Presentación Final Aprendizaje Estadístico

Julian Ferres Matias Iglesias Matias Scakosky Luciano Sportelli Castro

18 de marzo de 2019

Resumen

En este trabajo se realizará un análisis del artículo A Tutorial on Support Vector Machines for Pattern Recognition, por Chistopher J.C. Burges.

Índice

1. Introducción

Luego de cursar la asignatura *Aprendizaje Estadístico* en el 2^{do} cuatrimestre de 2018, materia dictada por el Dr. Sebastian Grymberg, se presentó la posibilidad de tomar un paper relacionado a los temas vistos en la materia y poder desarrollarlo, pudiendo, de esta manera exponer nuestro conocimientos adquiridos a lo largo de todo un cuatrimestre. En la materia se exponen dos grandes problemas, el problema de clasificación y el problema de regresión. Debido a la magnitud (en cantidad de contenidos) que presenta cada uno de estos tópicos, es imposible poder abarcar de forma completa estos dos problemas a lo largo de un solo cuatrimestre, pero gracias al pantallazo general que tuvimos, cualquier tema relacionado con esta materia no debería presentar ningún inconveniente a la hora de intentar leerlo y entenderlo.

Si bien dentro de las opciones que nos dio el docente para poder hacer un trabajo práctico que de un cierre a la materia, estaba la posibilidad de poder analizar un paper de algún tema visto en la materia y poder explicarlo, nosotros decidimos tomar un paper de un tema que esta fuera del programa de la materia, ya que consideramos que es un buen desafío tener la posibilidad de analizar cuestiones relacionadas al aprendizaje automático a raíz de conceptos vistos en la materia, pero con temas que no fueron explícitamente dados en clase

Para esto nos comunicamos con otro docente de la facultad, el profesor Luis Argerich docente titular a cargo de la materia 75.06 Organización de Datos. Quienes redactamos este informe fuimos sus alumnos y nos pareció interesante la posibilidad de vincular a los departamentos de Matemática y Computación de la Facultad de Ingeniería, a través de solicitarle una recomendación acerca de algún material relacionado a su materia, la cual esta completamente avocada a los temas de Aprendizaje Estadístico.

Luego de leer el material recomendado, acudimos a quien tiene la ultima palabra en cuestiones de lo que refiere esta materia, el Dr. Grymberg para dentro de todas las opciones posibles, definir hacia donde terminamos de encarar nuestro trabajo práctico final de la materia. En una reunión que tuvimos personalmente con el docente Sebastián, terminamos por inclinarnos hacia el paper: **A Tutorial on Support Vector Machines for Pattern Recognition, by Chistopher J.C. Burges.** En las siguientes páginas haremos un análisis profundo, no solo de los temas que el paper plantea de como usar SVM y su implementación junto con sus fundamentos matemáticos, sino que también analizaremos el rendimiento y las limitaciones del mismo.
Una vez analizado como funciona SVM dedicaremos una sección de este trabajo práctico a su uso en la actualidad, ¿Para que se puede utilizar hoy SVM? ¿Es un algoritmo que funciona dentro de los parámetros esperables de aceptación? Descubriremos dentro de la misma sección el porqué hoy quedó en desuso y que tipo de algoritmos le ganaron en popularidad y en resultados pragmáticos.

Al final del trabajo mostraremos ejemplos de aplicación práctica del mismo, en el cual se podrá poner bajo análisis cuestiones relacionadas a como funciona SVM, que tan buen clasificador es, y si las cuentas previamente realizadas en el informe se corresponden con lo que termina sucediendo en la realidad del clasificador. Para ello utilizaremos diferentes recursos informáticos de librerías de Python como `sklearn` y sus diversas bibliotecas especializadas para SVM.

Quien desee acceder a toda la información en la que nos basamos para realizar el informe, el paper base que dio origen al trabajo, el código utilizado en los ejemplos prácticos mencionados anteriormente, puestos en formato "Notebook"de JupyerLab y también por ultimo a la versión digital del informe. Dejaremos de forma pública el acceso al repositorio de GitHub en el siguiente link:

https://github.com/julianferres/Aprendizaje-Estadistico

2. Conceptos previos a SVM

Esta sección es encargada de revisar y generar una base sólida acerca del tema en exposición. Una vez vista la idea general de SVM y antes de involucrarse profundamente en los aspectos matemáticos mas duros del algoritmo y posterior a eso su implementación, debemos entender que SVM surge como la mejor solución a un problema previo, que es el algoritmo de Perceptrón.

2.1. Perceptrón

En el campo de las Redes Neuronales, el perceptrón, creado por Frank Rosenblatt, es un algoritmo de clasificación lineal y la neurona artificial (también conocida como unidad básica) a base de un discriminador lineal, desarrolla un algoritmo capaz de generar un criterio para seleccionar un sub-grupo a partir de un grupo de ciertos inputs. Una neurona responde si 0 o si 1 según el resultado de la combinación lineal fuese mayor o menor a un cierto umbral.

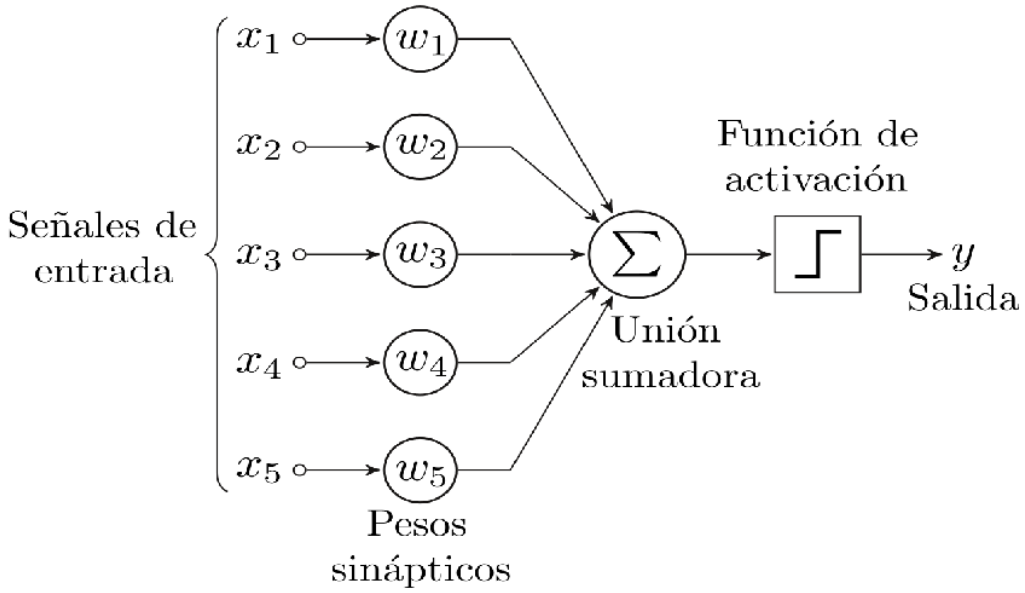

Figura 1: Ejemplo de Perceptrón

En esta figura se puede observar la idea básica de una neurona para perceptrón, donde llegan diferentes pesos y luego de unirlos mediante un mecanismo que explicaremos mas adelante se aplica una función matemática llamada función de activación que devuelve un valor binario que termina de clasificar. El perceptrón puede utilizarse con otros tipos de perceptrones o de neurona artificial, para formar una red neuronal artificial más compleja. Una neurona sola no tiene ningún sentido a la hora de querer hacer alguna clasificación. Su labor especializada se torna valiosa en la medida en que se asocia a otras neuronas, formando una red. Normalmente, la salida de una neurona entrega su información como entrada a los pesos de otra neurona y así sucesivamente. El perceptrón que capta la señal en adelante se extiende formando una red de neuronas.

¿Como se hace la unión sumadora que se muestra en el en la Figura 1? Es extremadamente sencillo, ya que esto junto con la función de activación son todo el secreto de perceptrón. Supongamos ω un vector de pesos reales y $\vec{x}$ un vector n-dimensional con valores de entrada, para cada uno de los vectores de entrada debo calcular el producto interno $x \cdot \omega$. Suponiendo la siguiente función de activación

$$f(\vec{x}) = \begin{cases} 1 \quad si \quad \sum x \cdot \omega \geq 0 \\ \\ 0 \quad si \quad otrocaso \end{cases}$$

El algoritmo clasificara según cada elemento del vector sumándolo con el del anterior hasta llegar a un resultado final ¿Que pasa si el vector ω no es correcto? Pues habrá que aplicar una pequeña corrección al mismo y volver a empezar. La corrección se realiza como $\omega = \omega + x * y$ Para mejorar el algortimo se

puede aplicar un pequeño factor de aprendizaje α de manera que se vaya actualizando el vector ω de forma gradual. La actualización quedaria de forma $\omega = \omega + \alpha * x * y$

El algoritmo final de perceptrón resultaría así.

Algorithm 1: Perceptrón

Result: Clasifica una serie de puntos
initialization;
while *not fin* **do**
  tomo un punto x_i y su clase y_i
  if $f(x \cdot \omega)$!= y **then**
    $\omega = \omega + \alpha * x * y$
  else
    continue
  end
end

Se podría simular el siguiente ejemplo.

```
umbral = 0.5
tasa_de_aprendizaje = 0.1
pesos = [0, 0, 0]
conjunto_de_entrenamiento = [((1, 0, 0), 1), ((1, 0, 1), 1), ((1, 1, 0), 1), ((1, 1, 1), 0)]

def producto_punto(valores, pesos):
    return sum(valor * peso for valor, peso in zip(valores, pesos))

while True:
    print('-' * 60)
    contador_de_errores = 0
    for vector_de_entrada, salida_deseada in conjunto_de_entrenamiento:
        print(pesos)
        resultado = producto_punto(vector_de_entrada, pesos) > umbral
        error = salida_deseada - resultado
        if error != 0:
            contador_de_errores += 1
            for indice, valor in enumerate(vector_de_entrada):
                pesos[indice] += tasa_de_aprendizaje * error * valor
    if contador_de_errores == 0:
        break
```

