente positivos, la sensibilidad mide que porcentaje de los positivos fueron predecidos correctamente y finalmente, el f1 score da una puntuación que representa la precisión y sensibilidad, por medio de la media armónica entre estos dos. Adicional a esto, también se encuentra la matriz de confusión, la cual da información sobre la cantidad de predicciones en los datos de prueba con respecto a la clase real, esto es, da información sobre los TP, FP, TN y FN.

Para el conjunto de datos, consideramos que el cáncer es maligno como un caso positivo y benigno lo contrario, y para ver el resultado del modelo, se calculan las métricas mencionadas anteriormente en cada repetición, y luego se calcula el promedio de estas para las 200 repeticiones, y para la matriz de confusión, se consideran todas las matrices de confusión de cada repetición.

- **Kernel lineal.** Los resultados al utilizar una función kernel lineal son los siguientes:

$$\begin{aligned}
\text{Accuracy} &= 91{,}68\,\% \\
\text{Precisión} &= 90{,}80\,\% \\
\text{Sensibilidad} &= 86{,}31\,\% \\
\text{F1 score} &= 88{,}49\,\%
\end{aligned}$$

De lo cual se puede concluir que en general, el modelo predice correctamente el 91.68 % de las veces, de los casos que predice positivos el 90.8 % de las veces es correcto, de los casos que son positivos el 86.31 % de las veces los predice correctamente y finalmente, el f1 score es de 88.49 %. Además, la matriz de confusión obtenida es:

		Predicted class	
		M	**B**
True Class	**M**	7283	1139
	B	759	13619

Tabla 1: Matriz de confusión total usando una función kernel lineal

A partir de esta, se puede decir que hay un total de 7283 verdaderos positivos, 13619 verdaderos negativos, 759 falsos positivos y 1139 falsos negativos. Finalmente se tomó el tiempo computacional que tarda este algoritmo, usando la función *tic toc* de MATLAB, así, el tiempo total necesario para este tipo de máquina de vector soporte es de **601.822 segundos**. Además el código asociado a esta máquina de vector soporte se encuentra en el archivo adjunto *svmLineal.m*

- **Kernel polinómico.** Los resultados al utilizar una función kernel polinómica son los siguientes:

$$\begin{aligned} \text{Accuracy} &= 62{,}5\,\% \\ \text{Precisión} &= \text{Indefinida} \\ \text{Sensibilidad} &= 0\,\% \\ \text{F1 score} &= \text{Indefinida} \end{aligned}$$

Además, la matriz de confusión obtenida es:

		Predicted class	
		M	**B**
True Class	**M**	0	8550
	B	0	14250

Tabla 2: Matriz de confusión total usando una función kernel polinómico

A partir de esta, se puede decir que hay un total de 0 verdaderos positivos, 14250 verdaderos negativos, 0 falsos positivos y 8550 falsos negativos, como es posible observar, hacer uso de esta función kernel no tiene sentido ya que simplemente predice todo en una única clase, por esta razón la precisión es indefinida y así mismo, el f1 score. Finalmente se tomó el tiempo computacional que tarda este algoritmo, usando la función *tic toc* de MATLAB, así, el tiempo total necesario para este tipo de máquina de vector soporte es de **1180.564 segundos**. Además el código asociado a esta máquina de vector soporte se encuentra en el archivo adjunto *svmPolinomial.m*

- **Kernel gaussiano.** Los resultados al utilizar una función kernel gaussiana son los siguientes:

$$\begin{aligned} \text{Accuracy} &= 63{,}06\,\% \\ \text{Precisión} &= \text{Indefinida} \\ \text{Sensibilidad} &= 0\,\% \\ \text{F1 score} &= \text{Indefinida} \end{aligned}$$

Además, la matriz de confusión obtenida es:

		Predicted class	
		M	**B**
True Class	**M**	0	8422
	B	0	14378

Tabla 3: Matriz de confusión total usando una función kernel gaussiano

A partir de esta, se puede decir que hay un total de 0 verdaderos positivos, 14378 verdaderos negativos, 0 falsos positivos y 8422 falsos negativos, como es posible observar, hacer uso de esta función kernel no tiene sentido ya que simplemente predice todo en una única clase, por esta razón la precisión es indefinida y así mismo, el f1 score. Finalmente se tomó el tiempo computacional que tarda este algoritmo, usando la función *tic toc* de MATLAB, así, el tiempo total necesario para este tipo de máquina de vector soporte es de **4.494 segundos**. Además el código asociado a esta máquina de vector soporte se encuentra en el archivo adjunto *svmGaussiano.m*

