# **Capítulo 5. Support Vector Machine SVM**

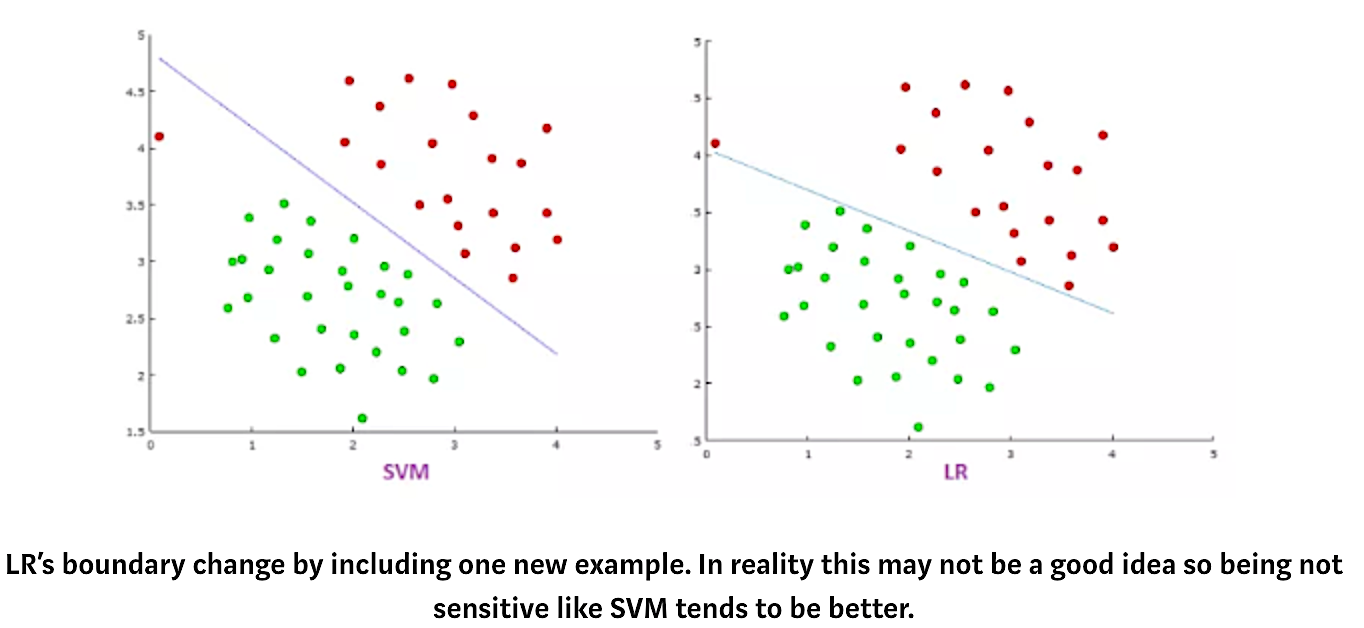
* **Introducción**

Support Vector Machine o en español, máquinas de vectores de soporte es un poderoso y versátil modelo de Machine Learning. SVM es capaz de desempeñar clasificaciones lineales, no lineales, regresiones y hasta detectar datos anormales. Normalmente se utiliza para set de datos complejos de pequeño y mediano tamaño.

Debido a su alcance, este modelo es uno de los más utilizados en Machine Learning, así que cualquier que quiera comenzar un proyecto de ML, debe de saber utilizar SVM.

## **Clasificación Lineal con SVM**

Comencemos analizando esta imagen. Aquí se muestra una comparación de modelos de clasificación, en uno se utilizó SVM y en el otro se utilizó regresión logística. El objetivo de los modelos fue separar la clase roja de la clase verde. Cabe resaltar que en el set de datos rojo, se tiene un outlier, pegadito al eje Y, cercano al 4.



Notemos que cada modelo tuvo una forma diferente de atacar ese outlier, mientras parece que el modelo SVM ignoró el outlier, la regresión logística inclinó bastante su frontera de decisión para adaptarse a él.

Aunque es tentador no cometer ningún error en la clasificación, en la realidad esto no es factible. En ocasiones por ajustar tan meticulosamente a un set de datos en específico, se cae en un sobreajuste y, muy probablemente, cuando se pruebe un nuevo set de datos, el modelo no tendrá un buen desempeño ya que la frontera de decisión se vio reducida.

