UNIVERSIDAD AUTÓNOMA METROPOLITANA.
UNIDAD IZTAPALAPA.

División de Ciencias Básicas e Ingeniería.
Posgrado en Ciencias y Tecnologías de la Información.

Discriminación de datos en máquinas de soporte vectorial.

Presenta: González Torres Guillermo.

Tesis para obtener el grado de maestro en Ciencias y Tecnologías de la información.

Asesor: Dr. René Mac Kinney Romero

Jurado calificador:

Presidente:	Dra. Alicia Morales Reyes.
Secretario:	Dr. René Mac Kinney Romero.
Vocal:	Dr. Oscar Yañez Suárez.

México, CDMX, enero del 2018.

Índice general

Resumen

El aprendizaje maquinal (ML por sus siglas en inglés) es un campo de la computación que permite a las computadoras descubrir patrones entre muchos ejemplos de un fenómeno, por lo que pueden actuar ante situaciones que no fueron programadas de manera explicita, gracias a esto se desarrollaron sistemas con comportamientos autónomos basados en experiencias. El ML ha demostrado ser muy eficiente aplicado al análisis de información, en particular en el campo de la clasificación de datos. Se ha hecho evidente que el ML realiza el análisis de grandes conjuntos de datos mucho más rápido que su contraparte humana lo que permite manejar cantidades de información que en un principio eran intratables. Dentro de ML existe una gran cantidad de técnicas desarrolladas con el fin de simular la inteligencia humana al resolver problemas, estos van desde toma de decisiones dentro de un juego, clasificaciones de datos, hasta robots autónomos que permiten interactuar con el medio ambiente para poder desplazarse de manera independiente.

La clasificación de datos ha sido uno de los problemas a los que más se han enfocado los métodos de aprendizaje maquinal, gracias a la gran cantidad de aplicaciones en las que es necesario tener herramientas autónomas que reconozcan y separen tipos de datos, esta clasificación debe de ser precisa a la vez que rápida, por lo que se requieren de métodos que realicen la tarea en tiempos tratables para manejar la información. Existen muchas técnicas que resuelven esta problemática que sin embargo necesitan de más investigación para mejorar su rendimiento, entre las técnicas más destacadas se encuentran: Los árboles de decisión, las redes bayesianas, los k-vecinos más cercanos, etc.

Uno de los métodos más destacados son las máquinas de soporte vectorial [1] (*SVM* por sus siglas en inglés) ya que tienen buenos resultados y son muy generalizados, esto lo logra creando modelos a partir de un conjunto de aprendizaje con los que es capaz de clasificar los datos dependiendo de sus atributos. Las *SVM* al igual que otras técnicas de aprendizaje maquinal tienen una fase de entrenamiento que llega a ser muy lenta cuando la cantidad de datos con las que crea los modelos de clasificación son muy grandes, por este motivo los tiempos de entrenamiento llegan a ser intratables usando equipo de cómputo convencional para ejemplos con bases de datos que contengan millones de registros, algunas técnicas en el aprendizaje maquinal tienden a estar en la escala de minutos u horas para una base de datos con esta cantidad de información, mientras que el tiempo requerido por las máquinas de soporte vectorial sobrepasan las horas o incluso días de procesamiento, donde también se tiene que considerar la gran cantidad de recursos de cómputo que requiere aplicar la técnica.

En este trabajo proponemos un método para filtrar los datos de entrenamiento de las máquinas de soporte vectorial, de esta manera utilizando un conjunto más pequeño las *SVM* entrenarán más rápido manteniendo su precisión al clasificar.

Capítulo 1

Introducción

Los avances tecnológicos de las últimas décadas han incrementado en gran medida la capacidad de almacenamiento gracias al aumento en el espacio de memoria en los equipos y su reducción de precios, lo que origina que la cantidad de datos digitales crezca y su manejo ocupe de mucho tiempo y esfuerzo. Esto ha generado la necesidad de crear métodos que permitan manejar los datos de manera rápida y precisa.

Un ejemplo de esto es analizar una base de datos en un centro de salud que atiende a miles de individuos, estos datos contienen la información médica del paciente obtenida de la exploración física y estudios complementarios, con los que pueden ser inferidos diagnósticos clínicos y pronósticos de enfermedades. Gracias a los datos obtenidos de cada individuo es posible hacer análisis en donde podemos predecir las posibles enfermedades a las que es susceptible un individuo y los tratamientos pertinentes que necesita.

El trabajo de revisar el expediente de miles de personas y analizar sus antecedentes médicos es una labor extremadamente lenta para una persona, lo que ha provocado que se desarrollen una gran cantidad de técnicas que permiten hacer el análisis de información de manera más rápida. El ejemplo anterior puede ser considerado un problema de clasificación donde se necesita conocer a que clase pertenece un objeto en base a sus atributos, es decir, que en base a sus niveles metabólicos e indices corporales se puede conocer el tipo de tratamiento que requiere el paciente. Los métodos para resolver estos problemas son muy variados y pueden estar basados en diversas disciplinas como son los análisis estadísticos, técnicas de la inteligencia artificial, etc.

Los métodos de aprendizaje maquinal que realizan el análisis de la información para su clasificación han demostrado tener un buen desempeño en la aplicación de problemas reales donde se ingresan datos de entrenamiento a una computadora para que analice y sea capaz de clasificar, o como en el ejemplo anterior, toman decisiones sobre el tratamiento más conveniente para los pacientes.

Existen una gran cantidad de algoritmos desarrollados con el fin de clasificar datos, cada uno de estos algoritmos tiene ventajas y desventajas con respecto a los demás. A continuación se listan algunos ejemplos de los métodos mas utilizados:

- Árboles de decisión: Los Árboles de decisión son modelos representados por esquemas de árbol que permiten hacer una clasificación. Esto lo logran construyendo diagramas que cuentan con nodos raíz, nodos intermedios, nodos hijo y conexiones entre ellos. El nodo raíz indica donde se inicia la clasificación, los nodos intermedios indican alguna característica especifica del dato, los nodos hijos son la clasificación final y las conexiones indican el tipo de atributo con el que cuenta el dato que se requiere clasificar.

- Vecinos más cercanos: Esta técnica esta basada en comparar los atributos de un dato que se desea clasificar con los atributos de un conjunto de datos que ya están clasificados. Para decidir a que clase pertenece un dato se utiliza un parámetro k que indica la cantidad de datos que son más parecidos (por la semejanza de sus atributos) al que se requiere clasificar y se asignará dicha clase.

- Redes bayesianas: Son grafos dirigidos que representan un conjunto de probabilidades condicionales que permiten hacer inferencias sobre una situación particular.

- Redes neuronales: La idea fundamental es programar un conjunto de perceptrones o neuronas artificiales que se entrenan para lograr hacer una clasificación adecuada de los datos dependiendo de las características de entrada que reciba. Cada perceptron es un algoritmo que recibe un vector de valores como entrada y dependiendo de ésta regresa una salida. En una red neuronal se conectan un conjunto de perceptrones donde la salida de cada uno puede ser la entrada de otro, en la etapa de entrenamiento se ajustan las salidas de los perceptrones para que dependiendo de una entrada especifica (en este caso los atributos de un dato) en la red neuronal la salida sea la clasificación adecuada de los datos.

- Máquinas de soporte vectorial o (*Support Vector Machine SVM*) Son técnicas parecidas a las redes neuronales que en la fase de entrenamiento recibe vectores que representan los atributos y clasificaciones de datos y con herramientas matemáticas crean hiperplanos que separán los datos en base a su posición dentro del plano dimensional en el que se encuentran.

Dentro de esta investigación nos enfocamos en las *SVM* ya que su campo de aplicación es muy generalizado en la clasificación de datos, esto significa que puede ser aplicado en un gran conjunto de problemas donde el tipo de clases y atributos que contengan los datos pueda ser representado numéricamente. También ha demostrado tener muy buen desempeño en la vida real cuando clasifica diferentes tipos de datos. Como un caso particular se utiliza en el reconocimiento de características biométricas como son el reconocimiento facial o de huella digital.

Aunque las ventajas de utilizar *SVM* son muchas, el tiempo de entrenamiento que necesita para generar los modelos de clasificación crece demasiado cuando el conjunto de entrenamiento sobrepasa los miles, por este motivo se han investigado diversos métodos que permitan agilizar la fase de entrenamiento, para que los tiempos de procesado en la construcción del modelo de clasificación sean menores aunque los datos de entrenamiento sobrepasen los millones.

La tesis esta basada en el hecho de que los datos utilizados en la fase de entrenamiento pueden ser filtrados de manera que con un conjunto de entrenamiento menor obtendría los mismos resultados para la clasificación, por lo que muchos de ellos pueden ser discriminados para solo tomar en cuenta datos que efectivamente son suficientes para que la *SVM* construya un modelo que clasifique los datos de manera adecuada sin perder la precisión.

Este trabajo esta organizado de la siguiente manera, en el capitulo 2 comenzamos con una explicación de la inteligencia artificial así como algunos métodos utilizados para la clasificación de datos, en el capitulo 3 se describirán algunos métodos que se han desarrollado para reducir los tiempos de entrenamiento de las SVM en su aplicación, en el capitulo 4 se propone un método para disminuir el entrenamiento de las máquinas de soporte vectorial y algunas técnicas necesarias para implementarlo, en el capitulo 5 se presentan experimentos donde se demuestra que el método puede agilizar en gran medida el entrenamiento de las máquinas de soporte vectorial sin perder precisión a la hora de clasificar y dentro capitulo 6 se dan las conclusiones del trabajo y el posible trabajo a futuro.

Capítulo 2

Aprendizaje Maquinal

2.1. Introducción

La inteligencia artificial(IA) y el aprendizaje maquinal son ramas de la computación que tienen la finalidad de que las computadoras se comporten como humanos ante ciertas situaciones. La IA es una disciplina que fue formalmente iniciada en 1956 cuando John McCarthy acuñó el término y definió como: "La ciencia e ingeniería de construir máquinas inteligentes", mientras que el aprendizaje maquinal o automático estudia técnicas que permiten a las computadoras aprender, esto es creando programas que cambien su comportamiento a partir de ejemplos anteriormente proporcionados. Para estas técnicas no existe una sola definición formal ya que existen muchas interpretaciones, a continuación se presentan definiciones que están divididas en cuatro diferentes categorías [2].

1. Sistemas que piensan como humanos
    - El esfuerzo de crear máquinas pensantes máquinas con mente, en el sentido literal de la palabra [3].
    - La automatización de las actividades que se asocian con el razonamiento humano, actividades como la toma de decisiones, resolver problemas, pensar. [4].
2. Sistemas que piensan racionalmente
    - El estudio de facultades mentales a través del uso de modelos computacionales [5].
    - El estudio de la computación que hace posible percibir, razonar y actuar [6].
3. Sistemas que actúan como humanos
    - El arte de crear máquinas que realicen funciones que requieren inteligencia cuando son realizadas por personas [7].
    - El estudio de cómo construir computadoras que realicen tareas que por el momento las personas realizan mejor [8].
4. Sistemas que actúan racionalmente

- Un campo que estudia y busca emular el comportamiento inteligente en términos de procesos computacionales.[9].
- Una rama en las ciencias de computación que se encarga de la automatización de comportamiento inteligente [10].

Estos cuatro diferentes enfoques provocan un debate entre las ideas basadas en emular el comportamiento humano y las basadas en un comportamiento racional, donde el enfoque centrado en humanos es una ciencia empírica que involucra hipótesis y experimentación. Mientras que un enfoque racional es una combinación de matemáticas e ingeniería.

El enfoque considerado en este trabajo es que el ML sintetiza y automatiza tareas intelectuales y es, por lo tanto, particularmente relevante para cualquier ámbito de la actividad intelectual humana. Esto quiere decir que automatizamos el comportamiento que tendría una persona ante cierta situaciones. Situaciones que abarcan desde realizar un movimiento en ajedrez hasta controlar un vehículo sin supervisión humana. El ML tiene una gran cantidad de técnicas que permiten hacer análisis o categorizar las opciones que existen para determinados problemas y cada una de estas técnicas está enfocado a un conjunto de problemas en particular.

Uno de los campos a los que se aplican estas técnicas son los de clasificación donde se requiere ordenar diferentes tipos de datos de forma rápida y precisa, trabajo que necesita una gran cantidad de tiempo y esfuerzo si es realizado por humanos. Por este motivo se han desarrollado muchas técnicas especializadas en realizar esta tarea que pueden dividirse en dos grandes campos: la clasificación supervisada donde se conoce un conjunto de clases a priori y a partir de éstas se realiza la separación, y la clasificación no supervisada donde no se conoce conjuntos o patrones que sirvan como ejemplo para la clasificación por lo que se agrupan entre los datos proporcionados según su semejanza.

Algunas técnicas para la clasificación de datos son:

- Árboles de decisión
- Redes Bayesianas
- K Vecinos más cercanos
- Redes neuronales artificiales
- Máquinas de soporte vectorial

En las siguientes secciones describiremos de manera más profunda el funcionamiento de cada uno de estos métodos, de igual manera sus ventajas y desventajas en la clasificación de datos.

2.2. Árboles de decisión

Los árboles de decisión[11] surgen a partir de la necesidad de modelar funciones discretas, donde el objetivo es determinar un valor que depende de un conjunto de propiedades o variables, partiendo de esto se clasifica o determina la acción a ser tomada. Se nombran árboles de decisión por el hecho

de que se puede asociar a los gráfos conocidos como árboles, donde cada nodo representa un estado en los datos, mientras que las ramas representan los posibles valores que pueden tomar los atributos con las que cuenta el dato en cuestión.

Dicho de otra manera esta herramienta construye un árbol que decide a que clase pertenece el dato junto con todos los atributos que lo describen, esto lo hace una herramienta de clasificación muy eficiente, siempre y cuando el árbol generado tenga dimensiones no tan grandes que sean tratables por las computadoras. La idea clave en los árboles de decisión es construir el árbol más simple que represente todas las combinaciones de atributos necesarias para conocer a que clase pertenece cada dato.

La pregunta inmediata que surge es si es posible determinar la clase de cualquier objeto a partir de los valores de sus atributos, si el conjunto de entrenamiento contiene dos objetos de clases diferentes que compartan los mismos atributos se observa que es imposible clasificar estos objetos por lo que en estos casos los atributos son inadecuados para la tarea de clasificación.

Si los atributos son adecuados siempre es posible construir un árbol de decisión que clasifique correctamente los objetos en el conjunto de entrenamiento y generalmente es factible crear más de un clasificador. La esencia de la inducción es ir mas allá del conjunto de entrenamiento y clasificar de manera correcta no solo los datos que se proporcionan en el conjunto de entrenamiento, también clasificar datos que no han sido considerados en éste. Para realizarlo el árbol debe capturar una relación significativa entre la clase de objeto y los valores de sus atributos.

Las ramificaciones a partir de este nodo representan cada posible valor del atributo y se verifica si es información suficiente para clasificar a partir de un conjunto de datos conocidos, en caso contrario por cada ramificación agregamos nodos donde se especifica el siguiente atributo y sus respectivas ramas comprobando si son los datos suficientes para diferenciar las clases.

Existen muchas heurísticas basada en teoría de la información que tratan de elegir al atributo mas útil para la clasificación, dado que es el que aporta mayor información. Una fundamental es la entropía donde la función más sencilla esta descrita como:

$$-(\frac{|p|}{|d|}) \bullet log_2 \frac{|p|}{|d|} - \frac{|n|}{|d|} \bullet log_2 \frac{|n|}{|d|}$$

Donde p es el conjunto de los ejemplos positivos, n el de los negativos.