Sin embargo, existe una diferencia entre los falsos positivos y los falsos negativos, ya que se puede decir que el error de un falso negativo es peor o más grave, ya que es preferible decir que un paciente tiene un tumor maligno cuando en realidad es benigno en comparación a concluir que un paciente tiene un tumor benigno cuando en realidad es maligno. Por tanto, lo ideal sería

disminuir la cantidad de falsos negativos, lo cual se ve reflejado en la sensibilidad, sin embargo, hacer esto implica aumentar el error de los falsos positivos, que es equivalente a disminuir la precisión.

Para tratar este problema, se agrega una penalidad a los falsos negativos, es decir, es más 'costoso' predecir un falso negativo a un falso positivo, y se entrena nuevamente la máquina de vector soporte con un procedimiento similar al anterior, y los resultados fueron los siguientes:

- **Kernel lineal con penalización.** Los resultados al utilizar una función kernel lineal con la penalización descrita anteriormente son los siguientes:

$$\begin{aligned} \text{Accuracy} &= 91{,}54\,\% \\ \text{Precisión} &= 85{,}99\,\% \\ \text{Sensibilidad} &= 92{,}86\,\% \\ \text{F1 score} &= 89{,}29\,\% \end{aligned}$$

  De lo cual se puede concluir que en general, el modelo predice correctamente el 91.54 % de las veces, de los casos que predice positivos el 85.99 % de las veces es correcto, de los casos que son positivos el 92.86 % de las veces los predice correctamente y finalmente, el f1 score es de 89.29 %. Además, la matriz de confusión obtenida es:

		Predicted class	
		M	**B**
True Class	**M**	7799	612
	B	1318	13071

Tabla 4: Matriz de confusión total usando una función kernel lineal con penalización

A partir de esta, podemos decir que hay un total de 7799 verdaderos positivos, 13071 verdaderos negativos, 1318 falsos positivos y 612 falsos negativos. Es posible evidenciar que la cantidad de falsos negativos son menos que en el caso anterior, y esto se ve reflejado en una sensibilidad mejor, sin embargo, como fue mencionado, la cantidad de falsos positivos es mayor y por tanto la precisión es menor. Pero la proporción de predicciones correctas y el f1 score son relativamente similares, y por tanto, para este ejemplo, es preferible este modelo con penalización.

Finalmente se tomó el tiempo computacional que tarda este algoritmo, usando la función *tic toc* de MATLAB, así, el tiempo total necesario para este tipo de máquina de vector soporte es de **582.024 segundos**. Además el código asociado a esta máquina de vector soporte se encuentra en el archivo adjunto *svmLinealPenalizado.m*.

- **Kernel polinomial con penalización.** Los resultados al utilizar una función kernel polinomial con la penalización descrita anteriormente son los siguientes:

$$\begin{aligned} \text{Accuracy} &= 63{,}25\,\% \\ \text{Precisión} &= \text{Indefinido} \\ \text{Sensibilidad} &= 0\,\% \\ \text{F1 score} &= \text{Indefinido}\,\% \end{aligned}$$

Además, la matriz de confusión obtenida es:

		Predicted class	
		M	**B**
True Class	**M**	0	8380
	B	0	14420

Tabla 5: Matriz de confusión total usando una función kernel polinomial con penalización

A partir de esta, podemos decir que hay un total de 0 verdaderos positivos, 14420 verdaderos negativos, 0 falsos positivos y 8380 falsos negativos, como es posible observar, hacer uso de esta función kernel no tiene sentido ya que simplemente predice todo en una única clase, por esta razón la precisión es indefinida y así mismo, el f1 score, igual que en el caso de la función kernel polinomial sin penalización.

Finalmente se tomó el tiempo computacional que tarda este algoritmo, usando la función *tic toc* de MATLAB, así, el tiempo total necesario para este tipo de máquina de vector soporte es de **1194.965 segundos**. Además el código asociado a esta máquina de vector soporte se encuentra en el archivo adjunto *svmPolinomialPenalizado.m*.