Podremos obtener y hacer objeto de estudio los siguientes resultados de out del algoritmo.

```
------------------------------------------------------------
[0, 0, 0]
[0.1, 0.0, 0.0]
[0.2, 0.0, 0.1]
[0.30000000000000004, 0.1, 0.1]
------------------------------------------------------------
[0.30000000000000004, 0.1, 0.1]
[0.4, 0.1, 0.1]
[0.5, 0.1, 0.2]
[0.5, 0.1, 0.2]
------------------------------------------------------------
[0.4, 0.0, 0.1]
[0.5, 0.0, 0.1]
[0.5, 0.0, 0.1]
[0.6, 0.1, 0.1]
------------------------------------------------------------
[0.5, 0.0, 0.0]
[0.6, 0.0, 0.0]
```

```
[0.6, 0.0, 0.0]
[0.6, 0.0, 0.0]
------------------------------------------------------------
[0.5, -0.1, -0.1]
[0.6, -0.1, -0.1]
[0.7, -0.1, 0.0]
[0.7, -0.1, 0.0]
------------------------------------------------------------
[0.6, -0.2, -0.1]
[0.6, -0.2, -0.1]
[0.7, -0.2, 0.0]
[0.7999999999999999, -0.1, 0.0]
------------------------------------------------------------
[0.7, -0.2, -0.1]
[0.7, -0.2, -0.1]
[0.7, -0.2, -0.1]
[0.7999999999999999, -0.1, -0.1]
------------------------------------------------------------
[0.7, -0.2, -0.2]
[0.7, -0.2, -0.2]
[0.7999999999999999, -0.2, -0.1]
[0.7999999999999999, -0.2, -0.1]
------------------------------------------------------------
[0.7999999999999999, -0.2, -0.1]
[0.7999999999999999, -0.2, -0.1]
[0.7999999999999999, -0.2, -0.1]
[0.7999999999999999, -0.2, -0.1]
```

2.2. La mejor solución

Habiendo entendido un poco la idea de perceptrón básicamente en todas las explicaciones y ejemplos de esta sección del informe fueron meramente una preparación para poder dilucidar cual es el objetivo principal del algoritmo. Si tenemos un conjunto de puntos en un plano, supongamos un plano real, y cada uno de esos conjuntos de puntos están clasificados de una forma binaria, es decir, dos clases, perceptrón intenta ser una solución de una manera muy ingeniosa. Si las clases están separadas de manera tal que se puede definir un hiper-plano separador, perceptrón hallará ese hiper-plano que resultará del vector ω anteriormente mencionado. Pero como a todo matemático es imposible no preguntarse. El hiper-plano hallado ¿Es único? La respuesta es que no, pueden haber infinitos hiper-planos que separen a las clases. Entonces por razonamiento inductivo podemos decir que si no es lo mismo el hiper-plano que elijamos a la hora de separar las clases, alguno necesariamente tiene que ser mejor que otro. Entonces ¿Cual es el **mejor** de los hiper-planos separadores? He aquí que entra en juego el objeto de estudio de este informe. Ya que SVM se trata de encontrar el mejor hiperplano que separe las clases. Toda la teoría posteriormente desarrollada, esta encargada de explicar por qué ese hiperplano hallado es el mejor y de que manera podemos encontrarlo, mas allá de todas las técnicas diferentes que se pueden aplicar sobre el algoritmo de SVM.

3. Idea general

Las *Support Vector Machines*, también conocidas por su traducción al español máquinas de vectores de soporte o máquinas de vector soporte (SVMs) son un conjunto de algoritmos de `aprendizaje supervisado` [1] desarrollados por Vladimir Vapnik, Alexey Ya. Chervonenkis y su equipo en los laboratorios AT&T a finales de 1963. Recien en 1992, Bernhard E. Boser, Isabelle M. Guyon y Vladimir N. Vapnik

[1] En aprendizaje automático y minería de datos, el aprendizaje supervisado es una técnica para deducir una función a partir de datos de entrenamiento. Los datos de entrenamiento consisten de pares de objetos (normalmente vectores): una componente del par son los datos de entrada y el otro, los resultados deseados. El objetivo del aprendizaje supervisado es el de crear una función capaz de predecir el valor correspondiente a cualquier objeto de entrada válida después de haber visto una serie de ejemplos, los datos de entrenamiento. Para ello, tiene que generalizar a partir de los datos presentados a las situaciones no vistas previamente.

sugirieron una manera de crear clasificadores no lineales aplicando *the kernel trick* a los hiperplanos de margen máximo. La actual corriente de *Soft Margin* fue propuesta por Corinna Cortes y Vapnik en 1993 y publicada en 1995.

Estos métodos están propiamente relacionados con problemas de clasificación y regresión. Dado un conjunto de ejemplos de entrenamiento (de muestras) podemos etiquetar las clases y entrenar una SVM para construir un modelo que prediga la clase de una nueva muestra. Es decir, que si tenemos un vector de muestra (Con la mera funcionalidad de entrenamiento) cada uno marcado como perteneciente a una u otra de dos categorías, un algoritmo de entrenamiento SVM construye un modelo que asigna nuevos ejemplos a esas mismas categorías, lo que lo convierte en un clasificador lineal binario no probabilístico [2]

Intuitivamente, una SVM es un modelo que representa a los puntos de muestra en el espacio, separando las clases a 2 espacios lo más amplios posibles mediante un hiperplano de separación definido como el vector entre los 2 puntos, de las 2 clases, más cercanos al que se llama vector soporte. Cuando las nuevas muestras se ponen en correspondencia con dicho modelo, en función de los espacios a los que pertenezcan, pueden ser clasificadas a una o la otra clase.

Además de realizar una clasificación lineal , las SVM pueden realizar una clasificación no lineal de manera eficiente utilizando lo que se llama *The kernel trick*, mapeando implícitamente sus entradas en espacios de características de alta dimensión. Esto será explicado de forma detallada en la sección 5.3 del correspondiente informe.

Para intentar entender como funciona SVM, supongamos que algunos puntos de datos dados pertenecen a una de dos clases, y el objetivo es decidir en qué clase estará un nuevo punto de target. En el caso de las SVMs, un punto de target se ve como un vector p-dimensional, y queremos saber si podemos serpararlo con un hiper plano de $(p-1)$ dimensiones. Con este criterio y como analizamos antes con perceptrón hay muchos hiper planos que pueden separar a las clases, y como también dijimos antes, hay uno que es el mejor, unia idea intuitiva de cual es el mejor es el que produzca la mayor separación (o **margen**) entre dos clases.

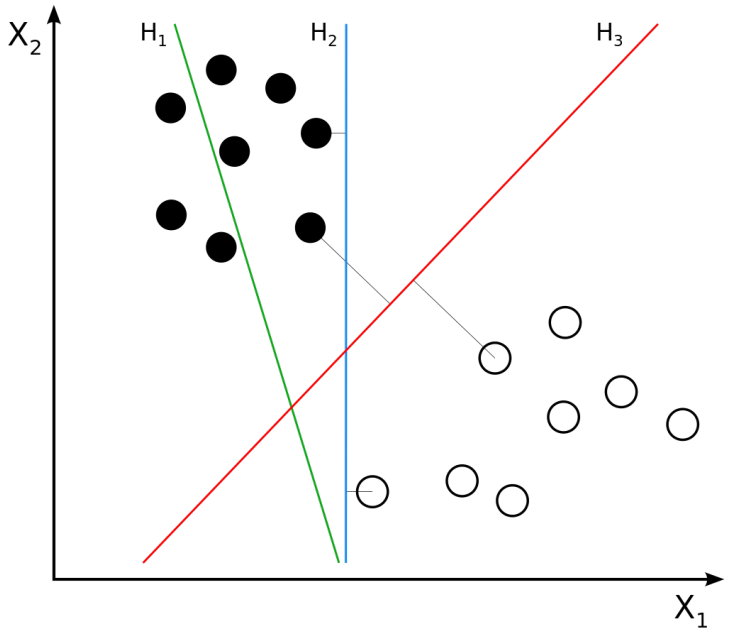

Figura 2: Ejemplo de hiper planos separadores

Si analizamos la Figura 2 podremos notar que H_1 no separa las clases, H_2 y H_3 si lo hacen, pero H_3 controla mejor el margen entre las dos clases, generando una mejor separación entre las mismas. De esta forma elegimos el hiper plano para maximizar la distancia desde él mismo hasta el punto entrenamiento más cercano en cada lado. Si existe un hiper plano tal, se conoce como el hiper plano y al clasificador lineal que lo define como de "máximo margen".

[2] Aunque los métodos como la escala o calibración de Platt (Que es una forma de transformar los resultados de un modelo de clasificación en una distribución de probabilidad sobre clases) existen para utilizar SVM en una configuración de clasificación probabilística

3.1. Usos

Los SVM pueden usarse para resolver varios problemas a los cuales hoy se enfrentan los data scientist:

- Detección de cara : SVM clasifica partes de la imagen como cara y no cara y crea un límite cuadrado alrededor del rostro.
  Clasifica las partes de la imagen como cara y no cara. Contiene datos de entrenamiento de $n \cdot n$ píxeles con cara de dos clases (+1) y sin cara (-1). Luego extrae características de cada píxel como cara o no cara. Crea un límite cuadrado alrededor de las caras sobre la base del brillo de los píxeles y clasifica cada imagen utilizando el mismo proceso.

- Los SVM son útiles en la categorización de texto e hipertexto: Las SVM permiten la categorización de texto e hipertexto para modelos inductivos y transductivos. Utilizan los datos de entrenamiento para clasificar documentos en diferentes categorías. Se categoriza en base a la puntuación generada y luego se compara con el valor del umbral.
  Utiliza datos de capacitación para clasificar documentos en diferentes categorías, como artículos de noticias, correos electrónicos y páginas web.
  Ejemplos:
  - Clasificación de artículos periodísticos en "negocios" y "películas".
  - Clasificación de páginas web en páginas personales y otras

  Para cada documento, calcula una puntuación y la compara con un valor de umbral predefinido. Cuando la puntuación de un documento supera el valor de umbral, el documento se clasifica en una categoría definida. Si no supera el valor de umbral, considérelo como un documento general. Clasifica las nuevas instancias calculando la puntuación de cada documento y comparándola con el umbral aprendido.

- La clasificación de imágenes también se puede realizar utilizando SVMs. Los resultados experimentales muestran que los SVM logran una precisión de búsqueda significativamente mayor que los esquemas tradicionales de refinamiento de consultas después de solo tres o cuatro rondas de comentarios relevantes. Esto también es válido para los sistemas de segmentación de imágenes, incluidos aquellos que utilizan una versión modificada de SVM que utiliza el enfoque privilegiado como lo sugiere Vapnik.

- Bioinformática : incluye clasificación de proteínas y clasificación de cáncer. Utilizamos SVM para identificar la clasificación de genes, pacientes en base a genes y otros problemas biológicos. Pliegue de proteínas y detección de homología remota: aplica algoritmos SVM para la detección de homología remota de proteínas. En el campo de la biología computacional, la detección de homología remota de proteínas es un problema común. El método más efectivo para resolver este problema es usar SVM. En los últimos años, los algoritmos SVM se han aplicado extensivamente para la detección de homología remota de proteínas. Estos algoritmos se han utilizado ampliamente para identificar secuencias biológicas.