Propiedad 1	*Propiedad 2*	*Propiedad 3*	Clasificación
carnívoro	Pelaje corto	vista aguda	felino
no carnívoro	Pelaje corto	sin vista aguda	no felino
carnívoro	Pelaje largo	vista aguda	felino
carnívoro	Pelaje corto	vista aguda	felino
no carnívoro	Pelaje corto	sin vista aguda	no felino
carnívoro	Pelaje corto	sin vista aguda	no felino

Cuadro 2.1: Tabla que muestra propiedades de animales carnívoros y no carnívoros

Con las propiedades mostradas en el cuadro 2.1 y utilizando la función de entropía construimos el árbol mostrado en la figura 2.1.

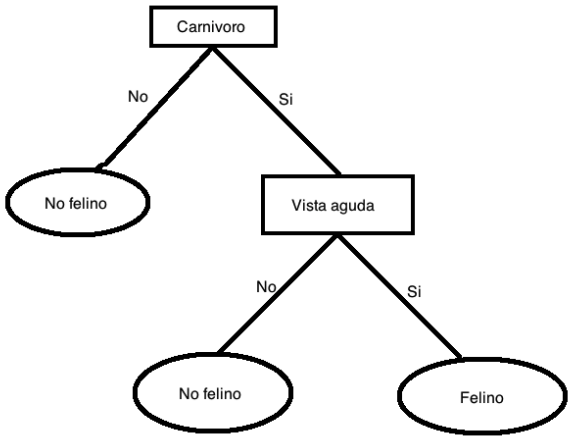


Figura 2.1: árbol de decisión

En este caso cuando conocemos que el animal no es carnívoro ya es un atributo suficiente para saber que el animal no es felino, pero si el animal es carnívoro todavía tenemos que desarrollar el árbol y preguntar si tienen vista aguda, al conocer que el animal es carnívoro y tiene vista aguda lo podemos clasificar como felino en caso contrario no seria felino.

El ejemplo anterior muestra como se utilizan los árboles de decisión binarios (solo se obtienen dos ramificaciones por nodo), aunque la clasificación se puede hacer con más propiedades y con más de dos decisiones.

El árbol construido no es el único que puede describir el ejemplo, existe una cantidad finita de arboles de decisión que pueden representar el caso anterior donde se considera como mejor opción el árbol más simple [11], que se construye siguiendo algoritmos que miden el valor obtenido de los diferentes atributos clasificatorios. Uno de los algoritmos seminales para construir los arboles de decisión es ID3 desarrollado por Quinlan[11], donde de manera recursiva se construye el árbol de decisión. En la figura 2.2 se describe el procedimiento.

Una limitación al aplicar esta técnica radica en que el número de atributos y valores con las que cuenta cada dato debe ser pequeño y de esta manera puede ser descrito en su totalidad dentro de un árbol, pero en ejemplos que cuenten con una gran cantidad de atributos al igual que valores, resulta poco práctico y casi imposible trazar todas las posibilidades. Un buen ejemplo de esto es tener un dato que es descrito con 10 atributos y si cada atributo puede ser descrito con 4 diferentes valores entonces el árbol de decisión construiría 4^{10} ramificaciones, para posteriormente generar por cada combinación de atributos una ruta dentro del árbol de decisión.

```
1: si todos los ejemplos son positivos entonces
2:   devolver nodo positivo
3: fin si
4: si todos los ejemplos son negativos entonces
5:   devolver nodo negativo
6: fin si
7: si Atributos esta vacío entonces
8:   devolver voto mayoritario del valor del atributo objetivo en Ejemplos
9: si no
10:  Sea A atributo el MEJOR de atributos
11:  mientras cada valor v del atributo hacer
12:    Sea Ejemplos(v) el subconjunto de ejemplos cuyo valor de atributo A es v
13:    si Ejemplos(v) está vacío entonces
14:      devolver un nodo con el voto mayoritario de atributo objetivo de Ejemplos
15:    si no
16:      devolver Id3(Ejemplos(v), Atributo-objetivo, Atributos {A})
17:    fin si
18:  fin mientras
19: fin si
```

Figura 2.2: Algoritmo ID3

2.3. KNN

K vecinos más cercanos [12] (*K nearest neighbors*) es una técnica de clasificación supervisada que permite a una computadora decidir a que clase pertenece un dato en base a un conjunto de muestra que tiene dicho sistema. Es un método no paramétrico que estima el valor de la función de densidad de probabilidad. A grandes rasgos lo que realiza el sistema es comparar el dato que se quiere clasificar con un conjunto de datos de los cuales ya se conoce su respectiva clase, al hacer la comparación se decide la clase a la que pertenece el dato en cuestión utilizando un parámetro K que nos indica la cantidad de datos mas cercanos con los que se compara el dato a clasificar, éste parámetro varía dependiendo del problema.

Dado un conjunto $X_i = (x_{1i}, x_{2i}, x_{3i}, ..., x_{ni})$ donde X_i es un vector con un espacio de n características usualmente para conocer su cercanía con los datos del conjunto de entrenamiento se utiliza la distancia euclidiana que es la distancia que existe entre el origen y un punto determinado en el espacio *n-dimensional*. Tomando en cuenta las distancias entre el dato a clasificar y los datos muestra, se cataloga observando la mayoría de las clases que se encuentran a su alrededor.

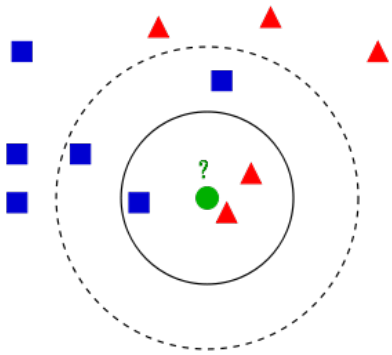


Figura 2.3: Ejemplo del método de KNN con $K = 3$

En un caso particular tenemos la tarea de clasificar plantas por sus características físicas como son el tamaño de los pétalos, sépalos, el color, etc. Utilizando una gran cantidad de ejemplos de plantas con sus atributos, se clasifica una planta desconocida realizando las comparaciones y conociendo cuales son las mas parecidas.

Esta técnica depende en gran medida del parámetro k (cantidad de datos con los que se estima su clase) por lo que existen casos en que con un cambio muy pequeño en este parámetro puede repercutir completamente en la clasificación final de los datos, por este motivo se deben de hacer una gran cantidad de pruebas y supervisar que valores de k realmente crean una buena clasificación.

Un problema recurrente en este método es considerar las características de los datos con la misma prioridad, esto significa que para la clasificación todos los atributos influyen de igual manera, cuando existen ejemplos en los que un atributo puede alterar en mayor grado el resultado de una clasificación. Para tener mayor precisión se debe estudiar y evaluar cada atributo y crear algoritmos que agreguen una ponderación adecuada a las n-caracteristicas y así que algunos atributos tengan mayor influencia a la hora de clasificar los datos.

2.4. Redes bayesianas

Las redes bayesianas [13] son métodos que permiten inferir situaciones o fenómenos a partir de gráfos que dan información probabilística relevante. Las redes bayesianas construyen modelos que van a caracterizar comportamientos ante ciertos escenarios, en el grafo los arcos indican dependencias e independencias probabilísticas directas entre las variables aleatorias, mientras que los nodos representan probabilidades. La red bayesiana es un grafo aciclico dirigido que indica la influencia que tienen los nodos padres sobre sus hijos por lo que el conjunto de probabilidades representadas en la red describe la distribución de probabilidad conjunta de todas las variables.

Considerando la probabilidad de que una persona sufra infarto depende de 4 factores, la practica deportiva que realiza la persona, la alimentación equilibrad, que tenga la presión sanguínea alta, y por ultimo si es fumador. Entre los factores también existen dependencias, ya que la presión sanguínea depende de deporte y la alimentación mientras que ser fumador es una variable independiente a las demás. Con esta información y las tablas de probabilidad relevantes construimos la red bayesiana de la figura 2.4

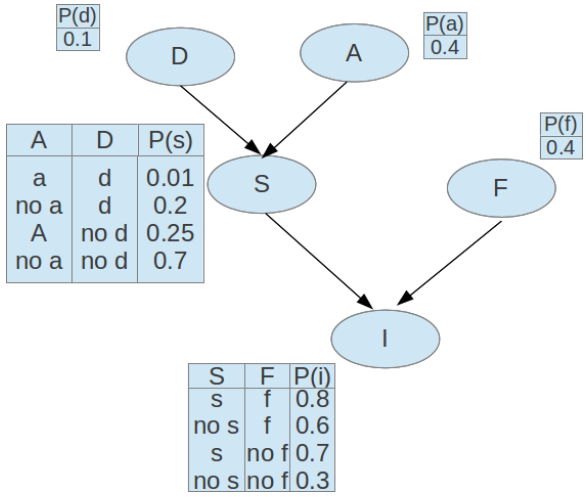


Figura 2.4: Red bayesiana

Donde:

- D - Practica deportiva habitual
- A - Alimentación equilibrada
- S - Presión sanguínea alta
- F - Fumador
- I - ha sufrido infarto

En esta red podemos inferir cual es la probabilidad de que una persona tenga un infarto si tiene la presión alta, es fumador, hace deporte y tiene una alimentación equilibrada. Como en cada nodo aparece la distribución de probabilidad con respecto a sus padres, nos permite factorizar la distribución de probabilidad conjunta, convirtiendolo en el producto de probabilidades condicionales independientes

$$P(x_1, x_2, , x_n) = \prod_{i=1}^{n} P(x_1 | padres(x_1))$$

$P(infarto = si \wedge \text{presión} = alta \wedge fumador = si \wedge deporte = si \wedge \text{alimentación} = equilibrada)$
=
$P(infarto = si | \text{presión} = alta, fumador = si)$
$P(presión = alta | deporte = si, alimentación = equilibrada)$
$P(fumador = si)P(Deporte = si)P(alimentación = equilibrada)$
$= (0.8)(0.01)(0.4)(0.1)(0.4) = 0.000128$

Por lo que la probabilidad de tener un infarto es muy baja (0.000128 %).

2.5. Redes neuronales

Las redes neuronales [14] son técnicas en la inteligencia artificial que tratan de emular el proceso de las neuronas para resolver problemas, esto lo hacen mediante un conjunto de perceptrones que trabajan y realizan operaciones análogas a las que harían las neuronas dentro del cerebro.

Un percetrón toma como entrada un vector de valores reales, calcula la combinación lineal de estas entradas y regresa 1 si el resultado es mas grande que un umbral y -1 en otro caso. En otras palabras, dadas las entradas x_1 hasta x_n, las salida $o(x_1, ..., x_n)$ procesada por el perceptrón será:

$$o(x_1, ..., x_n) = \begin{cases} 1 & \text{if } \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + ... + \omega_n x_n > 0 \\ -1 & \text{En otro caso} \end{cases}$$

Donde cada ω_i es un valor real constante, o peso, que determina la contribución de la entrada x_i para las salidas del perceptrón. Observando que la cantidad $(-\omega_0)$ es un umbral que el peso de las combinaciones de entradas $\omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + ... + \omega_n x_n$ deben sobrepasar para que la salida del perceptron sea 1.

Las redes neuronales utilizan de una fase de aprendizaje donde se da un conjunto de datos significativos como entrada de la red neuronal y se ajustan los pesos que tendrían los perceptrones, para adecuar a la salida deseada, de esta manera las redes neuronales conocen la manera de actuar dependiendo de los datos de entrada y salida del sistema.

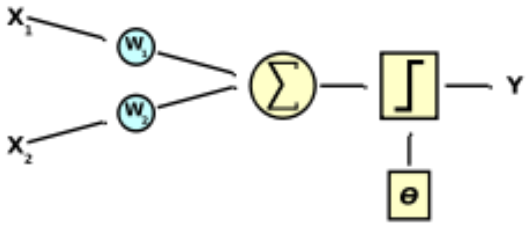


Figura 2.5: Perceptrón con dos entradas

La unión de todos los perceptores es lo que conforma la red neuronal y dependiendo de las conexiones entre estos crean arquitecturas diferentes, entre las que se encuentran las redes neuronales *feedforward* (alimentadas hacia adelante) que están formadas por un conjunto de capas de neuronas que alimentan con sus salidas a la siguiente capa. Estas capas son la de entrada donde los perceptores reciben la información del exterior, un conjunto de capas ocultas en la que se procesan los pesos de las entradas y se transfieren y por ultimo la capa de salida que nos entrega el resultado como lo mostramos en la imagen 2.6.

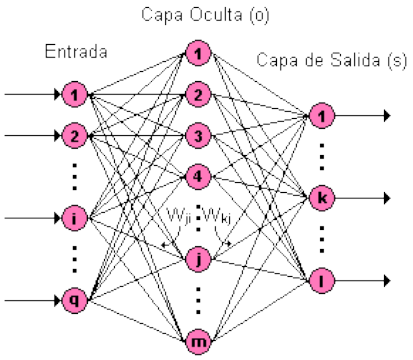


Figura 2.6: Red neuronal de 3 capas

Otra arquitectura de redes neuronales es la de retropropagación (del inglés *backpropagation*) que es una de las más aplicadas actualmente [15]. Aquí las capas de las redes neuronales ademas de tener comunicación con las capas posteriores, también tienen retrocomunicación con las capas anteriores. Como primer proceso se comunican las capas posteriores para generar una salida, terminando este proceso se calcula un error comparando la salida con un dato deseado y dicho error se envía a las capas anteriores con la finalidad de que cada neurona en las capas anteriores se adapte y clasifique con menos errores en posteriores pruebas. En la figura 2.7 se puede observar dicho comportamiento.

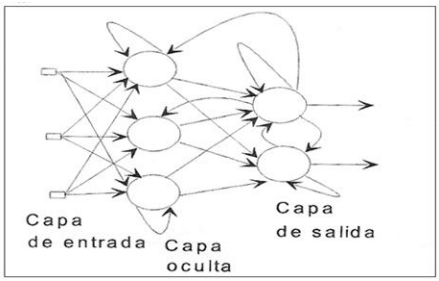


Figura 2.7: Red neuronal con retropropagación

Las redes neuronales en la actualidad son ampliamente utilizadas para el reconocimiento de patrones como los utilizados para reconocer huellas digitales, figuras en imágenes, etc. En la etapa de entrenamiento utilizamos muestras que le permiten a la red neuronal entrenar para poder reconocer

los patrones de cierta huella digital y así poder decidir a quien pertenece. Otro ejemplo es el reconocimiento de rostros, que trabajaría de manera muy similar, utilizando conjuntos de muestra muy variados para que en un futuro sea capaz de reconocer cualquier tipo de cara.

Figura 2.8: Las redes neuronales ayudan en el reconocimiento de patrones biométricos

Estos métodos han probado ser muy útiles en diversos campos, donde cada uno tiene sus ventajas y desventajas sobre los demás, donde solo en casos particulares se podría discriminar si una técnica es mas funcional que las otras. Los métodos mencionados anteriormente no son los únicos desarrollados para la clasificación autónoma de datos, existen muchas mas técnicas que se han investigado en el transcurso de los años donde cada vez se especializa más a campos particulares de aplicación.

