Fundamentos de IA y Machine Learning

Lección 4: Modelos de Regresión y Clasificación II

eip ESCUELA INTERNACIONAL POSGRADOS

ÍNDICE

Modelos de Regresión y Clasificación II 2

Presentación y objetivos 2

1. Introducción 3

1.1. Contextualización de la lección 3

1.2. Formulación de un problema de regresión 4

1.3. Formulación de un problema de clasificación 4

2. Redes neuronales artificiales 5

2.1. Fundamentos y modelo 5

2.2. Entrenamiento y estimación de los parámetros 8

2.3. Consideraciones importantes 9

2.4. Ejemplo de aplicación 10

3. Máquinas de vectores soporte 13

3.1. Fundamentos y modelo 13

3.2. Entrenamiento y estimación de los parámetros 15

3.3. Consideraciones importantes 17

4. Ensembles 18

4.1. Fundamentos 18

4.2. Diversidad 19

4.3. Generación de ensembles 19

4.4. Consideraciones importantes 22

5. Puntos clave 23

Modelos de Regresión y Clasificación II

PRESENTACIÓN Y OBJETIVOS

Esta lección es una continuación de la lección 3, por lo que se verán otros modelos de clasificación y regresión.

Objetivos

Al finalizar esta lección serás capaz de:

- Generar, entrenar e interpretar una red neuronal artificial.
- Conocer el fundamento de los vectores de máquina soporte para clasificación (*SVM*) y regresión (*SVR*).
- Hacer uso de ensembles conociendo sus fundamentos.

1. Introducción

1.1. Contextualización de la lección

Como se observó en lecciones anteriores, las fases en las que se componen la resolución de un problema de aprendizaje automático son las siguientes.

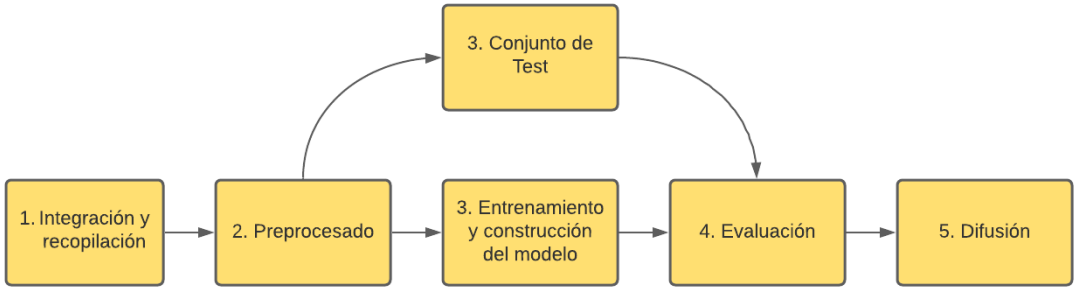

Figura 1.1: Fases de un problema de aprendizaje automático.

Una vez que se tiene el conocimiento necesario para recopilar los datos, preprocesarlos, elegir las técnicas de evaluación y las métricas de rendimiento, debemos determinar qué modelo es el más interesante.

Aunque existen muchos modelos en la literatura, en esta lección nos vamos a centrar en:

- **Redes neuronales artificiales.**
- **Vectores de máquina soporte (*SVM* y *SVR*).**
- **Ensembles.**

1.2. Formulación de un problema de regresión

Cualquier problema de regresión se puede formular de la siguiente forma:

- Disponemos de un espacio de entrada $\boldsymbol{X}$ compuesto por patrones etiquetados con $Y \subseteq \mathbb{R}$.
- Cada patrón se representa por un vector de características de dimensión K, $\boldsymbol{x} \in \boldsymbol{X} \subseteq \mathbb{R}^K$ y un valor continuo $y \in Y \subseteq \mathbb{R}$.
- El objetivo es aprender una función f que relacione los datos del espacio de entrada $\boldsymbol{X}$ al conjunto de valores continuos finito Y.
- El conjunto de entrenamiento $\boldsymbol{T}$ está compuesto de N patrones:

$$T = (\boldsymbol{x}_i, y_i) : \boldsymbol{x}_i \in \boldsymbol{X}, \boldsymbol{y_i} \in Y (i = 1, \dots, n), \text{ con } x_i = \left(x_{i,1}, x_{i,2}, \dots, x_{i,K}\right).$$