En cambio, con el modelo SVM se obtuvo una clasificación con una mejor frontera de decisión, es decir, se obtuvo un alto margen de clasificación.

Básicamente, puedes pensar el modelo de SVM como un algoritmo que intenta conseguir la mayor distancia posible entre el set datos de cada clase y la respectiva línea de decisión.

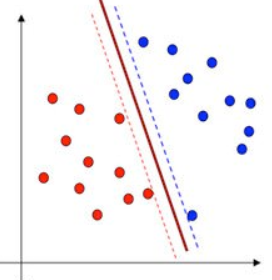
Para ello se utiliza un término llamado vectores de soporte. Estos vectores se encuentran a los lados de la línea de decisión y son los puntos que definen el margen máximo de separación del hiperplano que separa las clases.

Diagram

Description automatically generated

### **Clasificación con Margen Duro**

Digamos que alguien es muy estricto y quiere que los datos estén completamente separados, sean outliers o no. A esto se le llama clasificación de margen duro.



Este modo de clasificación tiene dos grandes desventajas: los datos tienen que ser linealmente separables y al ser tan estricto el modelo se vuelve sensible a los outliers.

Chart, scatter chart

Description automatically generated

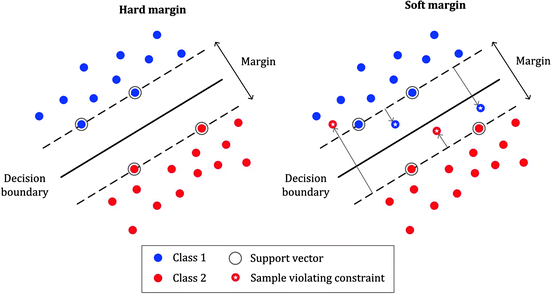
Como podemos ver gracias a este outlier, el modelo se recurrió a la derecha que generó un sobreajuste y a la vez una reducción de la “calle” de separación. De manera que si utilizamos este modelo con otro set de datos nos enfrentaremos con rendimientos deficientes.

Debido a las inflexibilidades de esta clasificación, en la realidad este método no es comúnmente utilizado. La mayoría de los sets de datos no seguirán un patrón tan riguroso, por lo que, la clasificación más utilizada es la de margen suave.

### **Clasificación con un Margen Suave**

La clasificación de margen suave permite establecer un tamaño de margen óptimo evitando el sobreajuste y conservando la flexibilidad que necesitan los datos reales, aunque estas impliquen violaciones de margen.

La parte importante de esta forma de clasificación radica en encontrar un equilibrio entre evitar sobreajustar y no permitir tantas violaciones de margen. Con este equilibrio se puede obtener un modelo que pueda adaptarse a nuevos sets de datos.



Vamos a ver un ejemplo de cómo crear un SVM de clasificación lineal con *scikit-learn.*

Para esto vamos a utilizar el mismo ejemplo que con la regresión logística: la clasificación de empleados admitidos y no admitidos.

candidates = {'gmat': [780,750,690,710,680,730,690,720,740,690,610,690,710,680,770,610,580,650,540,590,620,600,550,550,570,670,660,580,650,660,640,620,660,660,680,650,670,580,590,690],

'gpa': [4,3.9,3.3,3.7,3.9,3.7,2.3,3.3,3.3,1.7,2.7,3.7,3.7,3.3,3.3,3,2.7,3.7,2.7,2.3,3.3,2,2.3,2.7,3,3.3,3.7,2.3,3.7,3.3,3,2.7,4,3.3,3.3,2.3,2.7,3.3,1.7,3.7],

'work\_experience': [3,4,3,5,4,6,1,4,5,1,3,5,6,4,3,1,4,6,2,3,2,1,4,1,2,6,4,2,6,5,1,2,4,6,5,1,2,1,4,5],

'admitted': [1,1,0,1,0,1,0,1,1,0,0,1,1,0,1,0,0,1,0,0,1,0,0,0,0,1,1,0,1,1,0,0,1,1,1,0,0,0,0,1]

}

df = pd.DataFrame(candidates,columns= ['gmat', 'gpa','work\_experience','admitted'])

df.head()

Table

Description automatically generated.