Otro método que ha resaltado son las máquinas de soporte vectorial *SVM* que son técnicas de clasificación que permiten entrenar a la computadora para clasificar conjuntos de datos construyendo modelos de predicción. Esta técnica ha tomado gran importancia en el área de clasificación gracias a su amplio campo de aplicación, pero con la limitación de ser lentas cuando son usadas en problemas con grandes cantidades de datos. En este trabajo se propone un método que mejora el rendimiento de las *SVM* por lo que en las siguientes secciones se explica más a detalle el funcionamiento y limitaciones que tiene esta técnica.

2.6. Máquinas de soporte vectorial

Las máquinas de soporte vectorial (*SVM* por sus siglas en inglés) son técnicas de clasificación para conjuntos de datos desarrolladas por Vladimir Vapnik y colaboradores [1].

Las *SVM* son técnicas de clasificación supervisada por lo que trabajan con un conjunto de datos que sirve como entrenamiento, estos datos están clasificados en dos o más categorías y cada uno tiene un conjunto de atributos, las *SVM* construyen un modelo basado en el conjunto de entrenamiento que posteriormente será capaz de decidir la clasificación a la que pertenecen datos no vistos anteriormente, dichos datos pueden presentar información incompleta de sus atributos.

2.6.1. Funcionamiento

Para crear el clasificador las *SVM* consideran a los datos como puntos en un espacio n-dimensional donde las propiedades equivalen a las coordenadas y la n indica la cantidad de propiedades con las que cuenta el dato, a partir del espacio en el que se encuentran los datos se procede a construir un hiperplano que separe las diferentes clases.

En la figura 2.9 se muestra un caso bidimensional (datos con dos clases) construyendo rectas que separan datos de las diferentes clases.

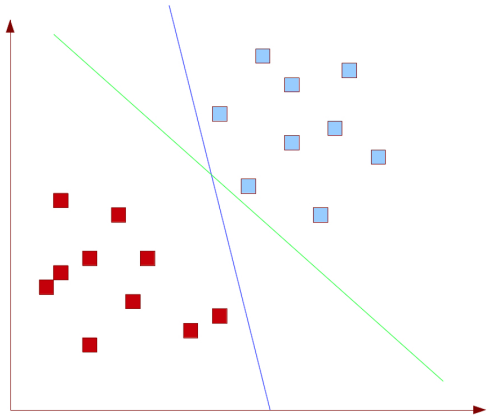


Figura 2.9: Ejemplo de dos clases linealmente separables

En el caso mostrado en la figura 2.9 existen una infinidad de hiperplanos que separan las clases, ahora queremos la mejor opción, para esto necesitamos maximizar las distancias que hay entre el hiperplano y todos los datos. Para esto tenemos un conjunto de entrenamiento D donde cada dato es representado como (x_i, y_i), x_i es un vector n-dimensional que contiene las características del dato y y_i indica a que clase pertenece. Para el caso donde hay solo dos clases identificadas como 1 y -1 tenemos:

$$D = ((x_i, y_i)|x_i \in R^n, y_i \in (-1, 1))_{i=1}^n$$

Si los datos están en un espacio bidimensional (tienen dos características) y sus clases son linealmente separables podemos representar dos hiperplanos de la siguiente manera:

$$w \cdot x_i + b \geq 1 \qquad \forall x_i \in Clase1$$

$$w \cdot x_i + b \leq -1 \qquad \forall x_i \in Clase2$$

Simplificando tenemos :

$$y_i(w \cdot x_i + b) + 1 \geq 0 \qquad \forall i$$

Las *SVM* buscan un hiperplano definido por: $w \cdot x + b = 0$ que sea el mejor candidato para ser el clasificador de datos. En la figura 2.10 mostramos 3 hiperplanos que separan las clases.

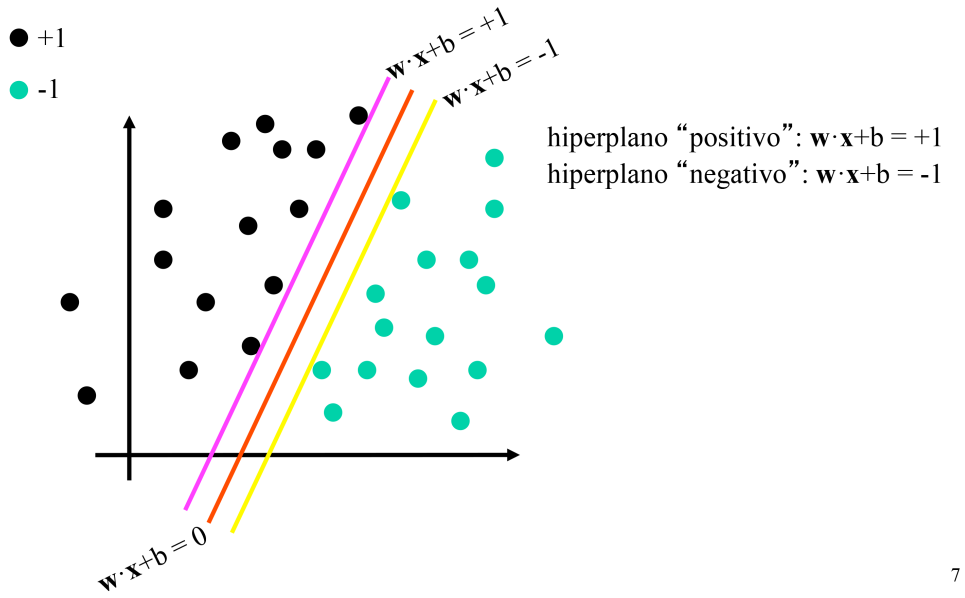


Figura 2.10: Clases separadas por un plano maximizando la distancia

Dado que w es la normal al hiperplano, entonces $\frac{|b|}{||w||}$ es la distancia desde el hiperplano hasta el origen, $\frac{|1-b|}{||w||}$ son las distancias entre el origen y los hiperplanos que interceptan con datos de las clases entonces la distancia que se desea maximizar es: $\frac{|2|}{||w||}$ éste problema de optimización también se puede plantear como:

$$\text{Minimizar:} \quad ||w||^2 \qquad \text{sujeto a:} \quad y_i(w \cdot x_i + b) + 1 \geq 0 \qquad \forall i$$

Cuando el conjunto de datos con el que se entrena la *SVM* no es linealmente separable se agregan variables que miden la dimensión de la violación a las restricciones impuestas en el caso anterior, de esta manera se calcula el margen pagando una penalidad proporcional a la violación de la restricción.

También se ocupa un método que permite transformar la variable de entrada (datos de entrenamiento) a un espacio de dimensión mayor donde si serán separables las clases.

2.6.2. Nucleo o *Kernel*

El truco de kernel (*kernel trick*) recibe un conjunto de datos de entrenamiento que originalmente no son linealmente separables en la dimensión en que se encuentran y los transforma a un espacio de características mayor donde se podrá trabajar de manera lineal. Esto facilita su clasificación con el uso de las *SVM*

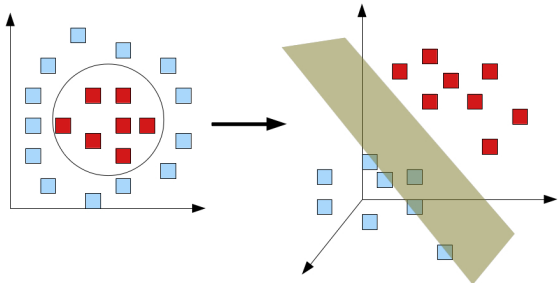


Figura 2.11: Transformación de espacio a través de una función kernel.

Existen muchas propuestas de funciones *kernel* que permiten separar clases no lineales, algunas funciones típicas son [20]:

- lineal $K(x_i, x_j) = x_i^t, x_j$
  El kernel lineal incrementa la posibilidad de que haya separación lineal en los datos de entrada.
- polinomial $K(x_i, x_j) = (\gamma x_i^t x_j + r)^d, \gamma > 0$
  Este tipo de kernel se utiliza cuando las clases son separadas por algún tipo de función polinomial donde d representa el grado de la función que podría separar a las clases de manera adecuada.

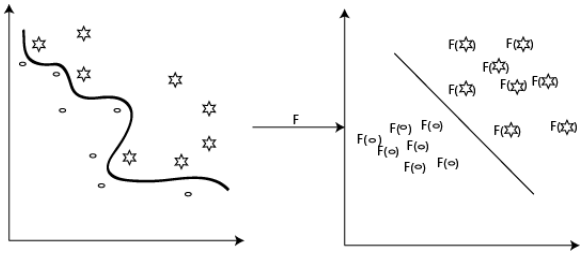
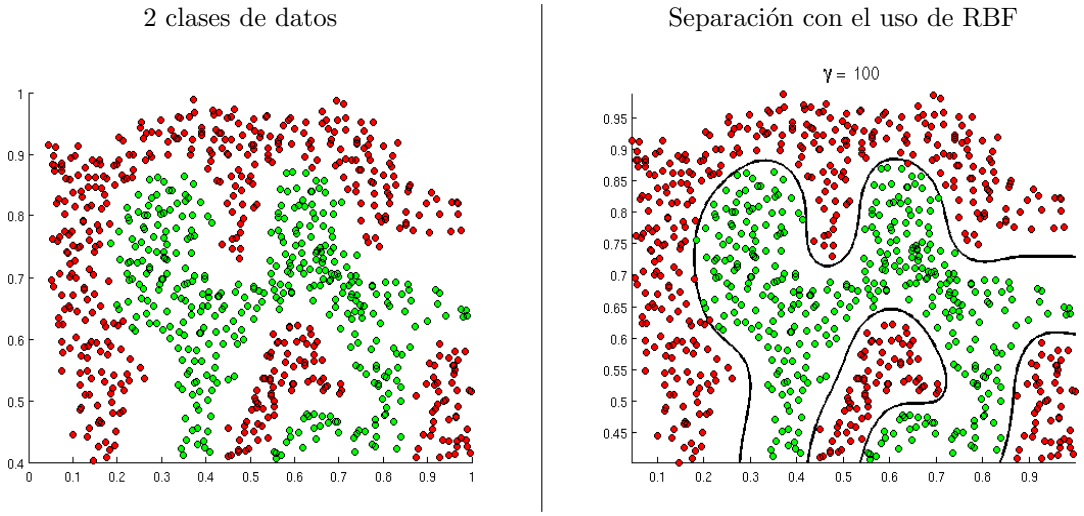


Figura 2.12: Ejemplo del uso de funciones polinomiales

- Función en base radial (radial basis fuction RBF) $K(x_i, x_j) = exp(-\gamma||x_i - x_j||^2), \gamma > 0$
  Éste kernel es una proyección a un espacio de dimensiones infinito donde si dos vectores se encuentran cerca entonces $||x_i - x_j||$ será pequeño. Por lo tanto para $\gamma > 0$ la expresión $-\gamma||x_i - x_j||^2$ será más grande. Esto conlleva a que para vectores cercanos entre ellos el valor del kernel *RBF* es más grande que para vectores más alejados.



Cuadro 2.2: Conjuntos de datos separados con kernel RBF [21]

- sigmoide $K(x_i, x_j) = tanh(\gamma x_i^t x_j + r)$
  γ, r y d son parámetros del kernel

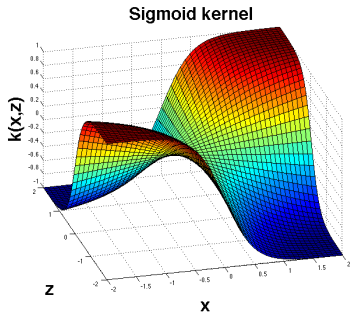


Figura 2.13: Gráfica del kernel sigmoide

2.7. Limitación de las *SVM*

La mayor limitación de las SVM es la poca escalabilidad al crear el modelo de predicción cuando crece el conjunto de entrenamiento, ya que los recursos de tiempo y memoria necesarios llegan a escalas intratables para muchos problemas comunes. En la siguiente tabla se muestran los tiempos de procesamiento que requieren las máquinas de soporte vectorial para crear un modelo de clasificación, el software utilizado es java con la librería LIBSVM.

Número de muestras	Tiempo de ejecución
1000	90 ms
2000	365 ms
3000	785 ms
4000	1.7 s
5000	2.6 s
10000	31.7 s
15000	2.1 m
20000	4 m
25000	6.8 m
50000	25 minutos
100000	4.12 horas
300000	13.57 horas
3000000	$\approx$ 24 días

Los conjuntos de muestras son datos de dos clases diferentes separados por una función no lineal, el kernel utilizado por la LIBSVM es RBF que entre los 4 mencionados anteriormente es el que tiene menor tiempo de procesamiento. Como se observa los tiempos no crecen de manera lineal junto con el conjunto de entrenamiento por este motivo se han estudiado diversas técnicas que permitan entrenar las SVM con mayor rapidez sin perder precisión en la clasificación.

Capítulo 3

Métodos para procesar grandes cantidades de datos en las *SVM*

Hasta la fecha se han desarrollado diversos trabajos que han tratado de encontrar posibles métodos para implementar SVM con conjuntos de entrenamiento grandes, aquí se busca entrenar el clasificador en un tiempo razonable sin perder precisión.

Generalmente estos métodos pueden ser divididos en dos tipos[24]:

- El primer tipo consiste en encontrar candidatos a vectores soporte, reduciendo el conjunto de trabajo y empleando técnicas de búsqueda, de tal forma que la SVM modificada sea capaz de entrenar conjuntos de datos con únicamente datos representativos reduciendo el tiempo de entrenamiento a un tiempo aceptable. Entre estos métodos se encuentran *Chunking* (fragmentación) [24] y *SBA* (Algoritmo de aprendizaje secuencial) [26].
- El segundo tipo consiste en descomponer el conjunto de datos de entrada en conjuntos pequeños, de tal forma que las SVM clásicas puedan ser utilizadas, como ejemplo de este tipo de métodos se encuentran *CB-SVM* (SVM basadas en cluster) [27] y *SebSVM* (SVM de conjuntos esféricos pequeños) [28].

A continuación se presentan algunos métodos propuestos que ayudan a que el entrenamiento de las SVM sea más rápido.

3.1. *Chunking* (fragmentación)

El método de fragmentación es una heurística que emplea un conjunto de datos llamados *chunk* [34] y se entrena a la SVM sobre esa porción de datos, con ese conjunto de datos encontramos los vectores de soporte y analizamos nuevos puntos que no se consideraron en el conjunto anterior. Si los puntos violan la condición *Karush-Kuhn-Tucker* [29] esos puntos se agregan al conjunto que elegimos anteriormente para formar un nuevo *chunk*. Esta iteración continua hasta que llegamos a un criterio de paro.

Aunque ninguna prueba teórica de la convergencia de este algoritmo ha sido realizada, en la práctica trabaja muy bien y hace posible entrenar conjuntos de datos de tamaño mediano [24].

3.1.1. Condiciones Karush-Kuhn-Tucker

Las condiciones Karush-Kuhn-Tucker juegan un rol muy importante en teoría de optimización, gracias a estas conocemos las condiciones necesarias para tener una solución óptima a un problema de optimización general. A continuación se describen dichas condiciones [25].