- **Kernel gaussiano con penalización.** Los resultados al utilizar una función kernel gaussiano con la penalización descrita anteriormente son los siguientes:

$$\begin{aligned} \text{Accuracy} &= 43{,}19\,\% \\ \text{Precisión} &= 36{,}14\,\% \\ \text{Sensibilidad} &= 67{,}24\,\% \\ \text{F1 score} &= 46{,}96\,\% \end{aligned}$$

De lo cual se puede concluir que en general, el modelo predice correctamente el 43.19 % de las veces, de los casos que predice positivos el 36.14 % de las veces es correcto, de los casos

que son positivos el 67.24 % de las veces los predice correctamente y finalmente, el f1 score es de 46.96 %. Además, la matriz de confusión obtenida es:

		Predicted class	
		M	**B**
True Class	**M**	5988	2526
	B	10427	3859

Tabla 6: Matriz de confusión total usando una función kernel gaussiano con penalización

A partir de esta, podemos decir que hay un total de 5988 verdaderos positivos, 3859 verdaderos negativos, 10427 falsos positivos y 2526 falsos negativos. Es posible evidenciar qué tanto la matriz de confusión como las demás medidas de efectividad presentan unos mejores resultados respecto a la máquina de vector soporte con un kernel gaussiano sin penalización, por tanto, para este ejemplo, es preferible este modelo con penalización.

Finalmente se tomó el tiempo computacional que tarda este algoritmo, usando la función *tic toc* de MATLAB, así, el tiempo total necesario para este tipo de máquina de vector soporte es de **3.116 segundos**. Además el código asociado a esta máquina de vector soporte se encuentra en el archivo adjunto *svmGaussianoPenalizado.m*.

A partir de estos resultados se puede ver que al realizar la penalización se presentan unos mejores resultados y así mismo, usar un kernel lineal o un kernel gaussiano representan unas mejores medidas de desempeño del modelo respecto al kernel polinomial.

4. Aporte

Como un aporte al tema, se propone hacer la máquina de vector soporte pero en lugar de utilizar un único kernel, se hace una combinación convexa de dos distintos kernels. Para este ejemplo, se uso el kernel lineal, con el motivo de comparación, y el gaussiano o de base radial, ya que fue el mejor después del lineal. Aclarando que usar el polinomial no tendría sentido ya que el lineal es un caso particular de este (cuando el grado es 1), y por tanto hacer una combinación convexa de estos es equivalente a usar solo el polinomial.

Para obtener que tanta parte o porcentaje de cada kernel se debe usar, el cual corresponde al parámetro de la combinación convexa, se hizo uso de simulación Montecarlo, así se obtiene que tener aproximadamente un 75 % del kernel lineal y un 25 % del kernel de base radial da buenos

resultados; claramente como esto se hizo generando números aleatorios, este valor no es necesariamente el mejor. Además este proceso de búsqueda del parámetro α de la combinación convexa tarda **329.52 segundos**. Este método se encuentra en el archivo adjunto *SimulacionMontecarlo.m*.

Haciendo el mismo procedimiento realizado anteriormente, se obtuviene que, para una máquina de vector soporte usando una combinación convexa de dos kernels, el resultado es

$$\begin{aligned} \text{Accuracy} &= 92{,}39\,\% \\ \text{Precisión} &= 92{,}36\,\% \\ \text{Sensibilidad} &= 86{,}93\,\% \\ \text{F1 score} &= 89{,}56\,\% \end{aligned}$$

Y la correspondiente matriz de confusión es

		Predicted class	
		M	**B**
True Class	**M**	7340	1105
	B	630	13725

Tabla 7: Matriz de confusión total usando una combinación convexa de kernels

Se puede ver que estos resultados son un poco mejor que el modelo usando únicamente el kernel lineal, ya que este tiene un poco mejor la proporción de predicciones correctas, aproximadamente un 7 % mejor precisión, una sensibilidad un poco menor y un f1 score aproximadamente 0.25 % mejor. Finalmente se toma el tiempo computacional que tarda este procedimiento con la funcion *tic toc* de MATLAB y se encuentra que el modelo tarda **621.047 segundos**, además el código asociado a este proceso se encuentra en el archivo adjunto *svmConvexa.m*.