- Control predictivo generalizado (GPC): use GPC basado en SVM para controlar la dinámica caótica con parámetros útiles para estabilizar algun objetivo.

- Los caracteres escritos a mano se pueden reconocer usando SVM.

- El algoritmo SVM se ha aplicado ampliamente en las ciencias biológicas y otras. Se han utilizado para clasificar proteínas con hasta un 90 % de los compuestos clasificados correctamente. Se han sugerido pruebas de permutación basadas en pesos SVM como un mecanismo para la interpretación de los modelos SVM. Los pesos de las *Support vector machines* también se han utilizado para interpretar modelos SVM en el pasado. La interpretación posthoc de modelos de SVMs para identificar características utilizadas por el modelo para hacer predicciones es un área de investigación relativamente nueva con especial importancia en las ciencias biológicas.

3.2. Actualidad

4. Fundamentos matemáticos en SVM

4.1. El caso separable

Vamos a comenzar con el caso mas simple: maquinas lineales entrenadas en datos separables (veremos mas adelante que el análisis para el caso general -maquinas no lineales entrenados en datos no separables- resultara en un problema muy similar de orden cuadrático).

Como datos de entrenamiento usaremos $\{x_i, y_i\}$ donde x_i e y_i son los datos de entrenamiento y los labels respectivamente, con $i = 1, \cdots, l$, $y_i \in \{-1, 1\}$, $x_i \in \mathbb{R}^d$

Supongamos que conocemos algun hiperplano que separe los ejemplos con label positivo de los que tienen label negativo (recibira el nombre de *hiperplano separador*)

Los puntos $\mathbf{w}$ que yacen en dicho hiperplano satisfacen que $\mathbf{w} \cdot \mathbf{x} + b = 0$, donde $\mathbf{w}$ es normal al hiperplano, $\frac{|b|}{||\mathbf{w}||}$ es la distancia perpendicular desde el hiperplano al origen, y $||\mathbf{w}||$ es la norma Euclideana de $\mathbf{w}$.

Vamos a definir como $d_+(d_-)$ sea la distancia mínima del hiperplano separador al punto más cercano, positivo (negativo). Definimos tambien como *margen* del hiperplano separador como $d_+ + d_-$. Para el caso linealmente separable, la Support Vector Machine simplemente busca el hiperplano separador con el mayor *margen*.

Esto puede ser formulado de la siguiente manera: supongamos que todos los datos de entrenamiento satisfacen las siguientes restricciones:

$$\begin{array}{lll} \mathbf{x}_i \cdot \mathbf{w} + b \geq +1 & para & y_i = +1 \\ \mathbf{x}_i \cdot \mathbf{w} + b \leq -1 & para & y_i = -1 \end{array}$$

Estas ecuaciones pueden ser combinadas en el sistema de desigualdades:

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0 \ \forall i$$

Ahora consideremos los puntos en los cuales se satisface la igualdad en $\mathbf{x}_i \cdot \mathbf{w} + b \geq +1$ se satisface (requerir que existe algún punto es equivalente a elegir una escala para $\mathbf{w}$ y b). Estos puntos yacen en el hiperplano $H_1 : \mathbf{x}_i \cdot \mathbf{w} + b = 1$ con normal $\mathbf{w}$ y distancia perpendicular al origen $\frac{|1-b|}{||\mathbf{w}||}$. Similarmente los puntos que satisfacen $\mathbf{x}_i \cdot \mathbf{w} + b \leq -1$ yacen en el hiperplano H_2: $\mathbf{x}_i \cdot \mathbf{w} + b = -1$, nuevamente con normal $\mathbf{w}$ y distancia perpendicular al origen $\frac{|-1-b|}{||\mathbf{w}||}$.

Claramente como $d_+ = d_- = 1/||\mathbf{w}||$, el margen es $2/||\mathbf{w}||$. Notar que H_1 y H_2 son paralelos (tienen la misma normal $\mathbf{w}$) y que ningún punto se encuentra entre ellos. Entonces es posible encontrar el par de hiperplanos que nos da el mayor margen minimizando $||\mathbf{w}||^2$, sujeto a las restricciones $(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0 \ \forall i$.

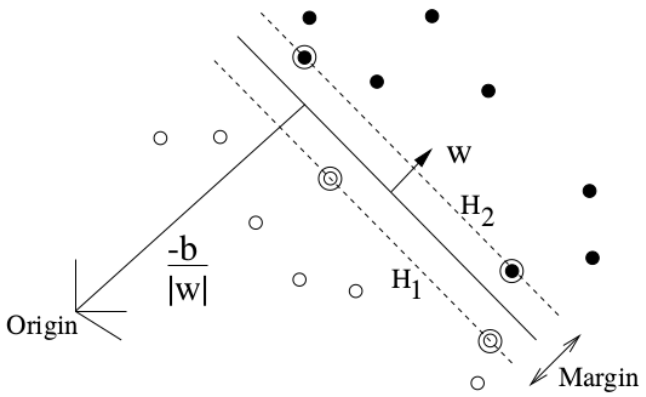

Figura 3: Hiperplanos separadores para el caso separable. Los support vectors tienen un contorno adicional.

Cambiamos ahora a la formulación Lagrangiana del problema. Hay dos razones para hacer esto. La primera es que las restricciones impuestas hasta ahora serán reemplazadas por restricciones en los multiplicadores de Lagrange en si mismos, que son mucho más sencillos de manejar. La segunda, y quizás la más significativa, es que en esta reformulación del problema, los datos de entrenamiento solo aparecerán (en los algoritmos de entrenamiento y testing) en la forma de productos escalares entre vectores. Esto es una propiedad que nos permitirá generalizar el procedimiento al caso no lineal.

Para esto introducimos multiplicadores de Lagrange α_i , $i = 1, \cdots, l$, uno para cada desigualdad de restricción.

Esto nos da el Lagrangiano:

$$L_P = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{l} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{w} + b) + \sum_{i=1}^{l} \alpha_i$$

Nuestro objetivo es minimizar L_P con respecto a $\mathbf{w}$ y b, y simultaneamente exigir que las derivadas de L_P con respecto a todos los α_i se anulen, a su vez sujeto a que $\alpha_i \geq 0$ (llamamos a este set de resticciones en particular con el nombre de $\mathcal{C}_1$). Observemos que este es un problema de programación cuadratica de tipo convexo, ya que la función objetivo es en sí misma convexa, y los puntos que satisfacen las restricciones también forman un conjunto convexo (cualquier restricción lineal define un conjunto convexo y un conjunto de N restricciones lineales simultáneas definen la intersección de N conjuntos convexos, que también es un conjunto convexo). Esto significa que podemos resolver de manera equivalente el siguiente problema "dual": "*maximizar* L_P, sujeto a las restricciones que presenta el gradiente de L_P con respecto a w y b (deben anularse las derivadas con respecto a esas variables), y sujeto también a las restricciones que $\alpha_i \geq 0$ (llamemos a ese conjunto particular de restricciones $\mathcal{C}_2$). Esta formulación dual particular del problema se llama la dual de Wolfe. La misma tiene la propiedad de que el máximo de L_P, sujeto a restricciones $\mathcal{C}_2$, se produce en los mismos valores de $\mathbf{w}$, b y α que el mínimo de L_P sujeto a restricciones $\mathcal{C}_1$.

Exigir que las derivadas parciales de L_P con respecto a $\mathbf{w}$ y b desaparezca da las condiciones:

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

$$\sum_i \alpha_i y_i = 0$$

Dado que estas condiciones son impuestas en la formulación dual, podemos substituirlas en la expresión de L_P para obtener:

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$

Notar que hemos asignado diferentes letras (P por pirmal, D por dual) paara enfatizar que las dos formulaciones son diferentes: L_P y L_D surgen de la misma función objetivo pero con diferentes restricciones, y la solución se hallará maximizando L_P o minimizando L_D.

Una aclaración interesante es que si formulamos el problema con $b = 0$, que exige que todos los hiperplanos pasen por el origen, entonces la restricción $\sum_i \alpha_i y_i = 0$ desaparece.

Observemos también que hay un multiplicador de Lagrange α_i para cada punto de entrenamiento. En la solución, los puntos para los cuales $\alpha_i > 0$ se denominan **vectores de soporte** y se encuentran en uno de los hiperplanos H_1, H_2. Todos los demás puntos de entrenamiento tienen $\alpha_i = 0$ y se encuentran en H_1 o H_2 (tal que la igualdad en la ecuación $y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0$ se cumple), o en ese lado de H_1 o H_2 tal que el desigualdad estricta se sostiene. Para estas máquinas, los vectores de soporte son los elementos críticos del conjunto de entrenamiento. Se encuentran más cerca del límite de decisión : si cualquier otro punto de entrenamiento fuera eliminado (o movidos, pero sin cruzar H_1 o H_2), y el entrenamiento fuera repetido, se encontraría el mismo hiperplano separador.

4.2. Etapa de Testeo

Una vez entrenada nuestra Support Vector Machine, hay que poder utilizarlo. Para eso (en el caso de clasificación binaria) hay que determinar de que "lado"del hiperplano un determinado test $\mathbf{x}$ yace y asignarle el correspondiente label. Dicho de forma concreta, predecimos la clase de $\mathbf{x}$ como $\boxed{sgn(\mathbf{w} \cdot \mathbf{x} + b)}$

4.3. El caso no separable

El algoritmo anterior para datos separables, cuando es aplicado en datos no separables, no produce soluciones satisfactorias. Esto se evidencia por un crecimiento arbitrario de la función objetivo (en otras palabras, el dual del Lagrangiano).

Para solucionar el problema, tendremos que relajar las condiciones:

$$\begin{array}{lll} \mathbf{x}_i \cdot \mathbf{w} + b \geq +1 & para & y_i = +1 \\ \mathbf{x}_i \cdot \mathbf{w} + b \leq -1 & para & y_i = -1 \end{array}$$

pero solo cuando sea necesario, es decir, deberíamos introducir un costo adicional (crecimiento en la función objetivo primal) para lograr nuestro cometido.

Esto puede lograrse introduciendo variables $\xi_i, i = 1, \cdots, l$ en dichas restricciones, obteniendo:

$$\begin{array}{lll} \mathbf{x}_i \cdot \mathbf{w} + b \geq +1 - \xi_i & para & y_i = +1 \\ \mathbf{x}_i \cdot \mathbf{w} + b \leq -1 - \xi_i & para & y_i = -1 \\ \xi_i \geq 0 \ \forall i. & & \end{array}$$

Por lo tanto, para que se produzca un error, el ξ_i correspondiente debe exceder la unidad, por lo que $\sum_i \xi_i$ es una bota superior en el número de errores de entrenamiento. Entonces naturalmente surge la forma de cambiar la función objetivo $||\mathbf{w}||/2$ a $||\mathbf{w}||/2 + C(\sum_i \xi_i)^k$ donde C es un parámetro que debe elegir el usuario, una C mayor que corresponde a una mayor penalización a los errores.