Dado un problema de optimización
minimizar

$$f(x_1, x_2, ..., x_n)$$

sujeto a

$$g_1(x_1, x_2, ..., x_n) \leq 0$$
$$g_2(x_1, x_2, ..., x_n) \leq 0$$

$$g_m(x_1, x_2, ..., x_n) \leq 0$$

Cambiamos la restricción de desigualdad $g_i \leq 0$ a una restricción de igualdad introduciendo una variable s_i de tal forma:

$$g_i \leq 0 \rightarrow g_i + s_i^2 = 0$$

De acuerdo a la técnica de los multiplicadores de Lagrange se construye la función:

$$F(x, \lambda, s) = f(x) + \sum_{i=1}^{m} \lambda_i \cdot (g_1 + {s_i}^2)$$

Los puntos que minimizan a f sujeta a las restricciones $g_i \leq 0 (1 \leq i \leq m)$ están dentro de los puntos críticos de F:

- Que hacen cero las parciales con respecto a las variables $x_j (j = 1, ..., n)$
- Que hacen cero las parciales con respecto a las variables $\lambda_i (i = 1, ...m)$
- Que hacen cero las parciales con respecto a las variables $s_i (i = 1, ..., m)$

Lo anterior se resume en el siguiente teorema que indica las condiciones que deben satisfacer los puntos que minimizan la función sujeta a las restricciones.

Suponga una formulación para el problema anterior de minimización. Si $x_0 = (a_1, a-2, ..., a-n)$ es un óptimo, entonces deben existir números reales llamados multiplicadores $\lambda_1, \lambda_2, ..., \lambda_n$ NO NEGATIVOS tales que $(a_1, a_2, ..., a_n, \lambda_1, ..., \lambda_m)$ es un punto crítico para F. Es decir que se cumple:

Bloque I

$$+\frac{\delta f(x_0)}{\delta x_j} + \sum_{i=1}^{m} \lambda_i \frac{\delta g_i(x_0)}{\delta x_j} = 0 \qquad j = 1, 2, ..., n$$

Bloque II: Condición de holgura complementaría

$$\lambda_i g_i(x_0) = 0 \qquad i = 0, ..., m$$

Bloque III

$$g_i \leq 0 \qquad i = 1, 2, ..., m$$

Si ahora el problema es de maximización la solución la cambiamos a un problema de minimización para $-f(x)$. En este caso la función F queda en la forma:

$$F(x, \lambda, s) = -f(x) + \sum_{i=1}^{m} \lambda_i \cdot (g_1 + {s_i}^2)$$

De tal forma que las condiciones que deben de satisfacer los óptimos ahora quedan de acuerdo al siguiente teorema.

Suponga una formulación para el problema anterior en el caso de maximización. Si $x_0 = (a_1, a_2, ..., a_n)$ es un óptimo, entonces deben existir números reales llamados multiplicadores $\lambda_1, \lambda_2, ..., \lambda_m$ NO NEGATIVOS tales que $(a_1, a_2, ..., a_n, \lambda_1, \lambda_2, ..., \lambda_m)$ es un punto crítico para F. Es decir que se cumple:

Bloque I

$$-\frac{\delta f(x_0)}{\delta x_j} + \sum_{i=1}^{m} \lambda_i \frac{\delta g_i(x_0)}{\delta x_j} = 0 \qquad j = 1, 2, ..., n$$

Bloque II: Condición de holgura complementaría

$$\lambda_i g_i(x_0) = 0 \qquad i = 0, ..., m$$

Bloque III

$$g_i \leq 0 \qquad i = 1, 2, ..., m$$

Uso de las condiciones KKT

La forma de operar las condiciones de KKT será la siguiente: Como lo que buscamos es el punto x_0 y de inicio se desconoce, entonces las ecuaciones de las condiciones de los bloques I y II se piensan como un sistema de ecuaciones en las variables x_j's y λ_j's: Se intenta resolver tal sistema de ecuaciones y en caso de encontrarse las soluciones se revisan una a una para ver cual de ellas cumple que los λ_j no son negativos y que también se cumplen las restricciones $g_i \leq 0$ en los puntos encontrados. Normalmente se realiza una tabla donde se hace la verificación.

3.2. Algoritmo de Optimización Mínima Secuencial

En las SVM maximizar el margen de separación entre 2 clases y el hiperplano se puede representar con el siguiente problema de optimización [30]:

$$\text{Minimizar} \quad ||w||^2 \quad \text{sujeto a} \quad y_i(w \cdot x_i + b) + 1 \geq 0, \forall i$$

Donde x_i es el vector del i-ésimo ejemplo de entrenamiento, y_i es la clase a la que pertenece. En este caso las clases de los datos están representadas por $y = +1$ y $y = -1$.

Para resolver la optimización se utilizan los multiplicadores de Lagrange, reemplazando nuestras condiciones por las condiciones propias de los multiplicadores de Lagrange. El problema se formula de la siguiente manera:

$$\text{Minimizar} \qquad f(w) = \frac{1}{2}||w||^2$$

$$\text{Sujeto a} \qquad y_i(w \cdot x_i + b) \geq 1 \quad i = 1, ..., l$$

El problema transformado con multiplicadores de Lagrange es:

$$L(w, b, \lambda) = \frac{1}{2}||w||^2 - \sum_{i+1}^{l} \lambda_i y_i (w \cdot x_i + b) + \sum_{i=1}^{l} \lambda_i$$

Para la solución debemos minimizar w y b mientras maximizamos con respecto a $\lambda \geq 0$. Calculando las derivadas con respecto a w y b e igualando a cero las ecuaciones se obtiene:

$$w^* = \sum_{i=1}^{l} \lambda_i^* y_i x_i \quad \lambda^* \quad \text{denota el mejor valor de la función de costos}$$

Se observa que la solución óptima es una combinación lineal de los datos de entrenamiento, en los cuales solo aquellos datos con $\lambda \geq 0$ contribuirán y son los llamados vectores de soporte.

El Algoritmo de Optimización Mínima Secuencial [30] (SMO por sus siglas en inglés) es un algoritmo que puede resolver rápidamente el problema de programación cuadrática QP. SMO descompone el problema general QP en subproblemas QP, usando el teorema Osuna [19] para garantizar convergencia.

SMO escoge resolver los problemas de optimización más pequeños en cada iteración. Para la mayoría de los problemas QP en las SVM el problema de optimización más básico involucra dos multiplicadores de Lagrange, ya que los multiplicadores de Lagrange deben de obedecer una restricción de igualdad lineal. En cada paso, SMO escoge dos multiplicadores de Lagrange para optimizarlos conjuntamente, encuentra el valor óptimo para estos multiplicadores y actualiza la SVM para reflejar el nuevo valor óptimo.

La ventaja de SMO recae en el hecho que resolver dos multiplicadores de Lagrange puede ser hecho analíticamente. Por lo tanto no se necesitan utilizar optimizaciones numéricas para resolver los problemas QP. El ciclo interior del algoritmo puede ser expresado en un conjunto de código C muy corto, más allá de invocar una librería entera para resolver problemas QP. A pesar de que se resuelven más sub-problemas de optimización dentro de la ejecución del algoritmo, cada sub-problema es tan rápido que el problema general QP se resuelve rápidamente.

Existen dos componentes para SMO: un método analítico para resolver dos multiplicadores de Lagrange, y una heurística para escoger cuales multiplicadores van a optimizarse.

SMO ha sido probado en problemas artificiales y de la vida real. Por las pruebas se puede deducir lo siguiente:

- SMO puede ser usado cuando el usuario no tiene fácil acceso a programas o paquetes de programación cuadrática o no desea actualizar las librerías de programación para resolver dichos problemas.

- SMO trabaja eficientemente en máquinas de soporte vectorial donde la mayoría de sus multiplicadores de Lagrange se encuentran en la frontera de los datos de entrenamiento.

- SMO trabaja adecuadamente para SVM con datos muy dispersos, incluso en problemas no lineales, ya que el tiempo de cálculo del *kernel* puede ser reducido, directamente acelerando el SMO. Dado que el método de fragmentación pasa la mayor parte del tiempo en procesos de programación cuadrática, no puede explotar tanto la linealidad de las SVM como la escasez de los datos de entrada.

- SMO trabaja bien para problemas grandes, gracias a que su escalabilidad con el tamaño de un conjunto de entrenamiento, es mejor que el método de fragmentación con las pruebas hechas hasta ahora.

Para varias pruebas el tiempo de entrenamiento de SMO se encuentra en un orden de $\sim N$ y $\sim N^{2,2}$. El tiempo de entrenamiento en el método de fragmentación se encuentra en una escala entre $\sim N^{1,2}$ y $\sim N^{3,4}$. Para problemas del mundo real, SMO puede se 1200 veces mas rápido para SVM lineales y 15 veces mejor para SVM no lineales.

El los cuadros 3.1, 3.2 muestra los tiempos de procesado entre el algoritmo *Chunking* y *SMO*, las pruebas se hicieron en un conjunto de datos que representan adultos donde se busca clasificar cuales tienen un ingreso de al menos $50,000, cada dato cuenta con 14 propiedades de las cuales 8 son categorías y 6 son continuas [32]. Para facilitar el experimento, los 6 atributos continuos fueron discretizados a quintiles.

Cuadro 3.1: Tabla comparativa SMO vs *Chunking* con SVM con kernel lineal
Fuente: Platt J., *Fast Training of support vector machine using sequential minimal optimization.*

Conjunto de entrenamiento	Tiempo SMO (segundos)	Tiempo *Chunking* (segundos)
1605	0.4	37.1
2265	0.9	228.3
3185	1.8	596.2
4781	3.6	1954.2
6414	5.5	3684.4
11221	17.0	20711.1
16101	35.3	N/A
22697	85.7	N/A
32562	163.3	N/A

Cuadro 3.2: Tabla comparativa SMO vs *Chunking*con SVM con kernel RBF
Fuente: Platt J., *Fast Training of support vector machine using sequential minimal optimization.*

Conjunto de entrenamiento	Tiempo SMO (segundos)	Tiempo *Chunking* (segundos)
1605	15.8	34.8
2265	32.1	144.7
3185	66.2	380.5
4781	146.6	1137.2
6414	258.8	2530.6
11221	781.4	11910.6
16101	1784.4	N/A
22697	4126.4	N/A
32562	7749	N/A

Las N/A en la tabla indican que las matrices eran demasiado grandes para 128mb de memoria RAM en el método *chunking* los resultados variaban menos por la imprecisión tolerada en las condiciones *KKT*.

3.3. Algoritmo de aprendizaje secuencial

En el algoritmo de aprendizaje secuencial (SBA Secuencial Bootstrapped Aceleration [26]) aprendemos de un subconjunto pequeño, y con la SVM construida comenzamos a encontrar cuales son los datos que están mal clasificados y los que se encuentran más alejados del hiperplano construido. Comenzamos con S^0 el conjunto inicial de datos usando los hiperplanos SBA busca a los siguientes mejores datos de S ya sea los que están más alejados del hiperplano h^+ o los que se encuentran mal clasificados h^- estos puntos son convexos a la clase que pertenecen y se van a agregar al conjunto S^k siendo k la iteración en la que se encuentra el algoritmo. Si se agregaron puntos a S^k el clasificador actualiza el conjunto de entrenamiento como $S^{k+1} = S^k \cup P$ donde P es el conjunto de datos que se están analizando en esta iteración . El algoritmo continúa hasta que no hay más puntos que cumplan los requerimientos . Se demuestra que el algoritmo converge por lo que siempre llega al punto donde no encontrará mas puntos para agregar.
El factor de aceleración de SBA en comparación con SVM es proporcional al cuadrado del tamaño de datos de entrada e inversamente proporcional al cubo del número de iteraciones. En los cuadros 3.3, 3.4, 3.5, 3.6, 3.7 se muestra una comparación del desempeño entre SBA utilizando SVM con kernel Gausiano contra SVM (libSVM) y CVM (Core Vector Machine), cada cuadro muestra diferentes conjuntos de datos fuente: Lichman, M. (2013). UCI Machine Learning Repository.

Cuadro 3.3: Conjunto de datos: *Splice*

	SBA/SVM	SVM	CVM
Precisión	89.43 %	89.12 %	88.94 %
Tiempo de entrenamiento	1.75s	1.89s	7.81s
vectores de soporte	475	691	666

Cuadro 3.4: Conjunto de datos: *Image*

	SBA/SVM	SVM	CVM
Precisión	96.98 %	97.04 %	97.01 %
Tiempo de entrenamiento	1.87s	3.03s	1.84s
vectores de soporte	166	163	179

Cuadro 3.5: Conjunto de datos: *Waveform*

	SBA/SVM	SVM	CVM
Precisión	90.75 %	91.03 %	90.95 %
Tiempo de entrenamiento	5.40s	7.82s	19.59s
vectores de soporte	129	1178	109

Cuadro 3.6: Conjunto de datos: *KingRook vs King*

	SBA/SVM	SVM	CVM
Precisión	92.97 %	93.08 %	92.53 %
Tiempo de entrenamiento	78.55s	122.2s	156.3s
vectores de soporte	2802	2998	2783

Cuadro 3.7: Conjunto de datos: *Adult A6*

	SBA/SVM	SVM	CVM
Precisión	83.67 %	84.55 %	83.51 %
Tiempo de entrenamiento	87.54s	152.7s	351.2s
vectores de soporte	651	4995	5187

Por lo que podemos observar en los cuadros 3.3, 3.4, 3.5, 3.6, 3.7 en cada uno de los ejemplos las precisiones varian muy poco, aproximadamente solo cambia un 1 %. Mientras que los tiempos de entrenamiento disminuyen en la mayoría de casos, incluso ahorran hasta un 40 % del tiempo de procesado y son mucho menores los vectores de soporte que encuentra.

3.4. SVM basado en cluster

La idea clave de las SVM basado en *cluster* (CB-SVM [27] por sus siglas en inglés) es utilizar una técnica jerárquica *micro-clustering* para conseguir una descripción fina cerca de la frontera y una descripción no tan fina más lejos de la frontera.
Esto lo logra construyendo árboles (*Clustering Feature Tree*) donde cada nodo contiene un conjunto de los datos que son considerados como un solo elemento y cada nodo puede a su vez contener otro subconjunto de datos, hasta considerar individualmente todos los puntos.
En cada árbol un nodo de un nivel superior es una representación resumida de sus nodos hijo como se representa en la figura 3.1.