Igual a lo anterior, también se prueba agregando la penalización a los falsos negativos, para disminuir este tipo de error y los resultados obtenidos son los siguientes:

$$\begin{aligned} \text{Accuracy} &= 91{,}59\,\% \\ \text{Precisión} &= 85{,}35\,\% \\ \text{Sensibilidad} &= 93{,}12\,\% \\ \text{F1 score} &= 89{,}07\,\% \end{aligned}$$

Y la correspondiente matriz de confusión es

		Predicted class	
		M	**B**
True Class	**M**	7867	566
	B	1352	13015

Tabla 8: Matriz de confusión total usando una combinación convexa de funciones kernels con penalización

Se puede ver que, igual que con el kernel lineal, la cantidad de falsos negativos disminuyó considerablemente, y en comparación con el kernel lineal con penalización, este predice una menor cantidad de falsos negativos, siendo un poco mejor que usar únicamente el kernel lineal. Así mismo el tiempo computacional empleado es de **571.969 segundos** y el código asociado a este proceso se encuentra en el archivo adjunto *svmConvexaPenalizado.m*.

5. Conclusiones

Se puede ver que las máquinas de vector soporte son una herramienta poderosa del aprendizaje automático para la clasificación de grupos de datos, además es posible observar que la diferencia entre los resultados al usar solo un kernel y al usar una combinación convexa de kernels no es demasiado significativa en este caso, sin embargo, la combinación convexa es un poco mejor. Sin embargo, este es solamente un caso donde el kernel lineal se ajusta bastante bien, entonces es posible que en otros casos o conjuntos de datos usar un único kernel y usar una combinación convexa de dos kernels tenga una mayor diferencia, y pueda resultar mucho mejor usar una combinación convexa de kernels. De esta manera, se pueden realizar varias combinaciones convexas entre funciones kernel teniendo en cuenta los mejores resultados en cuanto a medidas de precisión, exactitud y sensibilidad; por lo que dependiendo del conjunto de datos se deben decidir dichas funciones kernel para generar la combinación convexa, además, también dependiendo del contexto se querrá reducir alguno u otro error sobre el otro. Finalmente, también se debe tener en cuenta la complejidad computacional, es decir, el usuario debe decidir si prefiere aumentar un poco la precisión, exactitud y sensibilidad o prefiere un menor costo computacional.

Además, de los resultados obtenidos en este trabajo, se concluye que generalmente al reducir la dimensión de los datos se genera un buen resultado en cuanto a precisión, exactitud y sensibilidad del algoritmo; por lo que no sería necesario calcular todas las características en el software Xcyt a partir de los núcleos obtenidos de la aspiración con aguja fina, ya que esto implica un mayor esfuerzo computacional y operacional, por tanto, recomendamos en este caso particular de la clasificación del cáncer de mama extraer únicamente la información correspondiente a los valores extremos de la dimensión fractal y la simetría.

Finalmente, el algoritmo tiene buenos resultados para la clasificación del cáncer de mama, ya que los resultados en cuanto a medidas de precisión, exactitud y sensibilidad por lo general están muy próximas al 90 %. Como posible trabajo futuro se recomienda usar otro criterio para encontrar el parámetro α de la combinación convexa, ya que la simulación Montecarlo implica un alto costo computacional, una alternativa podría ser usar un cociente entre las medidas de f1 score de los kernels a combinar y observar si este arroja unos mejores resultados respecto a la simulación Montecarlo.

Referencias

[1] "Descubre los principales beneficios del 'Machine Learning'",*Iberdrola*. [En línea]. Disponible en: https://www.iberdrola.com/innovacion/machine-learning-aprendizaje-automatico

[2] "Support Vector Machine (SVM)", MathWorks. [En línea]. Disponible en: https://la.mathworks.com/discovery/support-vector-machine.html: :text=Las %20funciones %20de %20kernel %20asignan,lineales % 20en %20el %20espacio %20dimensional

[3] C.A. Villaseñor Padilla, *IA2.11 Máquinas de Soporte Vectorial.*[Vídeo de YouTube].Guadalajara: Universidad de Guadalajara,2020. Disponible en: https://www.youtube.com/watch?v=0N_7s1U6isE

[4] E.J. Carmona,"Tutorial sobre Máquinas de Vectores Soporte (SVM)", Universidad Nacional de Educación a Distancia, Madrid, España. nov. 2016

[5] E. Campo León, Ïntroducción a las máquinas de vector soporte (SVM) en aprendizaje supervisado", Trabajo de fin de grado, Univ. Zaragoza, 2016.

[6] Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[7] J. Millán, B. Robles, "Modelo en Machine Learning para el diagnóstico del cáncer de mama", Trabajo de fin de grado, Univ. Distrital Francisco José de Caldas, 2020.