Separamos nuestros predictores *x* de nuestra variable a predecir *y*.

x = df[['work\_experience','gpa','gmat']]

y = df['admitted']

No hay que olvidar que cuando utilizamos SVM es muy importante escalar las variables. Por lo tanto, vamos a crear un pipeline que haga esta función y que nos sirva para entrenar nuestro modelo.

svm = Pipeline([

("escalar", StandardScaler()),

("svc", SVC(kernel="linear", C=1))

])

Como podemos ver en nuestro pipeline, tenemos 2 instancias, una que escala los datos con *StandardScaler* y otra que se encarga de crear el modelo SVC.

Al SVC le agregamos 2 parámetros importantes. Primero, como estamos hablando de clasificaciones lineales, le agregamos el formato linear al parámetro kernel (más adelante veremos otro tipo de clasificaciones).

También especificamos el nivel de flexibilidad de los vectores de soporte con el parámetro C.

Por ejemplo, cuando creamos un modelo SVM con scikit-learn, podemos utilizar el parámetro C, para controla el tamaño de nuestros vectores de soporte. Si C tiene un valor alto, entonces nuestro margen de decisión será más grande (suave), y si C tiene un valor más pequeño entonces el margen será más pequeño (Duro). En este caso utilizamos un margen duro, ya que C es pequeño.

Vamos a probar una predicción aleatoria:

svm.fit(x,y)

svm.predict([[5, 3.9, 720]])

En este caso queremos saber si un candidato con 5 años de experiencia laboral, un GPA de 3.9 y un score en el GMAT de 720 sería admitido en la empresa.

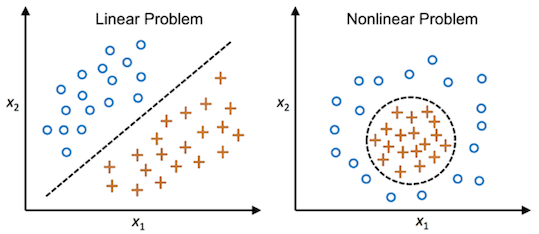


Nuestro modelo nos regresó un 1, lo que significa que si lo considera como alguien que sería admitido en la empresa.

## **Clasificación No Lineal Con SVM**

Los clasificadores lineales con SVM son eficientes, sin embargo, los datos necesitan tener una tendencia lineal para poder utilizarlos. En la realidad, hay muchos sets de datos, si no es que la mayoría que no son linealmente separables y se tiene que emplear un método diferente de clasificación.





En este caso el set de datos A funcionaria bien con un modelo lineal, mientras que el B no se podría separar linealmente.

Una forma de lidiar con clasificaciones que no son lineales es agregar más variables, un ejemplo pueden ser las variables polinomiales, como hicimos en la regresión polinomial.

Recordemos que, con variables polinomiales surgen nuevas variables de nuestra variable base, por ejemplo, tenemos un set de datos x1 y agregamos otra variable que surge de ese mismo set x2 = x12.

En el caso de los modelos de clasificación, al añadir estas variables, cambias la dimensión de los datos.

Chart, scatter chart

Description automatically generated

Observemos la imagen siguiente. Si ponemos los datos en la franja de de x1, nos encontramos con datos posicionados en una dimensión. La separación lineal sería imposible. Sin embargo, si agregamos la variable x2 al modelo, el set de datos se puede visualizar en dos dimensiones y la separación lineal resultaría.

Cuando agregamos características polinomiales a un modelo de clasificación, permite visualizar los datos en dimensiones superiores. Por ejemplo, en la imagen anterior vimos como un modelo unidimensional paso a ser bidimensional.

### **Kernel Polinomial**

Como vimos en el ejemplo para crear un clasificador lineal con scikit-learn, utilizamos un parámetro llamado kernel de tipo lineal utilizando el comando *linear*.

En este caso para crear un clasificador no lineal con parámetros polinomiales, únicamente tenemos que asignar el parámetro kernel polinomial con el comando *poly*.

Veamos un ejemplo:

Para ello utilizaremos un set de datos creado con la función de make\_moons. Esta función nos ayuda a crear sets de datos con diferentes tendencias para practicar.

Para crear estos datos solo tenemos que importar la función, indicar cuántas instancias de datos queremos y qué nivel de ruido o aleatoriedad queremos generar.

from sklearn.datasets import make\_moons

X,y = make\_moons(n\_samples=100, noise=0.15)

Los datos se generan de esta forma:

Text

Description automatically generatedA picture containing chart

Description automatically generated

Tenemos valores que pertenecen a la instancia 0, y sus valores en x1 y x2. También tenemos instancias que pertenecen a 1, con sus valores en x1 y x2.