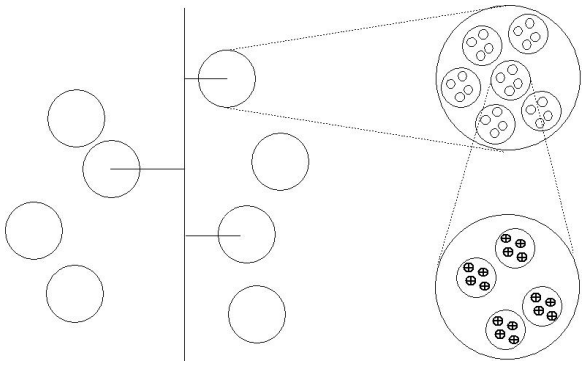


Figura 3.1: Representación gráfica de los grupos que se forman encapsulando datos de un solo tipo [27]

Por lo tanto *CF Tree* es una representación compacta de un gran conjunto de datos como se muestra en la figura 3.2

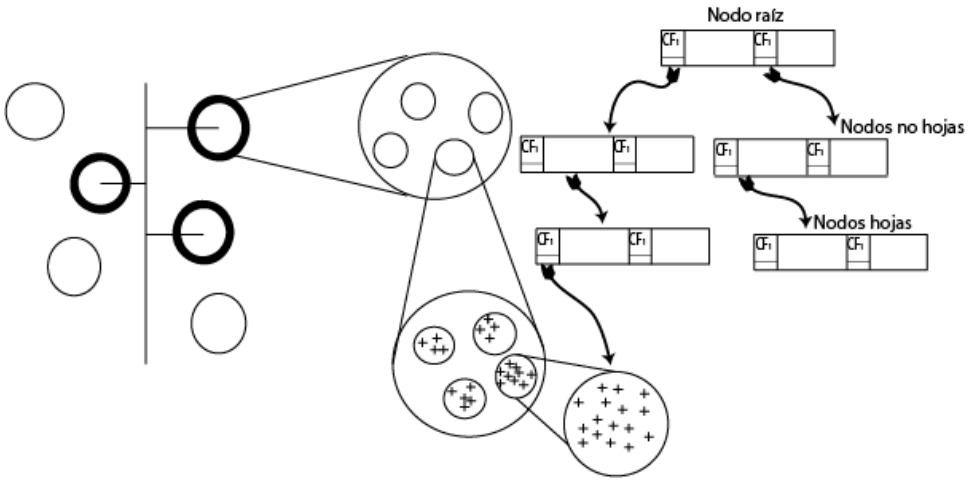


Figura 3.2: Árbol CF, cada esfera representa la clase de los datos que contiene. [27]

Una vez que se construyen los árboles para cada clase se empieza a entrenar la SVM solo con los nodos raíz (los subconjuntos de datos mas grandes). Una vez generados los vectores de soporte con los nodos raíz, se compara la distancia que existe entre estos y el hiperplano construido que separa las clases. Utilizamos los conjuntos de datos más cercanos a este hiperplano para crear subconjuntos (nodos hijo) y utilizarlos para la creación de nuevos vectores de soporte que clasifiquen mejor las clases como se muestra en la figura 3.3.

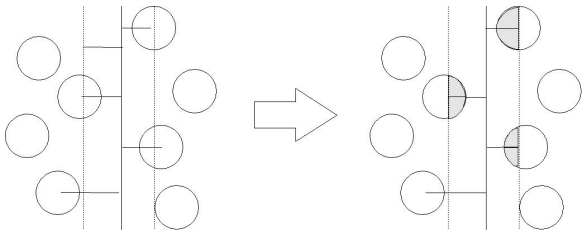


Figura 3.3: Desagrupar los conjuntos en subconjuntos más pequeños para construir con más precisión los vectores de soporte

En el cuadro 3.8 se muestra el desempeño del método con datos artificiales(Número de datos de entrenamiento: 113601, número de datos de prueba: 107072). FP:falso positivo;FN:falso negativo;Tiempo de muestreo para *CB-SVM* tiempo para construir el árbol CF.

	Original	CB-SVM	0,5 % muestras
Número de datos	113601	597	603
Tiempo de entrenamiento (seg)	160	0,003	0.003
Tiempo para construir el árbol CF(seg)	0.0	10.586	4.111
# de predicciones falsas (# de FP, # de FN)	69 (49,20)	86 (73,13)	243 (220,23)

Cuadro 3.8: Fuente: Hwanjo Yu, Jiong Yang, Jiawei Han. *Classifying Large Data Sets Using SVMs with Hierarchical Clusters.*

Con este método se pueden comentar los siguientes puntos:

- *CB-SVM* es muy escalable para conjuntos de datos muy grandes.
- Genera una alta precisión de clasificación

Sin embargo:

- Este método esta limitado al uso de un *kernel* lineal
- El radio y las distancias de los agrupamientos de datos que se generan cambian en un espacio de características muy alto.

3.5. Una aproximación geométrica para entrenar *SVM* en grandes conjuntos de datos

Esta técnica busca filtrar datos con un método geométrico intuitivo donde las muestras se encuentran en un espacio convexo y solo son considerados los datos que se encuentran en la frontera de esfera que encierra a todos los datos. En el caso de clases separables las *SVM* encuentran el plano separable con un máximo margen entre las clases. Como se muestra en la figura 3.4 la computadora utiliza algoritmos *convex hull* procesa los puntos mas cercanos a los datos que se encuentran en la frontera de las muestras.

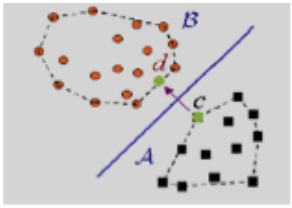


Figura 3.4: Demostración de algoritmo convex hull en un conjunto de puntos [33]

La propuesta en esta técnica se describe como sigue: En el inicio una función *kernel* adecuada deberá mapear los datos de entrenamiento a un espacio de características F, para que se obtenga un conjunto de datos que sea linealmente separable éste espacio de características. Entonces dos esferas de radio mínimo encerrarán los ejemplos de entrenamiento positivos y negativos. Con las esferas generadas identificamos los ejemplos que se encuentran en la frontera convexa dado que son los que se encuentran más cerca o dentro de la frontera de la esfera. Con estos datos entrenamos la *SVM* para generar el hiperplano de separación como se muestra en la figura 3.5

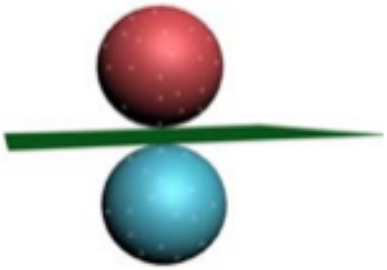


Figura 3.5: Hiperplano que clasifica a dos conjuntos de datos, cada conjunto se encuentra dentro de las esferas generadas [33]

Problema de encontrar la esfera mas pequeña

Dados un conjunto de puntos $S = x_1, x_2, ..., x_n$, donde cada $x_i \in R^d, i = 1, 2, ..., n$ la esfera mas pequeña de S (denotado SEB(S)) es la esfera mas pequeña que contiene todos los puntos de S.
El problema de encontrar la SEB de un conjunto de puntos es un problema bastante estudiado con un gran número de aplicaciones. Los algoritmos tradicionales para encontrar el SEB exacto no son eficientes para grandes dimensiones. Aun así es de gran interés estudiar algoritmos más rápidos que se aproximen a la solución con un factor multiplicativo $1 + \epsilon$ al valor óptimo, donde ϵ es un número entero pequeño. Sea $B(c, R)$ una esfera con centro c y radio R y dado un $\epsilon > 0$ una esfera $B(c, (1 + \epsilon)R)$ es una $(1 + \epsilon)$-aproximación de SEB(S) si $R \leq r_(MEB(S))$ y $S \subset B(c, (1 + \epsilon)R)$. En muchos problemas ajustados se encuentra que la solución del problema en un subconjunto, llamado conjunto núcleo, Q puntos de S pueden dar muchas veces una aproximación eficiente y precisa [28].

El algoritmo para obtener la $(1 + \epsilon)$-aproximación puede ser descrito como sigue: En la i-esima iteración el algoritmo procesa el aproximado SEB del conjunto actual $X \subset S$, y entonces revisa si la $(1+\epsilon)$-expansión de esta esfera contiene a S, si este es el caso el algoritmo regresa la esfera expandida y el actual conjunto como la solución, en otro caso el algoritmo selecciona los puntos mas alejados de S desde el centro de la esfera aproximada más pequeña de X, los agrega a X y repite el ciclo.

En el cuadro 3.9 se compara el desempeño entre los métodos *SebSVM*, LIBSVM 2.81, CVM, RSVM. Los datos para entrenar fueron obtenidos de los repositorios de la UCI [32] y consisten en 581012 muestras donde cada datos representa información para clasificar tipos de bosques. Los datos fueron divididos en dos grupos, uno de 530012 y otro de 50000. El conjunto mayor se considera como de entrenamiento y el menor es para verificar el clasificador. El kernel utilizado fue *RBF* con $g = 2$ y $c = 8$. En la pruebas se considero tomar del conjunto de entrenamiento un subconjunto cada vez más grande para analizar los tiempos de entrenamiento y la precisión.

	SebSVM	SVM2.81	RSVM	CVM
# de ejemplos de entrenamiento	100,000	100,000	100,000	100,000
Precisión (%)	88.06	89.42	83.39	90.12
Tiempo de entrenamiento (s)	328	5,446	15	33,269
# vectores de soporte	4,280	31,802	5,740	37,938
# de ejemplos de entrenamiento	200,000	200,000	200,000	200,000
Precisión(%)	89.45	90.49	85.71	91.38
Tiempo de entrenamiento (s)	474	10,492	285	55,970
# vectores de soporte	8,540	57,383	8,529	54,013
# de ejemplos de entrenamiento	300,000	300,000	300,000	300,000
Precisión(%)	90.16	91.04	86.71	92.06
Tiempo de entrenamiento (s)	640	19,347	700	72,764
# vectores de soporte	12,047	83,250	11,752	61,611
# de ejemplos de entrenamiento	400,000	400,000	400,000	400,000
Precisión(%)	90.96	91.49	87.39	
Tiempo de entrenamiento (s)	864	36,238	1,320	
# vectores de soporte	15,647	104,461	15,121	
# de ejemplos de entrenamiento	530,012	530,012	530,012	530,012
Precisión(%)	91.62	91.73	87.94	
Tiempo de entrenamiento (s)	1,207	77,686	2,451	
# vectores de soporte	18,757	133,465	18,162	

Cuadro 3.9: Comparación de la precisión y el tiempo de entrenamiento que se obtiene al utilizar algunos métodos que agilizan el entrenamiento en las SVM
Fuente: Zhi-Qiang Zeng, Hua-Rong Xu, Yan-Qi Xie, *A Geometric Approach to Train SVM on Very Large Data Sets*.

Como se puede observar la precisión de SebSVM es un poco menor que LIBSVM, pero es más rápido en la etapa de entrenamiento. Aunque CVM tiene las precisiones más altas entre todos los métodos también es el que más tarda al entrenar. Ademas el número de vectores de soporte encontrados por SebSVM es mucho menor que en el caso de LIBSVM y CVM, que indica pruebas más rápidas. Aunque la precisión de SebSVM es mejor que RSVM en un 3.82 % en promedio con una cantidad similar de vectores de soporte, el tiempo de entrenamiento es mucho menor en SebSVM cuando la cantidad de datos de entrenamiento crece.

Como se observó en este capitulo existen una gran variedad de trabajos de investigación que nos permiten agilizar los tiempos de entrenamiento de las SVM, los algoritmos encargados de realizar esta tarea buscan construir los modelos de clasificación en un tiempo menor a lo que tardaría normalmente. Para realizar esta tarea los algoritmos pueden basarse en realizar optimizaciones matemáticas cuando se construyen los hiperplanos (*chunking*, SMO) o en filtrar los datos que se ocupan en el proceso de entrenamiento (*CB-SVM*, SebSVM).

Capítulo 4

Método propuesto: Discriminación de datos usando detección de orillas.

Conocemos el funcionamiento de las *SVM* y la manera en como crean modelos de predicción a partir de conjuntos de entrenamiento, considerando un caso de dos dimensiones (datos que cuentan con dos características) que puede ser fácilmente representado en una gráfica, y mostrando el hiperplano construido por las máquinas de soporte vectorial observamos:

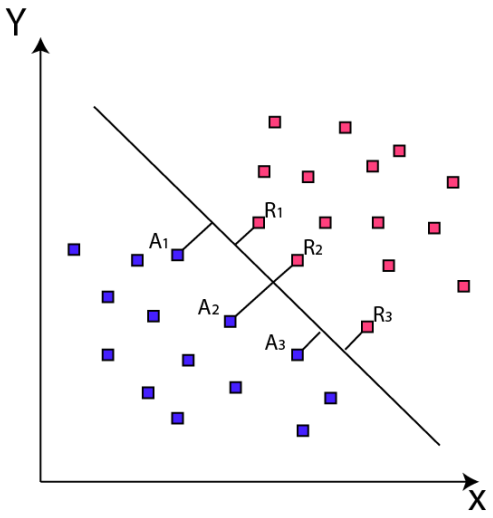


Figura 4.1: Representación de datos en un plano cartesiano considerando sus atributos como coordenadas construcción del hiperplano en *SVM*

En la imagen 4.1 El hiperplano esta descrito por los puntos $A_1, A_2, A_3, R_1, R_2, R_3$, idealmente

esos puntos son los únicos necesarios para la construcción de un modelo que clasifique los datos de manera adecuada, por lo que surge la idea de crear un filtro que sea capaz de identificar cuáles van a ser los datos necesarios y suficientes para que las *SVM* clasifiquen de manera adecuada.

Se considerá que los algoritmos de detección de bordes tienen el comportamiento esperado para poder crear el filtro, estos métodos identifican dentro de una imagen si el punto en que se encuentra representa la frontera de un objeto [37] y por ende es representativo, o en caso contrario el dato esta rodeado completamente de más datos similares a él y provocaría tiempos de proceso innecesarios para encontrar los vectores de soporte.

Se plantea aplicar un método en dos etapas para poder resolver el inconveniente de las *SVM*. A continuación se describe el proceso de cada etapa:

- Etapa de discriminación de datos: Se analiza el conjunto de datos con los que se entrena la *SVM* para analizar cuales son los datos más significativos. En un principio se considera un conjunto de datos con solo dos propiedades y de dos clases distintas, esto es con la finalidad de representar los datos en un plano de dos dimensiones y considerarlo como una imagen para utilizar técnicas de procesado de imágenes para realizar la discriminación.
- Etapa de aprendizaje: En esta etapa utilizamos el conjunto de datos generado en la etapa anterior para la construcción del clasificador, esta tarea se realiza con una librería de java llamada LIBSVM [36] que entrena las *SVM* y permite realizar pruebas de la precisión cuando clasifica. Gracias a la capa de filtrado reducimos el tiempo de procesado para esta tarea, la reducción de tiempo depende de la cantidad de datos representativos que se encuentran en el conjunto inicial, pero se espera tener en el peor de los casos un comportamiento casi idéntico a no utilizar la capa de filtrado, optimizando en el mejor casos el tiempo de manera exponencial.

4.1. Métodos de detección de bordes

Las herramientas de detección de bordes permite conocer las fronteras o limites en los objetos de una imagen, esto es de gran utilidad para el método propuesto dado que en una primera instancia se plantea ver a los datos en una imágen (dibujando cada dato como un punto con coordenadas x,y) para reconocer cuales se encuentran en la frontera y por lo tanto son los representativos para la construcción de los vectores de soporte necesarios para crear un modelo de clasificación adecuado. Si los datos se encuentran muy dispersos el uno del otro, los métodos de detección de bordes identificarían a todo el conjunto como datos significativos, por lo que se realizan algunas adaptaciones que nos permitan descartar datos.

A continuación presentamos algunas definiciones de los algoritmos para detectar los bordes en las imágenes y un planteamiento de como son aplicados dentro del proyecto.

4.1.1. Definición

Los bordes u orillas en una imagen son cambios o transiciones de intensidad en los píxeles que la representan. Los bordes generalmente ocurren en el limite entre dos diferentes regiones de la imagen [37]. La finalidad de estos métodos es resaltar un objeto en particular, conocer los limites de

una región, identificar objetos, etc. Para el análisis de imágenes es de gran importancia conocer el contorno de los objetos. Dentro del proyecto representaremos los datos como píxeles con tonalidades diferentes dependiendo de la clase a la que pertenecen.