En su forma actual, este es un problema de programación convexo para cualquier entero positivo k positivo; para $k = 2$ y $k = 1$ también es un problema de programación cuadrática, y la opción $k = 1$ tiene la ventaja adicional de que ni las ξ_i ni los multiplicadores de Lagrange aparecen en el problema dual de Wolfe, que se convierte en:

"*Maximixar:*

$$L_D \equiv \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j \mathbf{x}_i \cdot \mathbf{x}_j$$

sujeto a: $0 \leq \alpha_i \leq C$, $\sum_i \alpha_i y_i = 0$."

La solución es dada por:

$$\mathbf{w} = \sum_{i=1}^{N_S} \alpha_i y_i \mathbf{x}_i$$

donde N_S es la cantidad de Support Vectors. La unica diferencia con el caso del óptimo hiperplano es que ahora los α_i tienen una cota superior en C

Necesitaremos las condiciones de Karush-Kuhn-Tucker para el problema primal. El Lagrangiano primal es:

$$L_P = \frac{1}{2}||\mathbf{w}||^2 + C \sum_i \xi_i - \sum_i \{y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i\} - \sum_i \mu_i \xi_i$$

donde μ_i son los multiplicadores de Lagrange introducidos para reforzar la positividad de los ξ_i. Las condiciones de KKT para el problema primal son entonces (notar que i recorre desde 1 hasta el numero de puntos de entrenamiento y ν desde 1 hasta la dimensión de los datos)

$$\frac{\partial L_P}{\partial w_\nu} = w_\nu - \sum_i \alpha_i y_i x_{i\nu} = 0$$

$$\frac{\partial L_P}{\partial b} = -\sum_i \alpha_i y_i = 0$$

$$\frac{\partial L_P}{\partial \xi_i} = C - \alpha_i - \mu_i \geq 0$$

$$y_i(\mathbf{x}_i \cdot w + b) - 1 + \xi_i \geq 0$$

$$\alpha_i \geq 0$$

$$\xi_i \geq 0$$

$$\mu_i \geq 0$$

$$\alpha_i\{y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i\} = 0$$

$$\mu_i \xi_i = 0$$

Usaremos las últimas dos ecuaciones para determinar el corrimiento b. Notemos que combinando la tercera y la última ecuación muestran que $\xi_i = 0$ si $\alpha_i < C$.

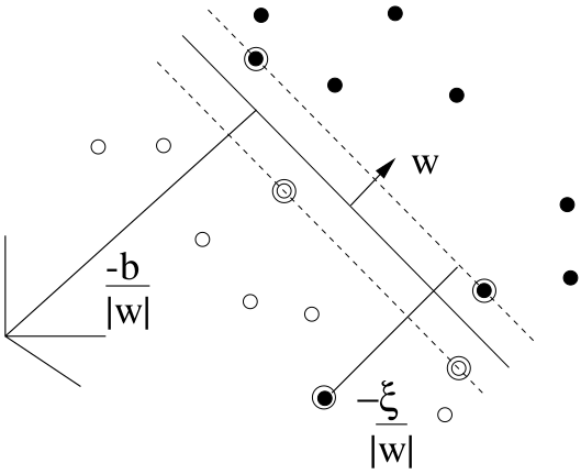

Figura 4: Separación lineal del caso no separable

5. Implementación

Formalmente una SVM construye un hiper plano si estamos separando dos clases o conjunto de hiper planos en el caso de multi-dimensión, dentro un espacio de dimensionalidad alta, muy alta, o incluso también infinita. Una buena separación entre las clases permitirá una clasificación correcta, también SVM permite afrontar problemas de regresión

Dado un conjunto de puntos, subconjunto de un conjunto mayor (espacio), en el que cada uno de ellos pertenece a una de dos posibles categorías, un algoritmo basado en SVM construye un modelo capaz de predecir si un punto nuevo (cuya categoría desconocemos) pertenece a una categoría o a la otra.

Como en la mayoría de los métodos de clasificación supervisada, los datos de entrada (los puntos) son vistos como un vector p-dimensional (una lista ordenada de p números).

La SVM busca un hiperplano que separe de forma óptima a los puntos de una clase de la de otra, que eventualmente han podido ser previamente proyectados a un espacio de dimensionalidad superior.

En ese concepto de "separación óptima.[es] donde reside la característica fundamental de las SVM: este tipo de algoritmos buscan el hiperplano que tenga la máxima distancia (margen) con los puntos que estén más cerca de él mismo. Por eso también a veces se les conoce a las SVM como clasificadores de margen máximo, como mencionamos en la seccion anterior del informe. De esta forma, los puntos del vector que son etiquetados con una categoría estarán a un lado del hiperplano y los casos que se encuentren en la otra categoría estarán al otro lado. En general cuanto mayor sea el margen, menor será el error de generalización de el clasificador. Al vector formado por los puntos más cercanos al hiperplano se le llama *support vector*.

En la literatura de los SVMs, se llama atributo a la variable predictora y característica a un atributo transformado que es usado para definir el hiper plano. La elección de la representación más adecuada del universo estudiado, se realiza mediante un proceso denominado selección de características.

5.1. Separabilidad de los datos

5.2. SVM Lineal

Supongamos un conjunto de datos de entrenamiento de n puntos de la forma $(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)$

Donde cada y_i son 1 o -1, cada uno de los cuales indica la clase a la que pertenece el punto $\vec{x}_i$. Cada $\vec{x}_i$ es un vector $p - dimensional$. Queremos encontrar el "hiperplano de margen máximo"que divide el grupo de puntos $\vec{x}_i$ para cual $y_i = 1$ del grupo de puntos para los cuales $y_i = -1$, que se define de modo que la distancia entre el hiperplano y el punto más cercano $\vec{x}_i$ de cualquiera de los grupos se maximiza.

Cualquier hiperplano puede escribirse como el conjunto de puntos. $\vec{x}$ satisfactorio $\vec{w} \cdot \vec{x} - b = 0$ dónde $\vec{w}$es el vector normal (no necesariamente normalizado) al hiperplano. Esto es muy parecido a la forma normal de Hesse , excepto que $\vec{w}$ No es necesariamente un vector unitario. El parámetro $\frac{b}{\|\vec{w}\|}$ determina el desplazamiento del hiper plano desde el origen a lo largo del vector normal $\vec{w}$.

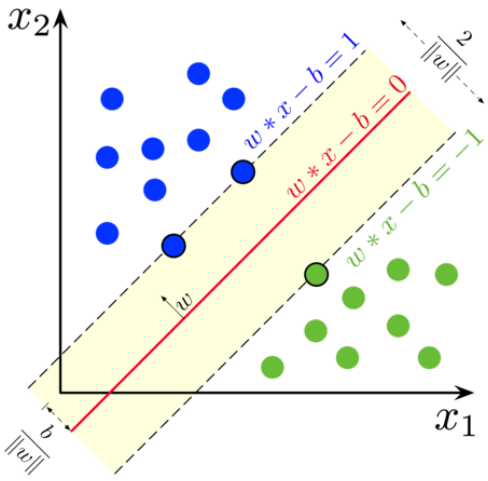

Figura 5: Ejemplo de SVM lineal

Notar en la Figura 5 que el hiper plano de margen máximo y los márgenes para un SVM entrenado

con muestras de dos clases. Las muestras en el margen se denominan *Support Vectors*.

5.2.1. Hard Margin

Si los datos de entrenamiento son linealmente separables , podemos seleccionar dos hiperplanos paralelos que separan las dos clases de datos, de modo que la distancia entre ellos sea lo más grande posible. La región limitada por estos dos hiperplanos se denomina "margen", y el hiperplano de margen máximo es el hiperplano que se encuentra a medio camino entre ellos. Con un conjunto de datos normalizado o estandarizado, estos hiperoplanos se pueden describir mediante las ecuaciones

$\vec{w} \cdot \vec{x} - b = 1$ (cualquier cosa en o sobre este límite es de una clase, con la etiqueta 1) y $\vec{w} \cdot \vec{x} - b = -1$ (cualquier cosa en o debajo de este límite es de la otra clase, con la etiqueta -1). Geométricamente, la distancia entre estos dos hiperplanos es $\frac{2}{\|\vec{w}\|}$, para maximizar la distancia entre los planos que queremos minimizar $|\vec{w}\|$. La distancia se calcula utilizando la distancia desde un punto a una ecuación plana. También tenemos que evitar que los puntos de datos caigan en el margen, agregamos la siguiente restricción: para cada i ya sea

$\vec{w} \cdot \vec{x}_i - b \geq 1$, Si $y_i = 1$ / o / $\vec{w} \cdot \vec{x}_i - b \leq -1$, Si $y_i = -1$

Estas restricciones establecen que cada punto de datos debe estar en el lado correcto del margen. Esto puede ser reescrito como

$y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1, \quad$ para todos$1 \leq i \leq n$.

Podemos poner esto juntos para obtener el problema de optimización:

"Minimizar $|\vec{w}\mid$ sujeto a $y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1$ para $i = 1, \ldots, n$. los $\vec{w}$ y b segundo Que resuelvan este problema determinan nuestro clasificador, $\vec{x} \mapsto \operatorname{sgn}(\vec{w} \cdot \vec{x} - b)$.

Una consecuencia importante de esta descripción geométrica es que el hiper plano de margen máximo está completamente determinado por aquellos $\vec{x}_i$ que se encuentran más cerca de ella. Estas $\vec{x}_i$ Se llaman vectores de soporte .

5.2.2. Soft Margin

Para extender el SVM a los casos en que los datos no son separables linealmente, introducimos la función de *Hinge Loss*

$\max(0, 1 - y_i(\vec{w} \cdot \vec{x}_i - b))$

Tengamos en cuenta que y_i es el objetivo (es decir, en este caso, 1 o -1), y $\vec{w} \cdot \vec{x}_i - b$ Es la salida de corriente. Esta función es cero si si $\vec{x}_i$ se encuentra en el lado correcto del margen. Para los datos en el lado incorrecto del margen, el valor de la función es proporcional a la distancia del margen. Entonces queremos minimizar

$\left[\frac{1}{n}\sum_{i=1}^{n} \max(0, 1 - y_i(\vec{w} \cdot \vec{x}_i - b))\right] + \lambda\|\vec{w}\|^2$

Donde el parámetro λ determina la compensación entre aumentar el tamaño del margen y garantizar que el $\vec{x}_i$Acuéstese en el lado correcto del margen. Así, para valores suficientemente pequeños de λ , el segundo término en la función de pérdida se volverá despreciable, por lo tanto, se comportará de manera similar a la SVM de margen duro, si los datos de entrada son linealmente clasificables, pero aún así aprenderán si una regla de clasificación es viable o no.