1.3. Formulación de un problema de clasificación

Análogamente, cualquier problema de clasificación se puede formular de la siguiente forma:

- Disponemos de un espacio de entrada $\boldsymbol{X}$ compuesto por patrones etiquetados con $C = \{C_1, C_2, \dots, C_Q\}$ donde Q es el número de clases.
- Cada patrón se representa por un vector de características de dimensión K, $\boldsymbol{x} \in \boldsymbol{X} \subseteq \mathbb{R}^K$ y una etiqueta de clase $y \in C$.
- El objetivo es aprender una función f que relacione los datos del espacio de entrada $\boldsymbol{X}$ al conjunto finito C.
- El conjunto de entrenamiento $\boldsymbol{T}$ está compuesto de N patrones:

$$T = (\boldsymbol{x}_i, y_i) : \boldsymbol{x}_i \in \boldsymbol{X}, \boldsymbol{y_i} \in C (i = 1, \dots, n), \text{ con } x_i = \left(x_{i,1}, x_{i,2}, \dots, x_{i,k}\right).$$

2. REDES NEURONALES ARTIFICIALES

Las redes neuronales artificiales son modelos que simulan los sistemas nerviosos biológicos. Debido a sus potentes propiedades y características, las redes neuronales se aplican en muchísimos problemas del mundo real. Las redes neuronales *Feed-Forward* (*FNN)* con una capa oculta son el tipo más básico, pero también uno de los más utilizados.

2.1. Fundamentos y modelo

Formalmente, las redes neuronales *FNN* son una generalización de un modelo de regresión estándar con la siguiente expresión:

$$y(x, \boldsymbol{\theta}) = \beta_0 + \sum_{j=1}^{M} \beta_j \, B_j(x, w_j)$$

Donde M es el número de neuronas en capa oculta, $B_j(\boldsymbol{x}, \boldsymbol{w_j})$ es la función base para la j-ésima neurona de capa oculta, $\boldsymbol{x}$ representa las características de entrada, $\boldsymbol{w_j}$ es el peso de la conexión entre la capa de entrada con la j-ésima neurona, β_j representa la conexión entre la neurona j de la capa oculta con la capa de salida, e $y(\boldsymbol{x}, \boldsymbol{\theta})$ es la función a optimizar. $B_j(\boldsymbol{x}, \boldsymbol{w_j})$ está formado por una función de activación que calcula el valor de entrada total que llega al nodo y una función de transferencia que es la salida de la activación de la neurona. La representación gráfica sería la siguiente:

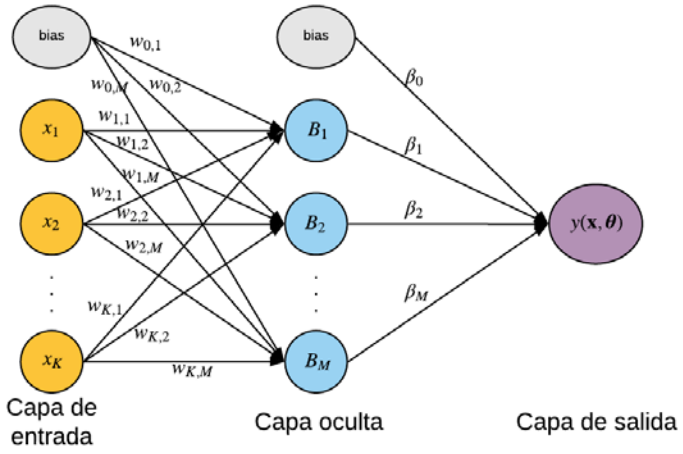

Figura 2.1: Modelo de red neuronal.