Existen muchas causas que provocan el cambio abrupto de intensidad, entre las que están:

1. Varios eventos físicos.
2. Eventos geométricos:

   Limites de un objeto

   Limites en una superficie
3. Eventos no geométricos

   Especularidad (reflexión de luz como un espejo)

   Sombras (De otros objetos o del él mismo)

Esto provoca que encontrar los bordes en algún objeto no sea una tarea fácil, en la figura 4.2 mostramos una imagen 1-dimensional que representa un conjunto de píxeles y con tonalidades diferentes, donde no existe un cambio abrupto en la intensidad de color.

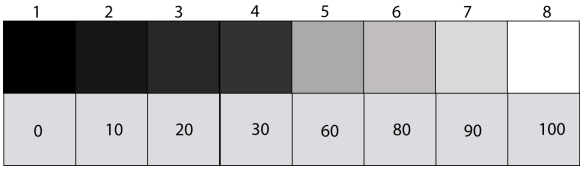


Figura 4.2: imagen 1-dimensional

Claramente se observa que existe un borde entre el recuadro 4 y 5, pero dado que cada uno tiene una intensidad de color diferente debemos de crear mecanismos con los que sea capaz de reconocer los cambios sutiles y los cambios pronunciados.
Para esto necesitamos conocer los descriptores de bordes que permiten tener un mejor control sobre la forma en que se identificaran las orillas de una imagen.

4.1.2. Descriptores de Bordes

Los atributos que permiten tener una mejor descripción de los bordes son:

1. Normal del borde: Vector unitario que tiene la dirección donde se produce la mayor intensidad de cambio. Esto significa un vector que apunta al píxel adyacente menos parecido al actual.
2. Dirección del borde: Vector unitario perpendicular a vector normal del borde.
3. Posición del borde o centro: La posición de la imagen donde se encuentra el borde.
4. Resistencia del borde: Relacionado al contraste de la imagen local a lo largo de la normal.

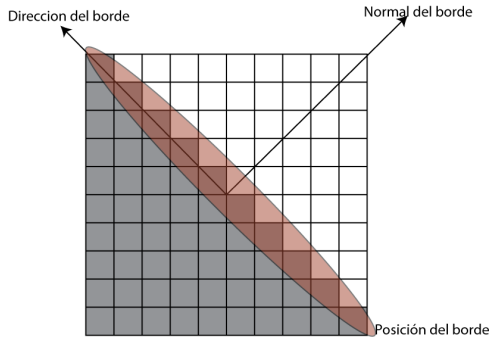


Figura 4.3: Vectores que describen el borde (vector normal y dirección de borde)

4.1.3. Modelando los cambios de intensidad

Los bordes pueden ser modelados de acuerdo a los cambios de intensidad y el tiempo en que cambia de un color a otro. De esta manera puede identificar diferencias de tonalidades sutiles o muy pronunciadas en los píxeles.
La figura 4.4 muestra algunos modelos de detección de bordes y el ruido provocado por diferentes factores.

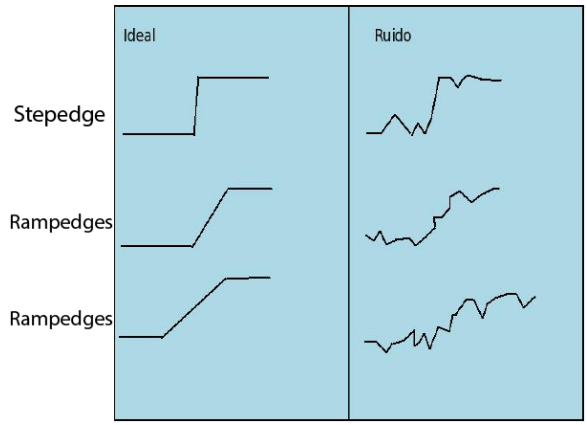


Figura 4.4: Ejemplo que describe a los cambios de la intensidad en *Stepedges* y *Rampedges*
Fuente: Edge detection Trucco, Chapter 4 and Jain et al. Chapter 5.

El ruido puede ser provocado por fallas en la imagen, compresión en los formatos, error de lectura, también la imagen puede contener ruido por factores como la luz ambiental, el dispositivo que captura la imagen, etc. Esto provoca que los píxeles no muestren el valor real de la imagen, o que muestren falsos bordes por lo que evitaremos crear estas situaciones cuando construyamos las imágenes.

Step edge: La intensidad de la imagen cambia abruptamente de un valor de un lado de la discontinuidad a un diferente valor en el lado opuesto. La figura 4.5 muestra claramente un cambio abrupto de intensidad en los colores del los píxeles 4 y 5 por lo que se considera un borde

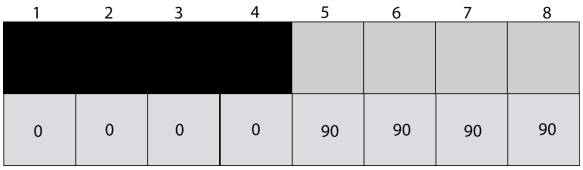


Figura 4.5: Ejemplo de *Stepedge*

Ramp edge: Un *Step edge* donde la intensidad de cambio no es instantánea pero ocurre en una distancia finita. La figura 4.6 muestra una cambio entre los píxeles 1 y 6 igual que en la figura 4.5 pero el cambio de intensidad es gradual a través de los píxeles intermedios

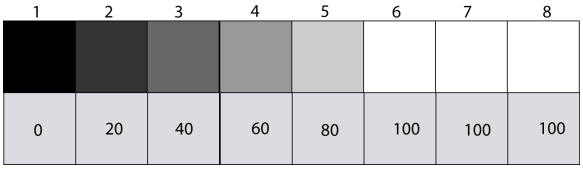


Figura 4.6: Ejemplo de *Rampedges*

La figura 4.7 muestra mas modelos de detección de bordes, también como el ruido con el que se puede presentar al identificar los bordes.

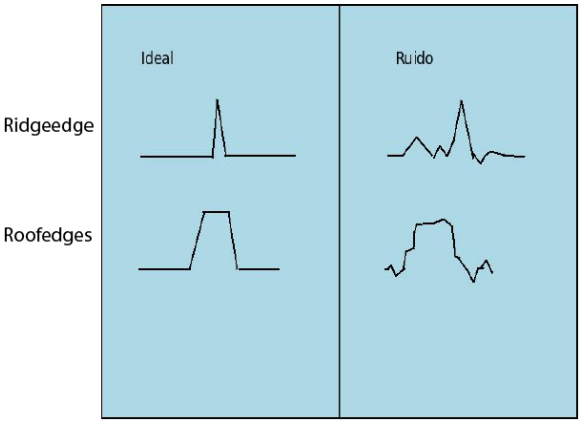


Figura 4.7: Modelo que describe los cambios de intensidad en *Ridgeedge* y *Roofedge*
Fuente: Edge detection Trucco, Chapter 4 and Jain et al. Chapter 5.

Ridge edge: La intensidad de la imagen cambia abruptamente pero regresa al valor inicial en un periodo de tiempo corto (generado usualmente por lineas). Se observa en la figura 4.8 como solo un píxel intermedio cambia en tonalidad y los demás se mantienen iguales.

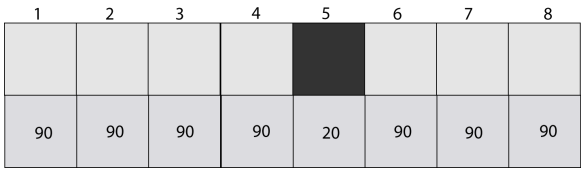


Figura 4.8: Ejemplo de *Ridgeedges*

Roof edge: Un *Ridge edge* donde la intensidad de cambio no es instantánea pero ocurre en una distancia finita (generados usualmente por la intersección de superficies). Muy parecido a la figura 4.8 con la diferencia que aquí el cambio tarda más en llevarse a cabo, por lo que los píxeles intermedios pueden presentar varias tonalidades pero que los extremos se mantienen con el mismo color como se puede obaservar en la figura 4.9.

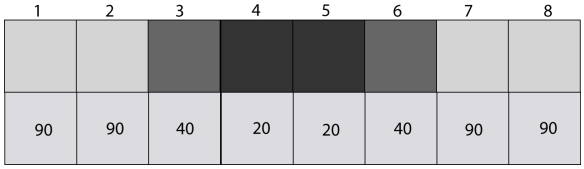


Figura 4.9: Ejemplo de *Roof edge*

Utilizando los modelos se utilizan máscaras que identifican las orillas dependiendo de los valores de cada píxel, para el caso de *Step Edge* una mascara adecuada seria $[-1, 0, 1]$ (primera derivada) este aplicado en una imagen de 1 dimensión identificaría los píxeles donde se produce un cambio.

En los cuadros del 4.1 al 4.4 mostramos ejemplos usando modelo de bordes y la máscara [-1, 0, 1] (centrado en x), lo que significa que para cada píxel se tomará el píxel anterior y el posterior para hacer una diferencia, si el resultado es diferente de 0 significa que hay un cambio de tonalidad por lo que representa un borde.

S_i Son los valores de los píxeles originalmente.
$S_i \otimes M$ Es el resultado de aplicar la mascara [-1, 0, 1]

S_1	12	12	12	12	12	24	24	24	24	24
$S_1 \otimes M$	0	0	0	0	12	12	0	0	0	0

Cuadro 4.1

S_2	24	24	24	24	24	12	12	12	12	12
$S_2 \otimes M$	0	0	0	0	-12	-12	0	0	0	0

Cuadro 4.2

S_3	12	12	12	12	15	18	21	24	24	24
$S_3 \otimes M$	0	0	0	3	6	6	6	3	0	0

Cuadro 4.3

S_4	12	12	12	12	24	12	12	12	12	12
$S_4 \otimes M$	0	0	0	12	0	-12	0	0	0	0

Cuadro 4.4

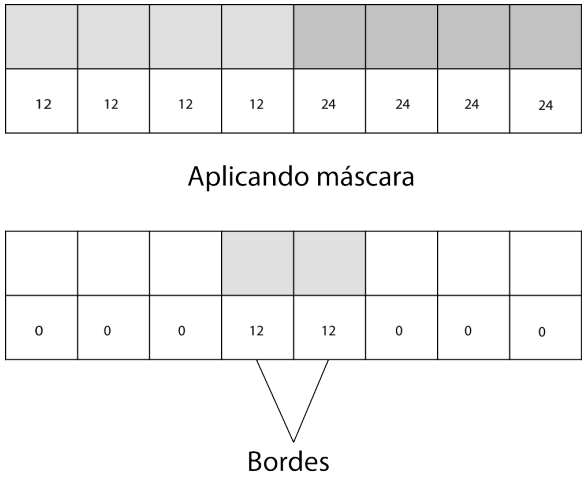


Figura 4.10: Aplicando máscara a imagen 1-dimensional

Dado que las imágenes creadas a partir de este método son una colección de puntos, las fronteras de las imágenes estarían bien definidas, por lo que no es necesario en profundizar en métodos de detección de bordes más sofisticados, sin embargo se estudia la forma en como tratar la dispersion que existe entre los datos para que se reconozcan bordes aún cuando los puntos no se encuentran de manera continuo. Las imágenes representan los datos por su posición en el plano (atributos) y el color de los puntos (clase). Con este planteamiento los modelos de detección de bordes que mejor se adaptan son *Ridge edge* y *Step Edge* ya que en los bordes siempre existirían cambios abruptos por la ausencia de datos o por el cambio de clase entre estos.

Clase	Atributo 1	Atributo 2
0	15	20
1	5	10
1	10	14
1	17	10
0	14	27
1	8	15
0	15	8
0	7	6
0	5	5

Cuadro 4.5: Datos de muestra

4.2. Filtro

Creamos un ejemplo en dos dimensiones para poder visualizar la imagen en un plano bidimensional y se propone una máscara *step edge* para detectar los cambios de intensidad abruptos, esta mascara es útil por la manera en que se construyen las imágenes, en donde cada tipo de clase pertenece a un color diferente, por lo que la detección trabajará de manera eficiente sin obtener falsos bordes o que tenga la necesidad de considerar sombras dentro de la imagen o fallas de origen.

Dado que es una imagen bidimensional, los datos son representados por una colección de números reales que están formados como lo muestra la tabla 4.5

En la figura 4.11 observamos los dos tipos de clase identificados por el color y que en conjunto forman patrones con lo que identificaremos los bordes o datos representativos.
Los datos son de dos tipos diferentes y limitamos el rango de los atributos a una escala de 0 a 200, por lo que la imagen tiene un tamaño de 200 x 200 píxeles. Y el filtro va a realizar un recorrido desde la posición (0,0) (superior izquierda) hasta la posición (200,200).

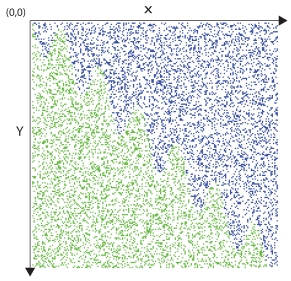


Figura 4.11

Para aplicar el método usamos una máscara: $\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$ (segunda derivada) con esto los datos (píxeles) identificarán cambios en 4 posiciones adyacentes a ellos (arriba, abajo, izquierda, derecha). y en caso de que cualquiera de las posiciones exista un color diferente al propio se reconocerá como borde.

Aplicando el filtro a la imagen mostrada en la figura 4.11 se observa que el filtro no es muy eficiente, esto lo provoca la gran dispersión de datos, en la figura 4.12 visualizamos el problema mencionado.

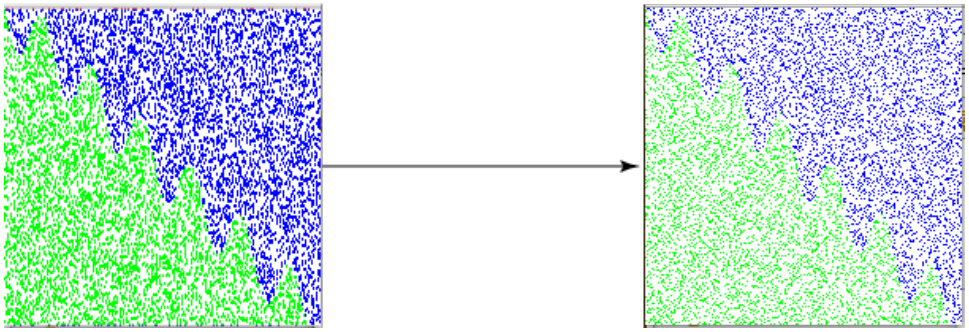


Figura 4.12

Dado que los datos son discontinuos se propone un algoritmo que permita identificar los bordes utilizando máscaras que permitan a cada punto identificar si esta rodeado de datos de su misma clase, si es el caso se considera que dicho dato no es necesario para el entrenamiento de las SVM.

Para el caso de datos con dos atributos, el algoritmo considera tomar cada dato de entrenamiento como un punto en el plano cartesiano $P = (x_i, y_i)$ y se procede a buscar si existen datos de entrenamiento que se encuentren adyacentes a dicho punto.

Por lo tanto se generan 4 puntos de búsqueda

$$A = (x_i - 1, y_i), B = (x_i + 1, y_i), C = (x, y_i - 1), D = (x, y_i + 1)$$

Si existen los cuatro puntos dentro del conjunto de entrenamiento se puede discriminar el datos ya que no se encuentra en un borde, en caso contrario se procede a realizar una búsqueda hacia posiciones del plano más alejadas, entonces tendremos la búsqueda sobre los siguientes puntos:

$$A = (x_i - 2, y_i), B = (x_i + 2, y_i), C = (x, y_i - 2), D = (x, y_i + 2)$$

Generalizando se tendrán los puntos $Q_i = (a_i, b_i)$ tal que:

$$\forall a_i \in \mathbb{N} : x_i - Z \leq a_i \leq x_i + Z$$

$$\forall b_i \in \mathbb{N} : y_i - Z \leq b_i \leq y_i + Z$$

El parámetro Z es un número entero que indica a que distancia se buscarán datos de la misma clase, entre más grande sea Z el algoritmo reconocerá mejor los bordes pero crecerá su tiempo de ejecución.