Para graficar el set de datos, x1 y x2, van a ser nuestras coordenadas, entonces vamos a crear una iteración que separe las instancias 0 y las instancias 1, y las grafique.

for class\_value in range(2):

row\_ix = np.where(y == class\_value)

pyplot.scatter(X[row\_ix, 0], X[row\_ix, 1])

pyplot.show()

De esta forma tenemos una gráfica así:

Chart, scatter chart

Description automatically generated

Ahora vamos a implementar el modelo de SVM.

svm = Pipeline([

("escalar", StandardScaler()),

("svc", SVC(kernel="poly", C=5, degree=3, coef0=1))

])

En este caso lo único que hicimos fue modificar nuestro pipeline que teníamos hecho para la clasificación lineal a clasificación polinomial. Esto se realizó añadiendo la especificación de poly en el kernel.

También, agregamos el parámetro degree, el cual nos deja elegir el grado de polinomio que queremos.

Y, por último, agregamos el parámetro coef0, el cual se recomienda siempre agregar, ya que controla cuánto se ve influenciado el modelo por el cambio de polinomio y hace que los valores se adapten mejor al margen de decisión.

Ejemplo, vamos a ver cómo se adaptaría un modelo de grado 3 con un coef0 en 1:

Chart

Description automatically generated

Ahora, así se vería si cambiamos el parámetro coef0 a 0.

Chart, scatter chart

Description automatically generated

Como podemos ver hay una gran diferencia entre cómo se adapta nuestro modelo a los datos dependiendo de los parámetros que selecciones.

Ahora vamos a comparar cómo se vería un polinomio de grado 3 contra uno de grado 10.

Chart

Description automatically generatedChart

Description automatically generated

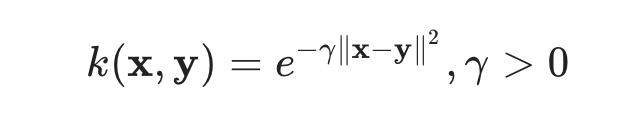
Como podemos ver, el polinomio 10 se adapta mejor a estos datos. Sin embargo, cuando se ajusta de esta manera, cabe la posibilidad de que exista un sobreajuste.

Una forma útil y popular para saber ¿qué grado de polinomio utilizar? Es por medio de un grid search que evalúe diferentes opciones.

### **Características de Similitud**

Otra técnica para enfrentarse a problemas no lineales es agregar variables usando una función de similitud. Esta función mide cuánto se asemeja cada instancia a cada punto de referencia.

Esta es una técnica matemática bastante avanzada que consiste en utilizar la función de Base Radial Gaussiana o Radial Basis Function (RBF).



Esta función lo que hace es, con el parámetro gamma γ, va probando diferentes relaciones de distancia entre los datos, creando diferentes dimensiones que pueden llegar hasta el infinito.

La función RGB gaussiana intenta darles a los datos una distribución normal, es decir en forma de campana. Al igual que la función logística, se intenta modelar los datos entre 0 y 1

Chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated

Ahora si tenemos un set de datos que se comporta de esta manera, en vez de elevar los datos a otras dimensiones (como en clasificación polinomial), el RBF intenta encontrar similitudes midiendo la distancia entre los datos.

Chart, line chart

Description automatically generated

A cada a cada curva encima de los datos se le conoce como landmark pero ¿cómo seleccionar que landmarks usar?

Lo que normalmente se hace es crear un landmark para cada instancia en el set de datos. Al hacer esto las dimensiones aumentan y a la par, las probabilidades de crear un set de datos linealmente separable también aumenta.

De esta manera se van creando landmarks hasta llegar a algo que se ve como esto, que dentro de las opciones habrá una forma que pueda ser linealmente separable.

Chart, scatter chart

Description automatically generated

Diagram

Description automatically generated

Bien, en resumen RGB con SVM, sirve cuando los datos no son linealmente separables, creando una serie de landmarks basándose en la distancia de los datos y su relación, para darles una nueva forma e intentar separarlos.

La explicación matemática detrás de RBF incluye varios temas complejos de álgebra lineal, series de Taylor, cálculos con infinito, entre otros términos. Para aquellos que quieran profundizar en estos temas les dejo un video para comenzar a hacerlo: <https://youtu.be/Qc5IyLW_hns>

En fin, ahora vamos a ver cómo se implementa un modelo de clasificación RGB con SVM.