5.3. SVM con kernel

Mientras que el problema original puede plantearse en un espacio de dimensión finita, a menudo sucede que los conjuntos a discriminar no son linealmente separables en ese espacio. Por este motivo, se propuso que el espacio de dimensión finita original se asigne a un espacio de dimensión mucho más alta, presumiblemente haciendo que la separación sea más fácil en ese espacio. Para mantener la carga computacional razonable, las asignaciones utilizadas por los esquemas SVM están diseñadas para garantizar que los productos de puntos de pares de vectores de datos de entrada puedan calcularse fácilmente en términos de las variables en el espacio original, definiéndolas en términos de una función del *kernel* seleccionada especialmente para el problema, un ejemplo que intenta explicar de manera sencilla la idea del *kernel*, también conocido como *núcleo*, es la siguiente. Supongamos la función del núcleo llamada $K(x, y)$.

Los hiper planos en el espacio de dimensión superior (En la Figura 6 aparece como "Feature Space") se definen como el conjunto de puntos cuyo producto escalar con un vector en ese espacio es constante, donde dicho conjunto de vectores es un conjunto de vectores ortogonales (y por lo tanto mínimos) que

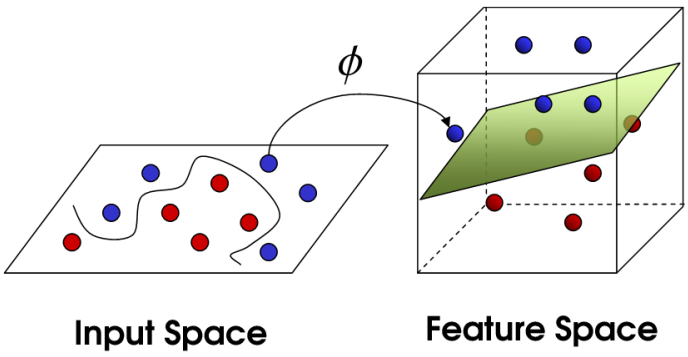

Figura 6: Ejemplo una función $K(x, y)$ que transforma las dimensiones

definen un hiper plano. Con esta elección del hiper plano, los puntos de x en el *Features Space* son mapeados al hiper plano definido por la relación $\sum_i \alpha_i K(x_i, x) = constante$ siendo α_i un parámetro dado por los "vectores features"x_i

Supongamos ahora que nos gustaría aprender una regla de clasificación no lineal que corresponde a una regla de clasificación lineal para los puntos de datos transformados
$\varphi(\vec{x}_i)$. $\varphi(\vec{x}_i)$. Además, se nos da una función de kernel. k que satisface
$k(\vec{x}_i, \vec{x}_j) = \varphi(\vec{x}_i) \cdot \varphi(\vec{x}_j)$

A este truco lo llamaremos el "kernel trick". El truco consiste en calcular el resultado de K(x, y) sin realizar el mapeo de los datos, es decir directamente a partir de x e y saber como será su producto interno, luego de la transformación a mas dimensiones.

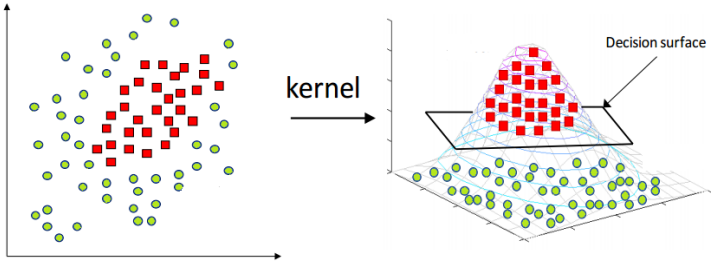

Figura 7: Ejemplo de un kernel trick

Tipos de funciones Kernel

- **Polinomial-homogénea:** No solo observa las características dadas de las muestras de entrada para determinar su similitud, sino también las combinaciones de éstas. En el contexto del análisis de regresión , tales combinaciones se conocen como características de interacción. El espacio de características (implícito) de un núcleo polinomial es equivalente al de la regresión polinomial , pero sin la explosión combinatoria en el número de parámetros a aprender. Cuando las entidades de entrada son de valor binario (booleanos), entonces las entidades corresponden a las conjunciones lógicas de las entidades de entrada. Esta definida por la formula

$$K(x, y) = (x^T y + c)^d$$

Donde x e y son vectores en el espacio de entrada, es decir, vectores de características calculadas a partir de muestras de entrenamiento o de prueba y $c \geq 0$ es un parámetro libre que intercambia la influencia de los términos de orden superior y de orden inferior en el polinomio. Cuando $c = 0$, el kernel (núcleo) se llama homogéneo

En la Figura 8, podemos observar: A la izquierda, un conjunto de muestras en el espacio de entrada, a la derecha, las mismas muestras en el espacio de features donde el núcleo polinomial $K(x, y)$ (para

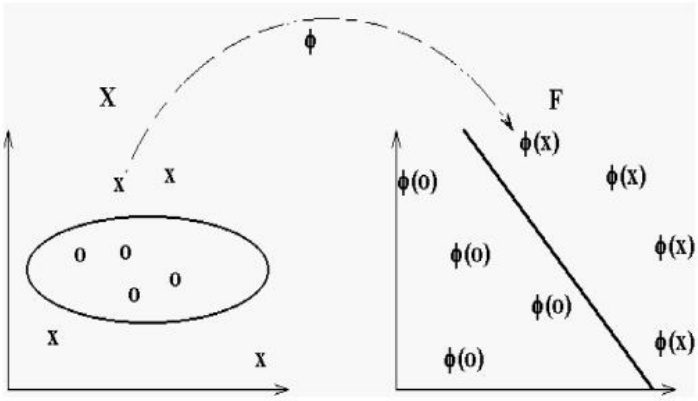

Figura 8: Ejemplo de un kernel polinómico

algunos valores de los parámetros c y d es el producto interno. El hiper plano obtenido en el espacio de features por un SVM es una elipse en el espacio de entrada.

- **Kernel Perceptrón:** El kernel Perceptrón es una variante del popular algoritmo de aprendizaje Perceptrón que puede aprender máquinas del kernel , es decir, clasificadores no lineales que emplean una función del kernel para calcular la similitud de muestras invisibles con muestras de entrenamiento

$$K(x_i, x_j) = \|x_i - xj\|$$

  Es un kernel poco utilizado debido a que presenta limitaciones.

- **Kernel Gausseano:** La función de base radial kernel, o RBF kernel, es una de las funciones mas populares utilizada en varios algoritmos de aprendizaje kernelizado. En particular, se usa comúnmente en la clasificación de SVMs.

  El kernel RBF en dos muestras x y x' , representado como vectores de características en algún espacio de entrada , se define como:

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

  Sigma es un parámetro que depende de los datos de entrada.

  Como el valor del kernel RBF disminuye con la distancia y oscila entre cero (en el límite) y uno (cuando x = x'), tiene una interpretación lista como una medida de similitud. El espacio de características del núcleo tiene un número infinito de dimensiones; para $\sigma = 1$, su expansión es:

$$\exp\left(-\frac{1}{2}\|\mathbf{x} - \mathbf{x}'\|^2\right) = \sum_{j=0}^{\infty} \frac{(\mathbf{x}^\top \mathbf{x}')^j}{j!} \exp\left(-\{12\|\mathbf{x}\|^2\right) \exp\left(-\frac{1}{2}\|\mathbf{x}'\|^2\right) \quad (1)$$

$$= \sum_{j=0}^{\infty} \sum_{\sum n_i = j} \exp\left(-\frac{1}{2}\|\mathbf{x}\,|^2\right) \frac{x_1^{n_1} \cdots x_k^{n_k}}{\sqrt{n_1! \cdots n_k!}} \exp\left(-\frac{1}{2}\|\mathbf{x}'\|^2\right) \frac{x'^{n_1}_1 \cdots x'^{n_k}_k}{\sqrt{n_1! \cdots n_k!}} \quad (2)$$

5.4. Separadores SVM multiclase

6. Dimensión VC de SVM

La dimensión de Vapnik-Chervonesky, conocida como dimensión VC es una forma de cuantificar la capacidad de una familia de funciones para separar un conjunto de puntos. Si tenemos un conjunto de l puntos y un miembro del conjunto de funciones $\{f(\alpha)\}$ es capaz de categorizarlos en todas las 2^l maneras

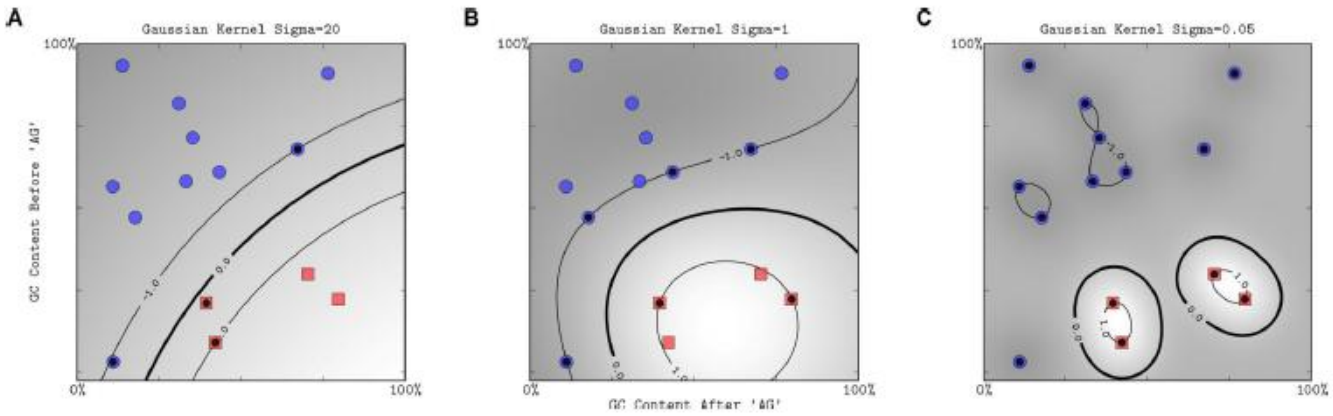

Figura 9: Ejemplo de un kernel gausseano con diferentes valores de σ

distintas que existen, entonces se dice que ese conjunto de puntos es destruido[3] por ese conjunto de funciones. Denotamos α a los parámetros que no dependen de los puntos del conjunto. La dimensión VC de $\{f(\alpha)\}$ es el máximo número de puntos de entrenamiento que pueden ser destruidos por $\{f(\alpha)\}$.