A continuación aplicamos el mismo filtro pero con distancia de búsqueda mayor y en la figura 4.13 observamos como el filtro reconoce mejor los datos que se encuentran en la frontera.

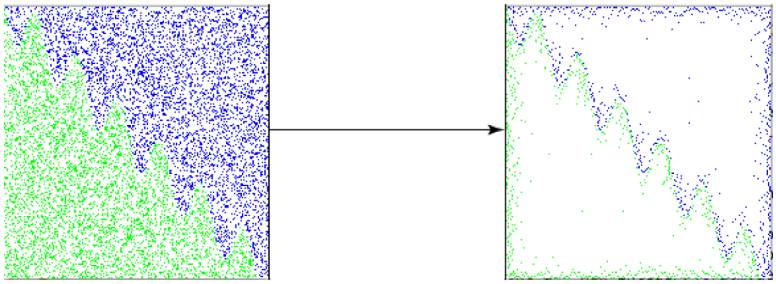


Figura 4.13

4.3. Método general

La generalización es necesaria para poder aplicar el filtro a un conjunto más amplio de problemas en la vida real, hasta ahora se esta limitando el uso del algoritmo a ejemplos donde solo se cuenta con dos propiedades. Se necesita escalar la solución para poder identificar los bordes en planos de 3 o más dimensiones. Bajo el mismo planteamiento se representan las propiedades en una matriz donde cada fila equivale a un dato y la primera columna especifica la clase del objeto mientras que las columnas subsecuentes indican los atributos.

clase	Atributo 1	Atributo 2	..	Atributo N
1.0	75.0	14.0	..	0
0.0	70.0	15.0	..	1

Para recorrer el arreglo se necesita conocer cual es el límite de los números que existen para cada propiedad, en los ejemplos utilizados en el capitulo anterior cada dato contaba con dos atributos y la clase a la que pertenecía, cada atributo era un número entero en el rango 0 a 200, por lo que dentro de la imagen el mapa solo tenia dimensiones de 200 x 200 píxeles. En el caso tridimensional (3 propiedades por atributo) el recorrido de toda la imagen se tendría que hacer desde la coordenada (0,0,0) hasta la coordenada (200,200,200), En este caso aun es posible hacer el recorrido en un tiempo factible, pero qué pasa cuando el problema crece a 7 o más dimensiones y el valor de las propiedades es mucho más grande. El problema que generamos crece de manera

$$N^M$$

Donde:

N es el número real al que llegan las propiedades.
M es el número de propiedades que contiene el dato.

Dado que el problema crece exponencialmente con el número de propiedades, planteamos otro tipo de recorrido e identificación de bordes siguiendo la idea de dos dimensiones, de esta manera el recorrido no sería a través de todas las coordenadas posibles en un plano multidimensional, solo se comprobaran los puntos (datos) que se tienen en el conjunto de entrenamiento.

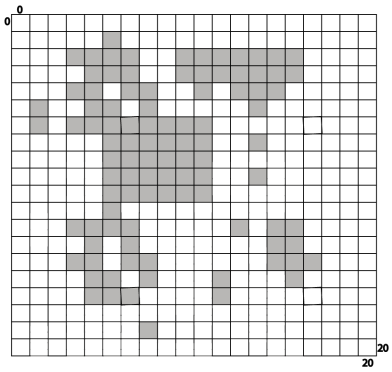


Figura 4.14: Representación de imagen de 20 x 20, las regiones grises representan datos

En la figura 4.14 se muestra un caso en dos dimensiones donde los recuadros grises representan datos del conjunto de entrenamiento mientras que los espacios en blanco son puntos vacíos. Donde para simplificar el proceso de filtrado solo se aplicaría la máscara del filtro sobre los puntos grises en la imagen, de esta manera el tiempo de computo disminuiría al no recorrer todo el espacio dimensional. Para obtener un mejor filtro se revisan los píxeles adyacentes y recorremos a una distancia apropiada para no perder datos representativos.

Como primera parte del algoritmo se genera una tabla *Hash* que va a contener todos los datos de entrenamiento y se comenzará a recorrer cada elemento generando los datos de búsqueda. Para el caso base de dos atributos por dato, se generar 4 nuevos puntos y se revisa si dichos puntos se encuentran en la tabla hash, para realizar una búsqueda mas exhaustiva se generan mas puntos para buscar

Ahora se necesita un parámetro que indicará la distancia de profundidad que manejara la máscara para encontrar otros puntos y conocer si es una borde o no. Mientras este parámetro crezca, la cantidad de datos filtrados crecerá pero tardará mas tiempo en recorrer los datos, en otro caso la distancia a la que buscará posibles puntos orillas disminuirá y la cantidad de datos filtrados será menor con ventaja de que terminara el tiempo de filtrado mas rápido.

Con el nuevo parámetro se realiza un recorrido de todos los datos y a partir de cada uno crear un subconjunto que representara los datos alrededor, de esta manera comparamos con el conjunto total de datos y verificamos si el dato en cuestión es borde si los datos del subconjunto se encuentran dentro del total de datos de aprendizaje.

4.3.1. Algoritmo

1: Guardar conjunto de entrenamiento en una tabla hash.
2: **para** Cada elemento en la tabla hash. **hacer**
3: Obtener valor n que indica la cantidad de atributos del elemento.
4: Crear un arreglo A con $2n$ números enteros inicializados en 0.
5: **para** k=1 hasta k=profundidad de busqueda **hacer**
6: **para** Cada atributo i en elemento **hacer**
7: Genera dato adyacente aumentando k en *i-ésimo* atributo
8: **si** dato generado se encuentra en la tabla hash **entonces**
9: Incrementa arreglo A en su posición $2i$.
10: **fin si**
11: Genera dato adyacente disminuyendo k en *i-ésimo* atributo
12: **si** dato generado se encuentra en la tabla hash **entonces**
13: Incrementa arreglo A en su posición $2i$+1.
14: **fin si**
15: **fin para**
16: **fin para**
17: **si** Todas las posiciones de arreglo A son diferentes de 0 **entonces**
18: El elemento es descartado
19: **fin si**
20: **fin para**

Capítulo 5

Experimentos

Para comprobar la precisión del método se utiliza un conjunto de datos separados por alguna función especifica. Cada dato se conforma de una colección de números donde el primero indica la clase a la que pertenece y los subsecuentes son duplas de números separadas por el signo de : el primer número de las duplas indica a qué atributo se refiere, mientras que el segundo indica el valor de dicho atributo. En la tabla 5.1 se muestra un ejemplo con datos que tienen dos atributos.

clase	tipo 1	tipo 2
1.0	1:75.0	2:14.0
1.0	1:30.0	2:2.0
1.0	1:61.0	2:37.0
1.0	1:89.0	2:3.0
0.0	1:40.0	2:79.0
0.0	1:24.0	2:76.0
1.0	1:72.0	2:6.0
0.0	1:54.0	2:44.0
0.0	1:24.0	2:96.0
1.0	1:65.0	2:43.0
1.0	1:96.0	2:68.0
0.0	1:2.0	2:42.0

Cuadro 5.1

Con los datos mostrados en la tabla anterior dibujamos en una imagen en 2D, el color depende de la clase a la que pertenece, y las propiedades nos representan la posición en la que se dibuja el píxel.

En la figura 5.1 representamos 40000 datos separados por la siguiente función:

$$y = 20 * sin(\frac{x}{5}) + x$$

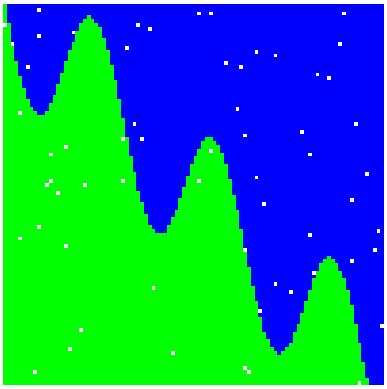


Figura 5.1: Dos clases dibujadas en un mapa 2D

A partir del ejemplo aplicamos mecanismos de detección de bordes para hacer el reconocimiento de las orillas en la imagen. Utilizamos Java para el procesado de las imágenes y aplicamos los filtros adecuados basados en el cambio de tono que existe entre los píxeles. En la figura 5.2 observamos la detección de orillas aplicada a la imagen mostrada anteriormente:

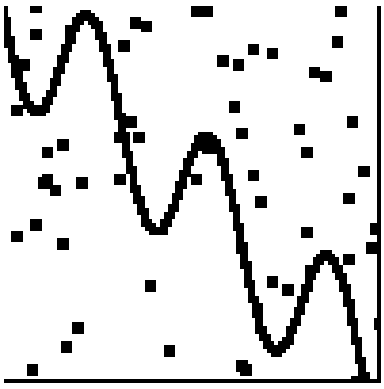


Figura 5.2: Detección de bordes

Una vez construida la imagen con los métodos de detección de bordes, se procesa la cantidad de datos original para filtrar los datos que se encuentran en las orillas, este proceso solo realiza comparaciones por lo que es muy rápido y no necesita muchos cálculos de procesamiento y su complejidad computacional crece linealmente con el número de datos. Dependiendo del conjunto de datos y su dispersión se puede producir un conjunto de datos significativamente más pequeño que el conjunto de datos original, por lo que el entrenamiento en las máquinas de soporte vectorial es mucho más rápido ahorrando una gran cantidad de procesamiento.

5.0.1. Experimentos utilizando las máquinas de soporte vectorial

Como herramienta de desarrollo utilizamos el lenguaje de programación JAVA y la librería LIBSVM [35]

LIBSVM es un software integrado para la clasificación con vectores de soporte,(C-SVC, nu-SVC), problemas de regresión, (epsilon-SVR, nu-SVR), y algoritmos de estimación de distribución. En este trabajo utilizamos las herramientas de clasificación con vectores de soporte, utilizando sus dos funciones principales, *svm-train* y *svm-predict*.
Para entrenar la máquina se debe usar la siguiente llamada:

svm-train [opciones] archivo_datos_entrenamiento [nombre_archivo_modelo]

Las opciones que se pueden utilizar son las siguientes:

-s : tipo de SVM.

0 - C-SVC
1 - nu-SVC
2 - one-class SVM
3 - epsilon-SVR
4 - nu-SVR

-t: kernel. El tipo de kernel con el que se quiere realizar la clasificación.

0 - Lineal: u' *v
1 - Polinomico: (gamma*u'*v + coef0) ∧ grado
2 - Función radial RBF: exp(-gamma*—u-v— ∧ 2)
3 - Función sigmoidea: tanh(gamma*u' *v + coef0)
4 - Kernel definido por el usuario

Existen otros parámetros que se pueden utilizar. En nuestro caso se utilizaron en su mayoría los parámetros por defecto ya que estos brindaban el mejor desempeño. Algunos otros parámetros utilizados son:
-d: Grado para el kernel polinómico. Por defecto 3
-g: Valor gamma de los kernels. El valor por defecto 1/k, donde k es la cantidad de atributos.
-r: Valor del coeficiente, coef0 del kernel. Valor por defecto 0.
-h: Usar la heurística *shrinking*.
-c: Costo o penalidad. Indica el valor de tolerancia a los outliers, a mayor valor menor tolerancia. Valor por defecto 1.
-v: Configura el modo de n cross-validation que consiste en dividir el conjunto de entrenamiento en n subconjuntos de modo tal que en cada iteración se toma un de los subconjuntos para prueba y los restantes n-1 de entrenamiento. Finalmente luego de todas las iteraciones se promedian los porcentajes de reconocimiento alcanzado. Esta opción se usa para probar los parámetros, una vez seleccionados estos, se realiza el entrenamiento sin usar esta opción. Es importante tener en cuenta que los parámetros de configuración de la máquina son gamma y C para todos los kernels y se agrega d para el caso del kernel polinómico.

Cuando se ejecuta la función obtenemos un modelo de predicción que es el que necesitamos para clasificar posteriores datos. Una vez construido el modelo de clasificación se procede a clasificar otro conjunto de datos y obtener la precisión de clasificación para esto se utiliza la función:

svm-predict [opciones] archivo_datos_test nombre_archivo_modelo archivo_resultado

LIBSVM nos presentará en pantalla el porcentaje de reconocimiento obtenido, de esta forma se utilizarán diferentes conjuntos de entrenamiento utilizando el algoritmo de filtración de datos se comprueba que el nuevo conjunto de datos generado por el filtro tiene un desempeño similar aunque el tiempo de entrenamiento es menor.

Ejemplo con dos atributos separados por la función:

$$(x-75)^2+(y-75)^2<6000$$

donde cada propiedad esta en el rango [0,190]

clase	tipo 1	tipo 2
1.0	1:97.0	2:139.0
1.0	1:130.0	2:34.0
1.0	1:145.0	2:102.0
1.0	1:86.0	2:136.0
0.0	1:146.0	2:132.0
1.0	1:86.0	2:42.0
0.0	1:143.0	2:27.0
0.0	1:5.0	2:131.0
1.0	1:110.0	2:26.0
0.0	1:35.0	2:7.0
0.0	1:147.0	2:146.0
1.0	1:143.0	2:39.0
1.0	1:103.0	2:131.0
1.0	1:132.0	2:74.0
1.0	1:17.0	2:31.0
1.0	1:64.0	2:122.0

El conjunto de entrenamiento es de 10000 datos y el conjunto de prueba 2000 para el ejemplo se realiza la siguiente tabla donde obtenemos el mejor valor de profundidad.

Búsqueda	Datos filtrados	Tiempo de procesado
1	31	$<$ 1 segundo
5	2767	$<$ 1 segundo
10	6132	1 segundo
15	6698	1 segundo
20	6240	2 segundos

Se observa que el mejor parámetro de profundidad es 15 ya que filtra más datos de entrenamiento y se procede a realizar las pruebas con SVM

Entrenamiento con 10000 datos Archivo de prueba con 2000 datos
optimización #iteraciones = 25294 Total nSV = 8149 El tiempo de demora es :28.5 segundos Precisión = 50.5 % (1010/2000) (classification)

Cuadro 5.2: Tiempo de entrenamiento y precisión con *kernel RBF*

Entrenamiento con 10000 datos Archivo de prueba con 2000 datos
optimización #iteraciones = 25298 Total nSV = 8192 El tiempo de demora es :30 segundos Precisión = 50.5 % (1010/2000) (classification)

Cuadro 5.3: Tiempo de entrenamiento y precisión con *kernel RBF* y usando la heurística *shrinking*

Entrenamiento con 3302 datos Archivo de prueba con 2000 datos
optimización #iteraciones = 6621 Total nSV = 2817 El tiempo de demora es :.963 segundos Precisión = 48.35 % (967/2000) (classification)

Cuadro 5.4: Tiempo de entrenamiento y precisión con *kernel RBF*

En el caso de más dimensiones tenemos un ejemplo con 3 atributos separados con la función

$$x < z + y$$

clase	tipo 1	tipo 2	tipo 3
1	1:15	2:0	3:19
1	1:1	2:6	3:19
1	1:9	2:13	3:4
1	1:6	2:1	3:11
1	1:7	2:9	3:8
1	1:14	2:16	3:5
0	1:11	2:6	3:0
1	1:8	2:2	3:17
1	1:10	2:8	3:14
1	1:9	2:10	3:16
1	1:6	2:11	3:1
0	1:10	2:7	3:1

Para realizar el filtrado se comienza con una distancia de búsqueda de 2 ya que realizando pruebas se observa que aunque la cantidad de datos sobrepase los cientos de miles el tiempo de filtrado no será mayor de un minuto, si la cantidad de datos discriminados es significativa se procede a realizar las pruebas de evaluación con el entrenamiento en SVM en caso contrario se amplia la búsqueda con distancia 10. Dado que la búsqueda de los bordes se realiza con tablas *hash* la complejidad del algoritmo en el peor de los casos será $O(n)$ es decir en función del número de elementos.
Como conjunto de entrenamiento original se cuenta con 20000 datos con atributos enteros entre 0 y 50, el filtro con una distancia de búsqueda de 2 discrimina solo 5 elementos y tarda menos de 1 segundo en ejecutar, se procede a utilizar una distancia de 10 que discrimina 2658 datos filtrados y su tiempo de procesado son 5 segundos, al ampliar la distancia a 45 el filtro genera 6213 y tarda 15 segundos.