### **Kernel RBF**

Al igual que el método polinomial, el método RBF puede ser un modelo útil en un proyecto de Machine Learning, aunque con un costo computacional importante cuando se trata de set de datos grandes.

Para implementar este modelo es necesario cambiarle el kernel con un comando llamado rbf.

svm = Pipeline([

("escalar", StandardScaler()),

("svc", SVC(kernel="rbf", C=5, gamma=5))

])

Como podemos ver agregamos el formato rbf al modelo y además le especificamos un parámetro gamma, que como vimos de este depende las comparaciones de similitud que se hacen en el modelo.

Vamos a ver como se adapta este tipo de modelo con diferentes valores en *gamma y C* a los datos en forma de medialuna que utilizamos anteriormente.

Empecemos con un valor bajo para C y para gamma.

* + - Gamma = 0.1 C= 0.001

Chart, scatter chart

Description automatically generated

Como podemos observar al tener bajos estos parámetros, el modelo no crea variables ni dimensiones adicionales y se ve casi como una clasificación lineal. Esto se debe a que el modelo es muy flexible y permite que haya muchas violaciones al momento de clasificar.

Probemos con la misma gamma, pero con un parámetro de C que tenga menos flexibilidad y siga el flujo de los datos.

* + - Gamma=0.1 y C = 1000

Chart

Description automatically generated

Como podemos ver el modelo intenta adaptarse más a la tendencia de los datos. Sin embargo, el modelo puede estar sobre ajustando.

Ahora vamos a ver qué pasa si balanceamos un poco más la flexibilidad y aumentamos el valor de gamma para balancear.

* + - Gamma = 5 y C = 10

Chart

Description automatically generated

En este caso ya se puede ver más claramente como el modelo RGB, encapsulo a la clase azul. Este es un ejemplo del porqué es recomendable usar el RGB cuando se tienen datos con tendencias no lineales.

Antes de concluir con el tema de clasificadores SVM es importante mencionar que existen más tipos de kernel. Entre ellos está el string que es utilizado para secuencias de ADN o clasificación de documentos de texto. Sin embargo, los más comunes fueron los anteriormente expuestos.

### **Complejidad Computacional**

Como mencionamos al inicio del capítulo el modelo de SVM se utiliza para sets de datos complejos de tamaño medio y pequeño, ya que los modelos más potentes como el polinomial y el RBF pueden llegar a ser lentos debido a la complejidad computacional que conlleva.

Si se quiere llegar a usar SVM en sets de datos más grandes, lo que se puede hacer es usar el módulo de Scikit LinearSVC, el cual está programado para hacer todo de forma lineal, por lo que consume menos memoria al hacer sus predicciones.

Con LinearSVC podemos utilizar clasificaciones lineales y polinomiales. Para generar clasificaciones lineales simplemente necesitamos hacer lo siguiente:

from sklearn.svm import LinearSVC

svm = Pipeline([

("escalar", StandardScaler()),

("svc", LinearSVC(C=10))

])

Como vemos el procedimiento es prácticamente el mismo que con SVC. Para crear un modelo polinomial con LinearSVC, tenemos que hacer lo siguiente:

from sklearn.preprocessing import PolynomialFeatures

svm = Pipeline([

("polinomial", PolynomialFeatures(degree=3)),

("escalar", StandardScaler()),

("svc", LinearSVC(C=10))

])

En este caso utilizamos el módulo que vimos en la regresión polinomial, el de polynomial features.

Recordemos que este módulo lo que hace es agregar variables polinomiales a nuestros datos. Una vez que esto sucede, pasamos al módulo de LinealSVC para crear el modelo de clasificación.

Por último, te preguntarás ¿Cómo puedo saber que kernel usar si no conozco la forma de mis datos?

La respuesta es fácil, normalmente el clasificador que genera mejores predicciones es el RBF. Si tu set de datos es pequeño o sabes que no va a consumir mucha memoria lo ideal sería empezar con este.

Si tu set de datos es bastante grande, lo mejor es empezar con la forma lineal, ya que esta toma menos memoria para generar el modelo.