La dimensión VC de SVM depende directamente del kernel utilizado con el mismo. Se puede demostrar que la dimensión es en muchos casos alta y en algunos casos inclusive infinita. Si bien tener una dimensión VC alta puede traer problemas de *overfitting*[4], SVM suele tener un buen rendimiento de generalización.

Podemos relacionar la dependencia del kernel utilizado con la dimensión VC de SVM mediante el siguiente teorema:

Sea K un kernel que satisface las condiciones de Mercer y se corresponde con un espacio de incrustamiento mínimo H. Entonces la dimensión VC del correspondiente SVM es dim(H) + 1.

A partir de acá podemos ver la dimensión VC para cada uno de los kernels planteados calculando la dimensión del espacio de incrustamiento de cada kernel.

Podemos considerar que el SVM clásico, es decir, sin utilizar un kernel, es equivalente a utilizar un kernel de la forma $K(x_i, x_j) = 1 \cdot x_i \cdot x_j$. Observamos que este kernel satisface las condiciones de Mercer y se corresponde con un espacio de incrustamiento mínimo igual al espacio en que residen los datos, R^d. De aquí se concluye que la dimensión VC de SVM sin kernel es $dim(R^d) + 1 = d + 1$.

6.1. Dimensión VC utilizando un kernel polinómico

Para el caso de los kernels de tipo polinómico, esto es $K(x_i, x_j) = (x_i \cdot x_j)^p$ se conoce la dimensión del menor espacio de incrustamiento según el siguiente teorema:

Teorema 6.1. *Si el espacio en el cual se encuentran los datos tiene dimensión d_L (i.e. $\mathcal{L} = \mathbb{R}^{d_L}$), la dimensión del mínimo espacio de incrustamiento para kernels polinómicos homogéneos de grado p ($K(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 \cdot \mathbf{x}_2)^p,\ \mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^{d_L}$) es $\binom{d_L+p-1}{p} + 1$*

6.2. Dimensión VC utilizando un kernel RBF

Los kernels RBF tienen un espacio de incrustamiento de dimensión infinita, lo cual hace que la dimensión VC de un SVM con kernel RBF sea infinita. Esto queda demostrado por el siguiente teorema:

Teorema 6.2. *Considerar la clase de los kernels que satisfacen la condición de Mercer para los cuales $K(x_i, x_j) \to 0$ cuando $\|x_i - x_j\| \to \infty$ y para el cual $K(x, x) = O(1)$, y asumir que los datos pueden ser tomados arbitrariamente de R^d. Entonces la familia de clasificadores que consiste de SVM utilizando estos kernels y para los cuales la penalización de error tiene permitido tomar todos los valores, tiene dimensión VC infinita.*

La demostración de este teorema se encuentra en el apéndice.

Tomando como ejemplo el kernel RBF Gaussiano, $K(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{\sigma}}$ podemos observar que cuando el parámetro $\sigma \to \infty$ se cumple lo siguiente:

$$K(x_i, x_i) \to 1$$

[3] Del inglés, *shattered*

[4] Overfitting refiere a ajustarse demasiado a los datos y no generalizar

y

$$K(x_i, x_j) \to 0$$

Esto da la idea de que independientemente del conjunto de datos indicado, para un valor suficientemente grande de σ la clasificación de cada punto depende casi exclusivamente de su propia etiqueta, ya que la contribución del resto de los puntos es despreciable.

7. Rendimiento

Cabe destacar la popularidad que han adquirido los SVMs desde que fue desarrollado por Vladimir Vapnik en la década del 90, principalmente por ser el estado del arte durante un tiempo para la clasificación de textos e imagines. Una de las características principales de los SVMs son su rendimiento a la hora de manejar datos que se encuentran en grandes dimensiones, situación que le hes desfavorable a muchas algoritmos por la llamada "maldición de la dimensionalidad".
El rendimiento, en la practica, al igual que muchos algoritmos, dependen de cada caso, es decir, es muy susceptible a factores tales como la cantidad de de datos del set de entrenamiento, la dimensión de los datos, el preprocesamiento de los datos, entre otros factores.
Los SVMs son herramientas muy poderosas pero sus requerimientos de espacio y de operaciones computacionales aumentan rápidamente con el numero de vectores de entrenamiento. La parte central del algoritmo de un SVM, separar los support vectors del resto de los datos de entrenamiento, es un problema de programación cuadrática (QP) y por ejemplo, haciendo referencia a la librería de sklearn (utilizada en los ejemplos de este informe), un problema QP es del orden $O(n_{features} * n^2_{samples})$ o $O(n_{features} * n^3_{samples})$.
Otra factor que resulta crucial a la hora de pensar en el rendimiento de un SVM, es la elección tanto del kernel a utilizar como del único parámetro del algoritmo (en este caso no tendremos en cuenta parámetros que pueden aparecer dentro de algunos kernels debido a habría que analizar cada kernel por separado) que en este caso llamaremos "C", el mismo como se menciona anteriormente esta relacionado con la penalización por clasificar mal. Un "C" muy chico hace que el modelo no responda correctamente a los datos, mientras que un "C" muy grande nos lleva a producir overfitting.
Lamentablemente no hay teoría hasta el momento sobre como elegir este parámetro y como mencionamos al principio de esta sección, en la practica depende mucho de cada caso, por lo tanto al igual que en muchos algoritmos la forma de elegir el valor de "C" que genera un mejor rendimiento se hace mediante la técnica llamada grid-search.

8. Limitaciones

Dentro de las limitaciones de SVM y la teoría desarrollada en torno a este algoritmo se encuentran principalmente en la elección del kernel a usar. Como vimos anteriormente en caso de que los datos no sean linealmente separables el uso de kernels es necesario para el correcto funcionamiento del algoritmo pero como se puede ver en los distintos ejemplos utilizados cada kernel mapea los datos a espacios distintos y, al menos teóricamente, no hay forma de determinar cual es el mejor a utilizar para un set de datos dado. El mismo problema tenemos a la hora de determinar el parámetro "C" mencionado anteriormente. Igualmente, haciendo uso de distintos métodos prácticos, como el ya mencionado grid-search, podemos ver, por ejemplo, que kernel tiene un mejor rendimiento frente a otro para un caso particular.
Otra limitación encontramos a la hora de usar set de entrenamientos muy grandes, como se menciono en la sección de rendimiento, los requerimientos de espacio y de operaciones computacionales aumentan rápidamente con el numero de vectores de entrenamiento, problema que todavía sigue abierto. Por ultimo cabe mencionar que a pesar de de que hay investigaciones hechas sobre el uso de SVM como clasificador multiclase y los resultados obtenidos generalmente son buenos, el diseño optimo de un SVM multiclase sigue siendo un área de investigación.

9. Ejemplos

En esta sección presentaremos una serie de ejemplos que utilizan SVM, se intentará ser exhaustivo mostrando simulaciones con los distintos Kernels y a su vez la mayor cantidad de situaciones en la que la aplicación de este algoritmo sea beneficiosa.

9.1. SVM Lineal

9.1.1. Clases Linealmente separables

Nuestro primer ejemplo [5] simplemente genera un hiperplano separador teniendo en cuenta los datos de training generados por la función *make_blobs* de *sklearn.datasets*.

Para empezar notemos que contamos con dos 'clusters' generados convenientemente con la función previamente mencionada (en la cual se puede modificar el número de centros, desviación estandar de los clusters , etc):

```
X, Y = make_blobs(n_samples=50, centers=2, random_state=6)
```

Para este ejemplo usaremos 50 puntos de $\mathbb{R}^2$ como Training con sus respectivos labels de clase.

Para clasificarlos usaremos la función svm.SVC de la librería *sklearn* que genera un clasificador SVM y podemos obtener, entre otras cosas:

- Parámetros de nuestro hiperplano **w**
- Constantes en la función de decisión
- Lista de Support Vectors

En la siguiente figura se puede ver el hiperplano separador **w** (línea continua) y los dos hiperplanos paralelos a **w** que pasan por al menos un support vector (línea a trozos):

[5] Se encuentra en:
https://github.com/julianferres/Aprendizaje-Estadistico/blob/master/Presentacion%20Final/SVM%20lineal.ipynb

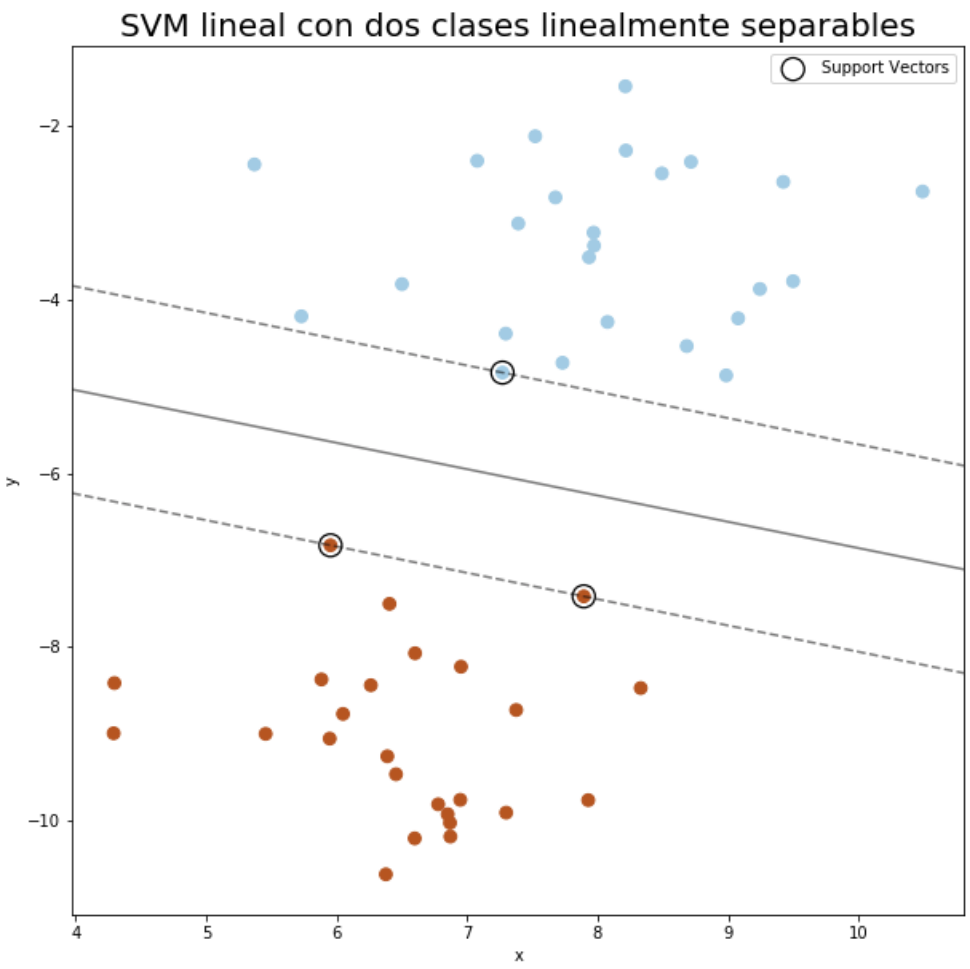

Figura 10: Ejemplo de SVM lineal

Llamaremos a los dos hiperplanos paralelos H_1 y H_2, notemos que esos hiperplanos son las curvas de nivel $c = -1$ y $c = 1$ de nuestra función de clasificación $\mathbf{x}_i \cdot \mathbf{w} + b = c$ y contienen a los support vectors de cada una de las clases.