Obtenemos que 6213 no pertenecen a un borde por lo que se descartan y al entrenar a la *SVM* con ambos conjuntos registramos la precisión y el tiempo de computo para clasificar datos. Realizamos 3 entrenamientos utilizando C-SVM, un entrenamiento con los valores por defecto, uno agregando la heurística *shrinking* y el último utilizando los datos obtenidos por el algoritmo de discriminación.

Entrenamiento con 20000 datos Archivo de prueba con 1000 datos
optimización #iteraciones = 50505 Total nSV = 15085 El tiempo de demora es :319 segundos Precisión = 99.4 % (994/1000) (classification)

Cuadro 5.5: Tiempo de entrenamiento y precisión con *kernel RBF*

Entrenamiento con 20000 datos Archivo de prueba con 1000 datos
optimización #iteraciones = 50409 Total nSV = 15162 El tiempo de demora es :298 segundos Precisión = 99.4 % (994/1000) (classification)

Cuadro 5.6: Tiempo de entrenamiento y precisión con *kernel RBF* y usando la heurística *shrinking*

Entrenamiento con 13787 datos Archivo de prueba con 1000 datos
optimización #iteraciones = 31838 Total nSV = 15085 El tiempo de demora es :132segundos Tiempo para encontrar el parámetro y filtrar datos 20 segundos Total de tiempo: 152 Precisión = 99.4 % (994/1000) (classification)

Cuadro 5.7: Tiempo de entrenamiento y precisión con datos de entrenamiento discriminados y *kernel RBF*

5.1. Problema de reconocimiento de piel

Otro ejemplo utilizado para probar el comportamiento del método es el problema de reconocimiento de piel en imágenes, donde se toman muestras sobre un espacio de color RGB (azul,verde,rojo). El repositorio es proporcionado por el centro de inteligencia maquinal y sistemas inteligentes en la universidad de California en Irvin [25]. El formato de los datos es de la forma: (c,b,g,r) donde c es la clasificación a la que pertenece (1 es piel, 2 no es piel), mientras que b, g, r indican la cantidad de saturación que presentan los píxeles en los colores: Azul, Verde, Rojo respectivamente.

El conjunto de datos es una colección de muestras aleatorias obtenidas en el formato B,G,R de varias imágenes con rostros de diferente grupo de personas (niños, jóvenes y ancianos) y de diferentes razas (blancos, negros, asiáticos). El total del tamaño de los datos son 245057, de los cuales 50859 son ejemplos de piel mientras que 194198 son muestras aleatorias que no pertenecen a piel.

Para las pruebas se separó el conjunto total de datos de forma aleatoria donde el conjunto de entrenamiento se conforma por 175057 y para el conjunto de prueba se utilizan 70000(20000 representan piel y 50000 no son piel).

Al realizar diversas pruebas se observa que el *kernel* más adecuado para el problema es el *RBF*, por la precisión y el tiempo que tarda en entrenar la *SVM*. Los tiempos de procesado se describen en los cuadros 5.8 y 5.9 mientras que el tiempo necesario para filtrar los datos fue de 95 segundos.

Entrenamiento con 175057 datos Archivo de prueba con 70000 datos
optimización #iteraciones = 101137 Total nSV = 36238 El tiempo de demora es :2733 segundos Precisión = 85.24714285714285 % (59673/70000) (classification)

Cuadro 5.8: Tiempo de entrenamiento y precisión con *kernel RBF*

Entrenamiento con 146153 datos Archivo de prueba con 70000 datos
optimización #iteraciones = 94483 Total nSV = 35092 El tiempo de demora es :2231 segundos Tiempo para encontrar el parámetro y filtrar datos 95 segundos Total de tiempo: 2326 segundos Precisión = 85.21142857142857 % (59648/70000) (classification)

Cuadro 5.9: Tiempo de entrenamiento y precisión con *kernel RBF*

Capítulo 6

Conclusiones

Las *SVM* han demostrado ser herramientas muy útiles para la clasificación de datos y su uso se ha propagado en los últimos años, gracias a su buen desempeño y capacidad de generalización. Por el hecho de que son algoritmos de aprendizaje supervisado, necesitan crear una función a partir de datos de prueba con la que será capaz de predecir la clase de datos solo observando sus atributos.

Esta técnica de clasificación utiliza los atributos de los objetos para posicionarlos en un espacio n-dimensional y trata de construir un hiperplano que separe de manera óptima dos diferentes tipos de clases, esto lo logra maximizando las distancias que existen entre el hiperplano y cada cada uno de los datos representados en el plano con sus atributos. En caso de no ser posible separar clases, las *SVM* utilizan una función *kernel* que transforme el espacio de características a una dimensionalidad superior y así poder construir el hiperplano óptimo.

Uno de los inconvenientes de las *SVM* son los tiempos de aprendizaje que dependen en gran medida del tamaño del conjunto de entrenamiento, ésta es una gran limitante por el tiempo de cálculo que lleva construir la función cuando el conjunto de entrenamiento es muy grande.

En este trabajo proponemos una técnica que permite filtrar los datos más significativos para la construcción de las funciones clasificadoras, este método se diseño con la idea de considerar los datos que se encuentran en los bordes y de esta manera sean los datos más cercanos al hiperplano construido.

Los resultados obtenidos demuestran que la técnica es muy eficiente cuando se utiliza *kernel* radial, sin embargo si se utiliza otro tipo de función la precisión baja drásticamente e inclusive el tiempo de entrenamiento puede crecer con una cantidad menor de datos.

6.0.1. Trabajo futuro

A continuación se señalan algunos temas para continuar la investigación como trabajo futuro:

1. Dentro del trabajo se utilizaron los kernels proporcionados por la herramienta *libsvm* (lineal, polinomial, radial y sigmoide). Es recomendable hacer una investigación utilizando más kernels y decidir si existen mejoras en el rendimiento utilizando el filtro investigado en este trabajo.
2. La cantidad de ejemplos fue muy limitada, se recomienda estudiar más cada caso particular y la eficacia con que la *SVM* filtra los datos y la repercusión en los tiempos de entrenamiento y precisión para predecir datos.
3. Varios ejemplos utilizados en la investigación necesitaron ser transformados y normalizados para ocuparse de manera adecuada en el filtro, se necesita tener un estudio más profundo en este campo para determinar si estas transformaciones no conllevan a arrastrar posibles incoherencias en el uso de esas bases de datos.
4. Se necesita probar el filtro con otro tipo de herramientas que proporcionen manejo de máquinas de soporte vectorial y comparar los resultados con los expuestos en este documento.

Bibliografía

[1] Vladimir Vapnik, Corinna Cortes, *Support-Vector Networks*, Hohndel, NJ 07733, USA. 1995.

[2] Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education 2003.

[3] John Haugeland. *Artificial Intelligence The Very Idea*. Cambridge: MIT Press. 1985.

[4] Richard Bellman. *An introduction to artificial intelligence: Can computers think?*. 1978.

[5] Charniak, Eugene and McDermott, Drew. *Introduction to Artificial Intelligence*,Boston, MA, USA. 1985.

[6] Patrick Henry Winston,*Artificial Intelligence*, Addison-Wesley Publishing Company, 1992.

[7] Ray Kurzweil, *The age of intelligent machines*, MIT Press, 1990.

[8] Kevin Knight,Elaine Rich. *Artificial Intelligence. the University of Michigan*, 1991.

[9] Robert J. Schalkoff. *Artificial Intelligence: An Engineering Approach*, the University of Michigan. 1990.

[10] George F. Luger, William A. Stubblefield. *Artificial intelligence: structures and strategies for complex problem solving*. the University of Michigan. 1993.

[11] J.R, Quinlan. *Induction of Decision Trees, Centre for Advanced Computing Sciences*, New South Wales Institute of Technology, Sydney 2007, Australia (1985).

[12] E. Fix y JL Hodges, *An important contribution to nonparametric discriminant analysis and density estimation*, Commentary on Fix and Hodges, 1951.

[13] Nir Friedman, Dan Geigar, Moises Goldszmidt, *Bayesian Network Classifiers*. Kluwer Academic Publishers (1997).

[14] Richard P. Lippman, *An introduction to Computing with Neural Nets*. IEEE (1987).

[15] Robert Hecht-Nielsen. *Theory of the Backpropagation Neural Network*, University of Califomia at San Diego. 1989.

[16] Breiman, L., Friedman, J. H., Olshen, R. A., Stone, P. J. (1984). *Classification and regression trees. Beltmon*, CA: Wadsworth International Group.

[17] Mitchell, T. M. *Machine learning*. McGraw-Hill, (1997).

[18] Burges, C. J. C., *A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery*, http://svm.research.belllabs.com/SVMdoc.html, (1998).

[19] Osuna, E., Freund, R., Girosi, F., *Improved Training Algorithm for Support Vector Machines*, Proc. IEEE NNSP ?97, (1997).

[20] Johan A. K. Suykens, Tony Van Gestel, Jos De Brabanter, Bart De Moor and Joos Vandewalle. *Least Square Support Vector Machines*. World Scientific, in press (ISBN 981-238-151-1)

[21] Ying Wang. Machine Learning, Exercise 8: Non-linear SVM classification with kernels. http://openclassroom.stanford.edu/MainFolder/DocumentPage.php?course=MachineLearning &doc=exercises/ex8/ex8.html

[22] Di You, Onur C. Hamsici, and Aleix M. Martinez, *Kernel Optimization in Discriminant Analysis*, IEEE Transaction on pattern analysis and machine intelligence, Vol. 33, No. 3, March 2011.

[23] Muhammad Hussain, Summrina Kanwal Wajid, et al. *A Comparison of SVM Kernel Functions for Breast Cancer Detection* Eighth International Conference Computer Graphics, Imaging and Visualization, 2011

[24] Jair Cervantes Canales, *Clasificación de grandes conjuntos de datos vía Máquinas de Vectores Soporte y aplicaciones en sistemas biológicos*. cinvestav, 2009.

[25] Departamento de Matemáticas, CSI/ITESM. Condiciones de Karush-Kuhn-Tucker,21 de abril de 2010.

[26] Eric Sung, Zhu Yan and Li Xuchun. *Accelerating the SVM Learning for Very Large Data set*. In School of Electrical and Electronic Engineering, Nayang Technological University.

[27] Hwanjo Yu, Jiong Yang, Jiawei Han. *Classifying Large Data Sets Using SVMs with Hierarchical Clusters*. Departament of computer Science University of Illinois.

[28] P. Kumar, J. S. B. Mitchell, and E. A. Yildirim, *Approximate minimum enclosing balls in high dimensions using core-sets*, J. Exp. Algorithmics 8 (2003), 1.1. Available at http://www.compgeom.com/piyush/meb/journal.pdf.

[29] Kuhn, H. W.; Tucker, A. W. *Nonlinear Programming*. Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability, 481–492, University of California Press, Berkeley, Calif., 1951.https://projecteuclid.org/euclid.bsmsp/1200500249

[30] Platt J., *Fast Training of support vector machine using sequential minimal optimization*. In A.S.B. Scholkopf, C. Burges, editor, Advances in Kernel Methods: support vector machine . MIT Press, Cambridge, MA 1998

[31] John C. Platt, *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines*. Technical Report, Microsoft Research. Abril 21, 1998.

[32] Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[33] Zhi-Qiang Zeng, Hua-Rong Xu, Yan-Qi Xie, *A Geometric Approach to Train SVM on Very Large Data Sets*. 3rd International Conference on Intelligent System and Knowledge Engineering 2008.

[34] Nianyi Chen et al. *Support Vector Machine in Chemistry*. World Cientific, Shanghai Jiao Tong University, China. Agosto 2004.

[35] Chih-Chung Chang and Chih-Jen Lin, *LIBSVM : a library for support vector machines*. ACM Transactions on Intelligent Systems and Technology, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm

[36] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. *A Practical Guide to Support Vector Classification*. Department of Computer Science, April 15, 2010.

[37] Edge detection Trucco, Chapter 4 and Jain et al. Chapter 5. https://www.cse.unr.edu/ bebis/CS791E/Notes/EdgeDetection.pdf. 2016.

[38] John Canny. *A Computational Approach to Edge Detection*. IEEE Transactions on pattern analysis and machine intelligence, vol. PAMI-8, No. 6, November 1986.

[39] Rajen Bhatt, Abhinav Dhall, 'Skin Segmentation Dataset', UCI Machine Learning Repository.

Casa abierta al tiempo
UNIVERSIDAD AUTÓNOMA METROPOLITANA

ACTA DE EXAMEN DE GRADO

No. 00068
Matrícula: 210383509

DISCRIMINACIÓN DE DATOS EN MÁQUINA DE SOPORTE VECTORIAL

En la Ciudad de México, se presentaron a las 11:00 horas del día 5 del mes de enero del año 2018 en la Unidad Iztapalapa de la Universidad Autónoma Metropolitana, los suscritos miembros del jurado:

DRA. ALICIA MORALES REYES
MTRO. OSCAR YAÑEZ SUAREZ
DR. RENE MAC KINNEY ROMERO

Bajo la Presidencia de la primera y con carácter de Secretario el último, se reunieron para proceder al Examen de Grado cuya denominación aparece al margen, para la obtención del grado de:

MAESTRO EN CIENCIAS (CIENCIAS Y TECNOLOGIAS DE LA INFORMACION)

DE: GUILLERMO GONZALEZ TORRES

y de acuerdo con el artículo 78 fracción III del Reglamento de Estudios Superiores de la Universidad Autónoma Metropolitana, los miembros del jurado resolvieron:

Aprobar

Acto continuo, la presidenta del jurado comunicó al interesado el resultado de la evaluación y, en caso aprobatorio, le fue tomada la protesta.

GUILLERMO GONZALEZ TORRES
ALUMNO

REVISÓ

LIC. JULIO CESAR DE LARA ISASSI
DIRECTOR DE SISTEMAS ESCOLARES

DIRECTOR DE LA DIVISIÓN DE CBI

DR. JOSE GILBERTO CORDOBA HERRERA

PRESIDENTA

DRA. ALICIA MORALES REYES

VOCAL

MTRO. OSCAR YAÑEZ SUAREZ

SECRETARIO

DR. RENE MAC KINNEY ROMERO