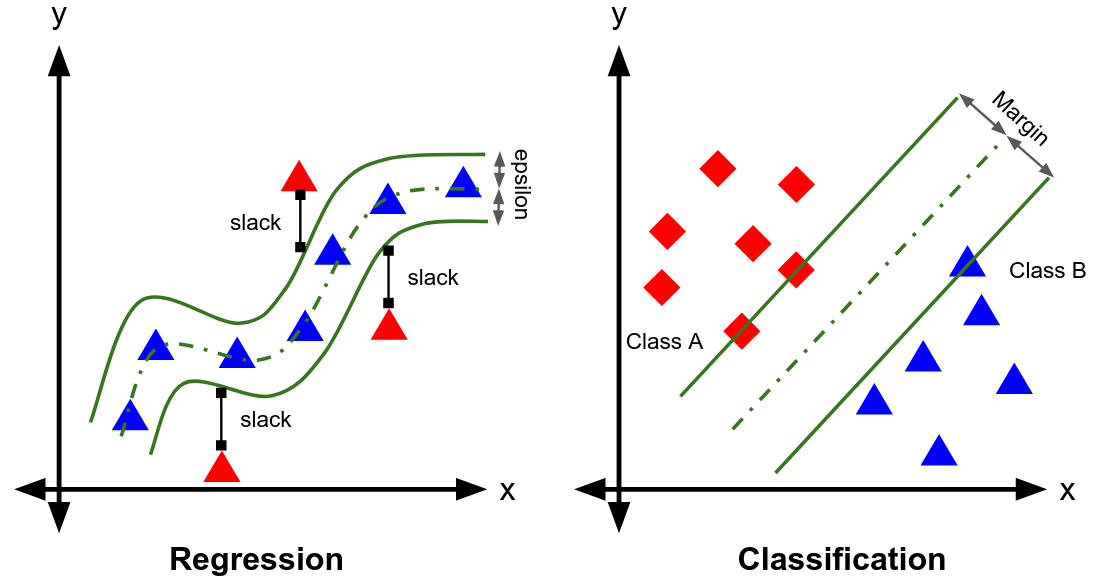
## **Regresión con SVM**

Como se mencionó antes el SVM es un algoritmo versátil, no solo funciona para clasificaciones lineales y no lineales, sino que también para regresiones lineales y no lineales.

El truco para pasar de un modelo de clasificación a uno de regresión es revertir su objetivo.

En un modelo de clasificación estamos empeñados en conseguir el mejor balance en el margen de clasificación y sus vectores de soporte. A la par, se es cuidadoso cometer el menor número de violaciones posibles (datos que quedan dentro del vector).

En cambio, en un modelo de regresión hacemos todo lo contrario. La regresión con SVM intenta incluir el mayor número de instancias dentro de los márgenes, a medida que una violación de calle se atribuye a aquellas instancias fuera de la calle.



En el caso de la regresión tenemos el parámetro épsilon, siendo el que abre y cierra el margen en los vectores de soporte.

Para implementar la **regresión SVM**, en vez de utilizar el módulo SVC y el LinearSVC, utilizaremos los análogos a ellos, pero en la regresión: SVR y Linear SVR.

Con SVR:

from sklearn.svm import SVR

svm = Pipeline([

("escalar", StandardScaler()),

("svr", SVR(kernel="linear",epsilon=1.5))

])

Con LinearSVR:

from sklearn.svm import LinearSVR

svm = Pipeline([

("escalar", StandardScaler()),

("svr", LinearSVR(epsilon=1.5))

])

Como podemos ver la forma de implementarlos es muy similar a la del SVC, pero en este caso el parámetro principal es épsilon, que delimita la anchura de la calle.

Al igual que con los modelos de clasificación también tenemos la posibilidad de usar el parámetro C para la regularización del margen de los vectores.

De igual manera si el data set que se está utilizando es demasiado grande, se recomienda utilizar el módulo lineal LinearSVR.

También se pueden hacer regresiones de forma polinomial, al igual que con el clasificador lo único que tenemos que hacer es seleccionar el kernel polinomial e indicar el grado de polinomio que queremos probar.

svm = Pipeline([

("escalar", StandardScaler()),

("svr", SVR(kernel="poly",C = 10, degree=3,epsilon=1.5))

])

Así es como se vería una regresión polinomial con SVM.

Chart, scatter chart

Description automatically generated

## **Lo Que Hay Por Debajo Del SVM**

En esta sección vamos a explicar a detalle cómo es que SVM puede hacer sus predicciones y como es que funcionan los diferentes modelos. Para desarrollar ML no necesariamente tienes que tener esta fundamentación, aunque nunca está de más el saber cómo es que funcionan los modelos que utilizas.