Podemos por ejemplo tener una lista de los support vectors y ver que coinciden con los resaltados en la imagen:

```
clf.support_vectors_
----------------------
array([[ 7.27059007, -4.84225716],
       [ 5.95313618, -6.82945967],
       [ 7.89359985, -7.41655113]])
```

Se puede observar que el primer support vector es el de la curva de nivel donde $c = -1$ y los últimos de la curva de nivel $c = 1$

9.2. Reconocimiento de dígitos manuscritos con kernel rbf (Gaussiano)

En esta subsección, mostraremos un ejemplo[6] de como se puede utilizar el clasificador SVM de scikit-learn para el reconocimiento de dígitos manuscritos.

Para eso utilizamos un data set con imagenes de 8x8 pixeles y representan dígitos manuscritos.

Mostramos a modo de ejemplo una ilustración de las primeras 4 imagenes. Cabe aclarar que en el código que se presenta, las mismas se encuentran en el atributo 'images' del set de datos.

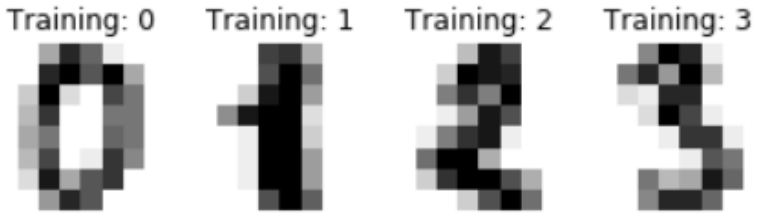

Figura 11: Algunas muestras de nuestro set de Entrenamiento

Algunos datos importantes sobre el set de datos con el que estamos trabajando:

- Cada muestra es una imagen de tamaño 8x8 que representa un dígito.
- Cantidad de clases: 10 (números del 0 al 9)
- Tamaño de muestra por clase: aproximadamente 180
- Tamaño de muestra total: 1797.
- Dimensionalidad: 64
- Features de cada muestra: enteros del 0 (blanco) al 16 (negro).

Notemos que las imagenes tienen que tener el mismo tamaño (para respetar el espacio Hilbertiano en el que se encuentran).

En este caso, si denotamos por A al conjunto de enteros mayores o iguales a 0 y menores o iguales a 16: $A = \{x \in \mathbb{Z} : 0 \leq x \leq 16\}$, entonces las muestras pueden ser representadas como puntos del espacio A^{64}.

En este caso conocemos los labels de nuestro set de entrenamiento (sera un aprendizaje supervisado), que es un numero entero entre 0 y 9 (los guardamos en el atributo 'target').

Simplemente iniciamos nuestro clasificador y lo entrenamos (en este caso) con la primera mitad de nuestro set de entrenamiento (dejando la segunda mitad para testing):

```
# Creamos el clasificador SVM
classifier = svm.SVC(gamma = 0.001)

# En este caso usamos como set de entrenamiento la primera mitad de nuestros datos.
classifier.fit(data[:n_samples // 2], digits.target[:n_samples // 2])
```

En la función svm.SVC(), 'gamma' es el coeficiente del Kernel para 'rbf', 'poly' y 'sigmoid'.

Una vez entrenado el clasificador, procedemos a testearlo

```
# Ahora nos toca usar como test la segunda mitad de los digitos.
expected = digits.target[n_samples // 2:]
predected = classifier.predict(data[n_samples // 2:])
```

[6] El codigo se encuentra en: https://github.com/julianferres/Aprendizaje-Estadistico/blob/master/Presentacion%20Final/plot_digits_classification.ipynb

Antes de pasar a los resultados, mostramos la configuración de parametros que se utilizó en este ejemplo, para consultar sobre el significado de cada uno, visitar la pagina Oficial de Sklearn y buscar la función correspondiente [7]:

```
Reporte de nuestro clasificador SVC(C=1.0, cache_size=200, class_weight=None,
coef0=0.0, decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
max_iter=-1, probability=False, random_state=None, shrinking=True,
tol=0.001, verbose=False):
             precision    recall  f1-score   support

          0       1.00      0.99      0.99        88
          1       0.99      0.97      0.98        91
          2       0.99      0.99      0.99        86
          3       0.98      0.87      0.92        91
          4       0.99      0.96      0.97        92
          5       0.95      0.97      0.96        91
          6       0.99      0.99      0.99        91
          7       0.96      0.99      0.97        89
          8       0.94      1.00      0.97        88
          9       0.93      0.98      0.95        92

avg / total       0.97      0.97      0.97       899
```

También se muestran los resultados según diferentes métricas(tales como precisión[8], recall [9] y F1-score [10]) para cada clase en particular y el promedio global. Tambien se muestra la cantidad de muestras de cada clase que fueron utilizadas.

Se procede a mostrar ahora la 'matriz de confusión' [11] del sistema:

```
Matriz de confusión:
[[87  0  0  0  1  0  0  0  0  0]
 [ 0 88  1  0  0  0  0  0  1  1]
 [ 0  0 85  1  0  0  0  0  0  0]
 [ 0  0  0 79  0  3  0  4  5  0]
 [ 0  0  0  0 88  0  0  0  0  4]
 [ 0  0  0  0  0 88  1  0  0  2]
 [ 0  1  0  0  0  0 90  0  0  0]
 [ 0  0  0  0  0  1  0 88  0  0]
 [ 0  0  0  0  0  0  0  0 88  0]
 [ 0  0  0  1  0  1  0  0  0 90]]
```

En esta matriz las columnas representan la predicción realizada y las filas la verdadera clase de cada imagen. (Por lo tanto en la diagonal principal se encuentran los aciertos correspondientes a cada clase, siendo consecuentemente la traza la cantidad de aciertos totales).

Se muestran también algunos ejemplos con la predicción realizada sobre ellos:

[7] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
[8] https://es.wikipedia.org/wiki/Precision_y_exhaustividad
[9] https://en.wikipedia.org/wiki/Recall
[10] https://en.wikipedia.org/wiki/F1_score
[11] https://es.wikipedia.org/wiki/Matriz_de_confusion

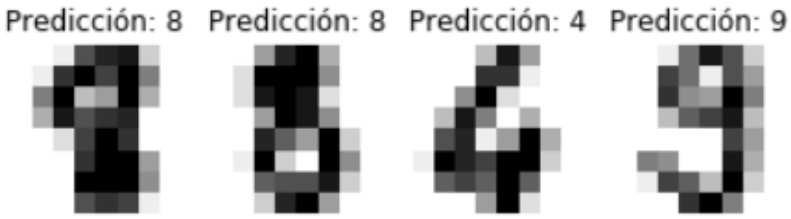

Figura 12: Algunas predicciones realizadas sobre la muestra.

9.3. Simulación de SVM para el dataset 'iris' con distintos kernels

En este caso simplemente trataremos de clasificar con el clasificador svm.SVC() de sklearn una set de datos clásico que contiene información de flores del genero Iris [12]. Contiene tres clases (i.e tres especies del genero) con 50 observaciones de cada clase.

Algunos datos importantes sobre el set de datos con el que estamos trabajando:

- Cada muestra es una imagen de tamaño 8x8 que representa un dígito.
- Cantidad de clases: 3 (Especies setosa, versicolor, virginica)
- Tamaño de muestra por clase: 50
- Tamaño de muestra total: 150.
- Dimensionalidad: 4
- Features de cada muestra:
  - ⋆ sepal length in cm.
  - ⋆ sepal width in cm.
  - ⋆ petal length in cm.
  - ⋆ petal width in cm.

Mostramos un gráfico (pair plot) que saca a relucir las relaciones entre cada par de atributos, y ademas separamos según la clase a nuestras muestras en tres colores:

[12] https://es.wikipedia.org/wiki/Iris_(planta)