Empecemos. El objetivo principal sobre los SVM es tener el margen más ancho posible con el menor número de violaciones.

* + **Margen Duro**

Para ver el lado matemático de esto, empecemos definiendo las ecuaciones de nuestros vectores de soporte.

A picture containing text, sky, antenna

Description automatically generated

Como vemos nuestra línea central o frontera de decisión es igual a

Donde:

* es nuestro vector que contiene todos los valores de nuestra variable predictora.
* es nuestro intercepto o parámetro de intercepción.
* es el vector que contiene los valores necesarios para que toda la ecuación sea 0.

Por ejemplo, si tenemos la ecuación y sabemos que y entonces , para ajustarse a la frontera de decisión w pasará a ser 3, entonces

Es importante señalar que la frontera de decisión es igual a 0 porque, este es nuestro punto de partida para medir la distancia de los otros vectores de soporte.

Recordemos que w y x son vectores, lo que significa que tenemos un grupo de valores dentro de cada uno de ellos.

Al multiplicar vectores multiplicamos cada valor por su valor correspondiente por lo que: es también .

Y si decimos que , entonces ¿por qué decimos que el soporte de arriba es ?

Cuando queremos recorrer los vectores de soporte, realmente tenemos la siguiente ecuación , en realidad, le pasamos un vector unitario multiplicado por un parámetro k a nuestro vector de valores .

Si multiplicamos w tenemos que .

Como sabemos que , entonces podemos decir que , si lo simplificamos tendremos que .

Ambos vectores de soporte tienen la misma distancia a nuestra frontera de decisión, y ahora sabemos que entre más grande sea nuestro parámetro k, más grande será el margen de los vectores de soporte.

Como dijimos al inicio, uno de los objetivos del SVM es que el margen sea lo más grande posible (pretendamos que las violaciones de datos no existen por ahora).

Entonces uno de nuestros objetivos es que k sea lo más grande posible y como , entonces tenemos que encontrar el menor número de posible. Por ejemplo, si es muy grande como 1000 entonces k = 1/1000 = 0.001, pero si tiene un valor pequeño como 0.0001, entonces k = 1/0.0001 = 10000.

Ante esto, surge el primer objetivo matemático de la explicación: minimizar .

A picture containing text, sky, antenna

Description automatically generated

Ahora que sabemos que tenemos que minimizar , hay que echar un vistazo a la grafica y ver que hay un par de cosas que son muy importantes.

Para controlar que no haya violaciones en los vectores de soporte, tenemos que definir lo siguiente.

Todos los valores de la clase verde están clasificados con un valor de -1 o inferior, mientras que todos los valores de la clase roja están clasificados como un valor de 1 o superior.

Por lo que es importante cuidar esas 2 restricciones:

Por lo tanto, imaginemos que tenemos estos vectores:

En este caso tenemos un set de datos que cumple nuestras condiciones, ya que .

Al tener un valor de positivo lo clasificamos como clase 1.

Mientras que y2 está clasificado como -1, por lo que w2x2 tiene un valor negativo.

Este proceso se repite para cada uno de los datos en nuestro set de datos, por lo que todos nuestros valores van a ser o un 1 o un -1 siendo un modelo de clasificación binario.

Podemos concluir que tenemos un modelo que intenta minimizar para tener un margen lo más grande posible.De esta manera nos aseguramos de que todas nuestras clasificaciones queden por fuera del margen 1 y -1.

Esto funciona para modelos donde queremos tener un margen pesado.

* + **Margen Suave**

Para tener un margen suave el modelo funciona de una forma completamente diferente.

A diferencia del margen pesado el margen suave acepta valores dentro del margen de decisión, mediante una serie de penalizaciones que se rigen por la función de perdida llamada Hinge Loss o pérdida de Bisagra:

Esta función básicamente nos dice que valor está fuera de él margen de decisión *1* y *-1* y qué instancia está dentro.

Por ejemplo:

Con estos 2 ejemplos tenemos:

lo que es igual a por lo que la función selecciona 0 ya que obviamente es mayor a -14.

Cuando esta función detecta ceros no pasa nada ya que esto indica que los valores están adecuadamente afuera del margen, y no tiene alguna perdida.

Ahora hay 2 tipos de penalizaciones:

Chart, radar chart, scatter chart

Description automatically generated

Como vemos en la clase roja, hay 2 violaciones una que está completamente del lado de la clase azul y una que está del lado rojo, aunque cruza el margen de decisión.

La función de pérdida de bisagra nos ayuda a darles diferentes penalizaciones a este tipo de violaciones. Sin embargo, estás penalizaciones se hacen acorde al tipo de error. Por ejemplo, el punto que está dentro de la clase azul merece una penalización más grande que el que está dentro del margen ¿por qué? Porque, aunque estaba dentro del margen, la clasificación era correcta.

El punto rojo más alejado se vería así: por lo que tenemos por lo que su pérdida sería de 11.

Mientras que el otro punto se vería de la siguiente manera: , o , por lo que su pérdida sería de 0.4.

Entonces todo el conjunto de un modelo suave se enfoca igualmente en minimizar , aunque también en minimizar el hinge loss de todo el set de datos, para que la pérdida sea la menor posible.

Afortunadamente no tenemos que optimizar estos modelos manualmente, ya que módulos de scikit learn ya lo hacen por nosotros mediante algoritmos de programación cuadrática.

### **El Kernel Del SVM**

Como vimos antes, al implementar los modelos de SVM y sus diferentes modos de utilización lo único que bastaba para hacerlo era seleccionar el parámetro kernel y especificar qué tipo de modelo queríamos utilizar.

Vamos a dar una pequeña explicación de cómo es que funciona matemáticamente este kernel. Sin embargo, no profundizaremos mucho debido a la complejidad de los conceptos.

El inicio comienza con un problema de optimización con restricción llamado el problema primal.

Todo empieza con un problema de optimización con restricciones que surgió en el mundo de las matemáticas que se llama problema primal.

Después de hacer ciertas pruebas con modelos de optimización, encontraron otro problema llamado el problema dual.



Se dieron cuenta que la forma de resolver el problema primal se podría aplicar en la resolución de problemas de maximización y el problema dual en la resolución de problemas de minimización. Ojo: sin dejar de lado las respectivas restricciones de cada una.

Bien, como sabemos nuestra intención al modelar SVM es minimizar. Entonces, resulta que la forma de resolver el problema dual se adapta muy bien a cómo se resuelven los problemas de minimización para SVM haciendo el cambio de kernel posible.

A grandes rasgos, las personas se dieron cuenta que para utilizar SVM lineal, polinomial o RBF, no se necesitaba transformar los valores del set de entrenamiento, proceso que, por cierto, hacía a los modelos lentos y con bajos rendimientos. Con el problema dual, se creó un set de ecuaciones matemáticas que dependen del mismo set de entrenamiento, tan solo agregando diferentes parámetros o efectuando diferentes operaciones “simples” como multiplicación división, elevar al cuadrado, etc.

Text, letter

Description automatically generated

Se creó este set de diferentes ecuaciones, el cual con el módulo de scikit, solo tiene que cambiar y agregar estos diferentes parámetros para que podamos obtener el tipo de modelo que necesitamos, cuando antes de la existencia de estas fórmulas era prácticamente imposible.

Si se desea profundizar más en cómo funciona el kernel del SVM, te dejo un video que encontré el cual explica bastante bien su funcionamiento en solo 15 minutos, aunque igualmente hacen falta conocimientos avanzados en matemáticas.

<https://www.youtube.com/watch?v=OmTu0fqUsQk>

### **SVM Online**

Antes de terminar con este capítulo, vamos a darle un vistazo a los modelos de clasificación SVM con aprendizaje online.

Si recordamos, un modelo de aprendizaje permite al modelo seguir aprendiendo conforme se añaden nuevos sets de datos. Mientras que uno offline es aquel que se queda estático para cualquier set de datos nuevo.

Una forma de hacer que tu modelo de SVM tenga un aprendizaje online es usando el descenso de gradiente. Sin embargo, es poco común utilizarlo ya que este converge mucho más lento en SVM que usando la programación cuadrática que vimos anteriormente.

Realmente cuando se quiere tener un modelo que implemente el aprendizaje online se utilizan las redes neuronales, lo cual se verá más adelante, aunque si te interesa saber cómo implementar estos modelos partiendo solo del SVM estos artículos son bastante buenos.

<https://jmlr.org/papers/v6/bordes05a>