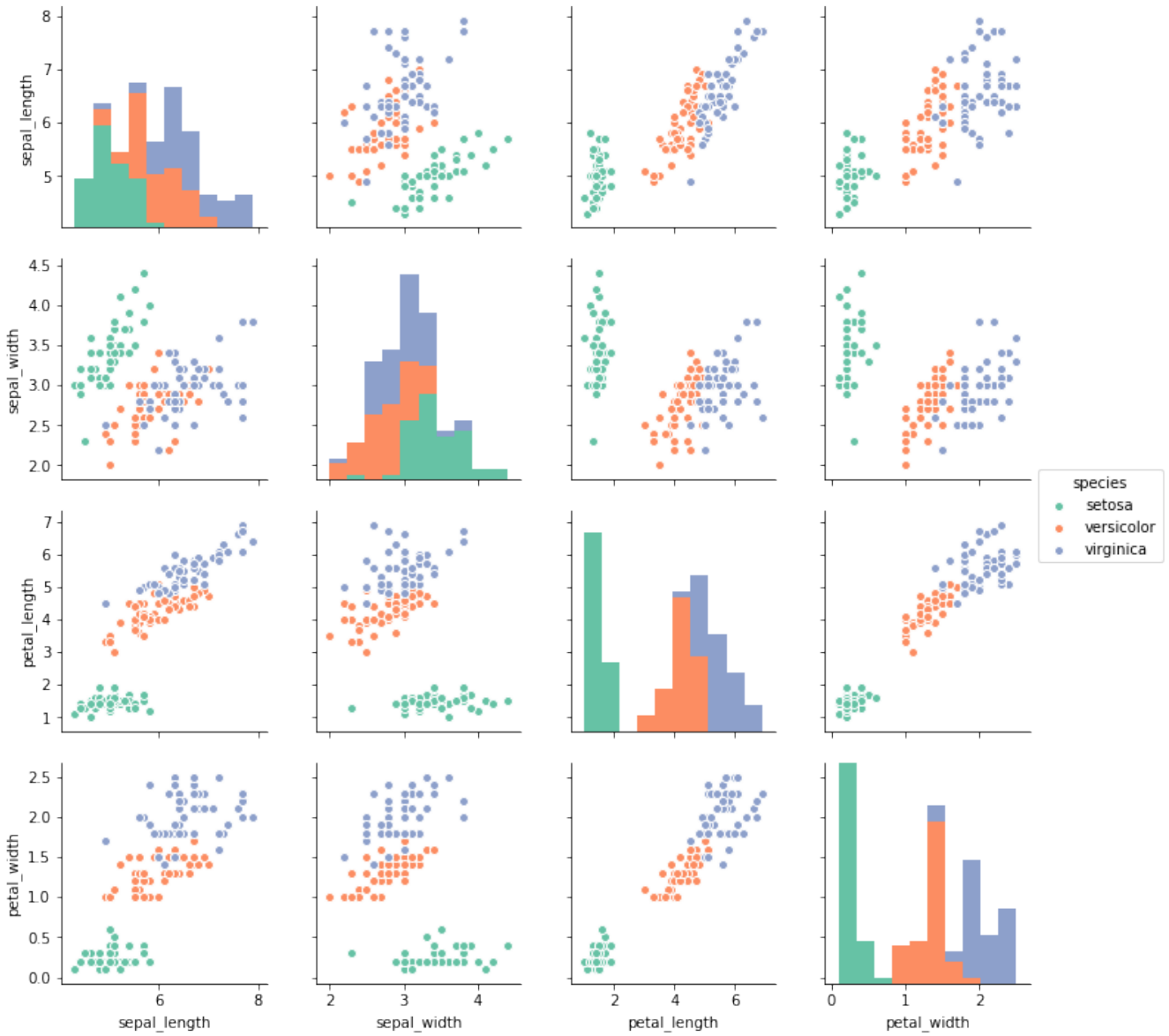

Figura 13: Pair plot de los atributos del set de datos iris.

Notar que en la diagonal principal se ilustra un histograma (stacked) para ese atributo y las tres clases correspondiente

Pasamos ahora a entrenar el algoritmo:

```
#Defino el clasificador SVM
clf = svm.SVC(kernel='linear', C=1).fit(X_train, y_train)
print(clf.score(X_test, y_test))
(105, 4) (105,)
(45, 4) (45,)
0.9777777777777777
```

Ahora veamos el reporte del clasificador con las mismas métricas que en el ejemplo anterior y la matriz de confusión.

```
expected = y_test
predicted = clf.predict(X_test)
```

```
Reporte de nuestro clasificador SVC(C=1, cache_size=200, class_weight=None, coef0=0.0,
decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',max_iter=-1, probability=
```

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	16
versicolor	1.00	0.94	0.97	18
virginica	0.92	1.00	0.96	11
avg / total	0.98	0.98	0.98	45

```
Matriz de confusión:

    [[16 0  0]
    [ 0  17 1]
    [ 0  0  11]]
```

Se puede ver por el reporte que se uso un clasificador lineal, y por la matriz de confusión la cantidad de muestra que usamos para testear.

10. Extensiones

En esta sección describiremos dos de los mas simples y eficientes métodos para mejorar la performance de SVM.

10.1. SVM Virtual

El método de SVM virtual intenta incorporar invariantes conocidas del problema (por ejemplo, invariante a la traslación para el problema de reconocimiento en imágenes) mediante el previo entrenamiento de un sistema, para luego crear nueva información mediante la distorsión de los *"Vectores de soporte"* (por ejemplo, en el caso mencionado, con una traslación correspondiente) y finalmente entrenar un nuevo sistema con la información distorsionada y sin distorsionar.

La idea es sencilla de implementar y parece funcionar mucho mejor que otros métodos que intentan incorporar invariantes.

10.2. Método del Set Reducido

Este método fue introducido para impactar en la velocidad de SVMs en la fase de Test, y de la misma manera que el método anterior, comienza con un SVM entrenado.

La idea de este método es reemplazar la suma $\mathbf{w} = \sum_{i=1}^{N_S} \alpha_i y_i \mathbf{x}_i$ (en esta suma participan tantos términos como *support vectors* tenga la maquina) por una suma similar, donde en lugar de *support vectors* son usados *computed vectors* (vectores que no son parte del set de entrenamiento) y en lugar de los α_i, un conjunto diferente de pesos es usado. El número de parametros es elegido de antemano para lograr el *speedup*[13] deseado.

El vector resultante es un todavia un vector en el mismo espacio de Hilbert $\mathcal{H}$ y los parámetros son encontrados minimizando la Norma Euclideana de la diferencia entre el vector original $\mathbf{w}$ y su aproximación a el.

La misma técnica puede ser usada en regresión mediante SVM para encontrar representaciones de funciones mucho mas eficiente (que pueden ser usadas, por ejemplo, en compresión de datos).

10.2.1. Rendimiento

Combinando estos dos métodos se puede alcanzar un factor de 50 en *speedup* (es decir, un tiempo de ejecución 50 veces menor incorporando las mejoras) con solo un incremento del error de 1,0 % a 1,1 % en el set de dígitos de la base de datos $NIST$.

[13] Es la mejora en la velocidad de ejecución de una tarea ejecutada en dos arquitecturas similares con diferentes recursos (en este caso, los modelos)

A. Anexo I: Demostraciones de Teoremas

Teorema 6.1: Si el espacio en el cual se encuentran los datos tiene dimensión d_L (i.e. $\mathcal{L} = \mathbb{R}^{d_L}$), la dimensión del mínimo espacio de incrustamiento para kernels polinómicos homogéneos de grado p ($K(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 \cdot \mathbf{x}_2)^p,\ \mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^{d_L}$) es $\binom{d_L+p-1}{p} + 1$.

Demostración. Antes de comenzar veamos que podemos construir explícitamentee' para este caso el mapeo Φ. Denotando $z_i = x_{1i}x_{2i}$, tenemos que $K(\mathbf{x}_1, \mathbf{x}_2) = (z_i + z_2 + \cdots + z_{d_L})^p$, entonces se ve que cada dimensión de $\mathcal{H}$ corresponde a un termino con potencias dadas de las z_i en la expansión de K. En efecto, si elegimos notar las componentes de $\Phi(\mathbf{x})$ de esta manera, podemos explícitamente escribir el mapeo para cualquier p y d_L:

$$\Phi_{r_1 r_2 \cdots r_{d_L}}(\mathbf{x}) = \sqrt{\left(\frac{p!}{r_1! r_2! \cdots r_{d_L}!}\right)} x_1^{r_1} x_2^{r_2} \cdots x_{d_L}^{r_{d_L}},\ \sum_{i=1}^{d_L} r_i = p,\ r_i \geq 0 \tag{3}$$

En principio, probaremos que el número de componentes de $\Phi(\mathbf{x})$ es $\binom{d_L+p-1}{p}$. Nombramos las componentes de $\Phi(\mathbf{x})$ como en la ecuación (3). Luego una componente es unívocamente determinada por la elección de los d_L enteros $r_i \geq 0, \sum_{i=1}^{d_L} r_i = p$. Ahora consideramos p objetos distribuidos entre $d-1$ particiones (numeradas desde 1 hasta $d-1$), tales objetos tienen permitido también estar a la izquierda de todas las particiones, o a la derecha de todas las particiones. Si m objetos caen entre las particiones q y $q+1$, entonces le hacemos corresponder al término x_{q+1} el exponente m, es decir, en el producto de la ecuación (3) aparecerá x_{q+1}^m. Similarmente, m objetos a la izquierda de todas las particiones se corresponden con x_1^m, y m objetos a la derecha de todas las particiones se corresponden al término $x_{d_L}^m$. Entonces la cantidad de términos distintos de la forma $x_1^{r_1} x_2^{r_2} \cdots x_{d_L}^{r_{d_L}}$, $\sum_{i=1}^{d_L} r_i = p$, $r_i \geq 0$ se puede deducir de la siguiente forma:

En lugar de pensar $d_L - 1$ separaciones y p objetos a ubicar entre ellos, pienso en $d - 1 + p$ espacios en los cuales hay que colocar o bien un objeto, o bien un separador. Entonces como $\sum_{i=1}^{d_L} r_i = p$, puedo colocar los p objetos, y los $d_L - 1$ espacios libres deberán ser llenados por separadores. Esto nos lleva a que la cantidad de componentes es $\binom{d_L+p-1}{p}$.

Ahora tendrémos que mostrar un conjunto de vectores con componentes $\Phi_{r_1 r_2 \cdots r_{d_L}}(\mathbf{x})$ abarca el espacio $\mathcal{H}$. Esto sigue del hecho que las componentes de $\Phi(\mathbf{x})$ son funciones linealmente independientes.

Pero supongamos en lugar de esto, para arribar a un absurdo, que la imagen de Φ actuando en $\mathbf{x} \in \mathcal{L}$ es un subespacio de $\mathcal{H}$. Entonces existe un vector no nulo, fijo, $\mathbf{V} \in \mathcal{H}$ tal que:

$$\sum_{i=1}^{\dim(\mathcal{H})} V_i \Phi_i(\mathbf{x}) = 0,\ \forall \mathbf{x} \in \mathcal{L} \tag{4}$$

Usando la notación anterior, consideremos una componente particular de Φ

$$\Phi_{r_1 r_2 \cdots r_{d_L}}(\mathbf{x}),\ \sum_{i=1}^{d_L} r_i = p \tag{5}$$

Como la ecuación (4) se cumple para todo $\mathbf{x} \in \mathcal{L}$, y del hecho que el mapeo de Φ en (3) tiene todas sus derivadas parciales definidas, podemos aplicar el operador:

$$(\frac{\partial}{\partial x_1})^{r_1} (\frac{\partial}{\partial x_2})^{r_2} \cdots (\frac{\partial}{\partial x_{d_L}})^{r_{d_L}} \tag{6}$$

a la ecuación (4), que va a tomar el término particular con las correspondientes potencias de los x_i en la ecuación (3), dando:

$$V_{r_1 r_2 \cdots r_{d_L}} = 0. \tag{7}$$

Como esto es cierto para todas las elecciones de $r_1, r_2, \cdots, r_{d_L}$ con $\sum_{i=1}^{d_L} r_i = p$, entonces cada componente de $\mathbf{V}$ debe anularse. Esto es un absurdo que vino de suponer que la imagen de Φ actuando en $\mathbf{x} \in \mathcal{L}$ es un subespacio de $\mathcal{H}$. Por lo tanto la imagen de Φ actuando en $\mathbf{x} \in \mathcal{L}$ abarca $\mathcal{H}$

□

Referencias

[1] **A Tutorial on Support Vector Machines for Pattern Recognition , CHRISTOPHER J.C. BURGES**

[2] http://www.svms.org/

[3] sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[4] Set de Datos iris: https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data

[5] Set de datos digitos: http://yann.lecun.com/exdb/mnist/