Generalmente, se consideran **dos tipos de funciones de activación** de las neuronas.

- **Modelo aditivo:** es el más común y su expresión es la siguiente:

$$B_j(\mathbf{x}, \boldsymbol{w_j}) = h(w_{0,j} + w_{1,j}x_1 + w_{2,j}x_2 + \cdots + w_{K,j}x_K)$$

Siendo $w_j = \{w_{0,j}, w_{1,j}, \ldots, w_{K,j}\}$ los pesos de entrada para las conexiones al nodo j, $h(\cdot)$ la función de transferencia, y $w_{0,j}$ el sesgo. Hay distintos tipos de nodos aditivos, por ejemplo, el conocido perceptrón usa una función de paso como las funciones sigmoidales o tangentes hiperbólicas, entre otras.

- **Modelo multiplicativo:** es una estrategia más reciente usada para aquellos casos en los que existe una gran interacción entre las variables de entrada, y las regiones de decisión no son separables en hiperplanos. La expresión sería:

$$B_j(x, w_j) = x_1^{w_{1,j}} \cdot x_2^{w_{2,j}} \cdot \ldots \cdot x_K^{w_{K,j}}$$

Donde $w_{0,j}$ no tendría sentido.

Considerando el espacio característico de entrada de las funciones base, podemos clasificarlas en:

- **Funciones locales o *kernel*:** presentan un mayor valor sobre una región específica del espacio de entrada. Son buenas para aproximar datos aislados, pero más pobres en entornos globales y cuando el número de entrada es elevado. Un ejemplo son las funciones de base radial o *RBF*.

- **Funciones globales o de proyección:** son mejores en el caso contrario, es decir, se comportan mejor para entornos globales, pero de forma peor en la aproximación de datos aislados. Un ejemplo son las unidades sigmoide (*SU*) y las unidades producto (*PU*).

Existen tres tipos de redes neuronales *FNN* muy extendidas en la literatura dependiendo del tipo de funciones de base que se utilizan en la capa oculta:

- **Redes neuronales sigmoide (*SUNN*):** son también conocidas como perceptrón multicapa (*MLP*) y sus nodos presentan un modelo de proyección aditivo. Una función sigmoide se define como:

$$B_j(\mathbf{x}, \mathbf{w_j}) = \frac{1}{1 + e^{-(w_{0,j} + w_{1,j}x_1 + w_{2,j}x_2 + \cdots + w_{K,j}x_K)}} = \frac{1}{1 + e^{-(w_{0,j} + \sum_{i=1}^{K} w_{i,j}x_i)}}$$

- **Redes neuronales producto (*PUNN*):** permiten a la red neuronal formar combinaciones de alto orden de las entradas, con las ventajas de una mayor capacidad de información y arquitecturas más simples cuando estas interacciones están presentes en las variables de entrada. Estas redes utilizan nodos producto (*UP*) en la capa oculta, siguiendo un modelo de proyección multiplicativa con la función de salida:

$$B_j(\mathbf{x}, \mathbf{w_j}) = x_1^{w_{1,j}} \cdot x_2^{w_{2,j}} \cdot \ldots \cdot x_K^{w_{K,j}} = \prod_{i=1}^{K} x_i^{w_{i,j}}$$

- **Redes neuronales RBF:** consideran funciones de transferencia de tipo *kernel* denominadas *RBF* (del inglés *Radial Basis Function*). Cada *RBF* realiza una aproximación local independiente del espacio de entrada, normalmente utilizando una función gaussiana. Luego, la capa lineal de salida combina el efecto de los nodos ocultos. La idea es que cada nodo se sitúa en una región del espacio de entrada con un radio determinado y el aprendizaje se realiza moviendo los nodos por este espacio, variando el centro y este radio, para ajustar los datos de entrenamiento. La función *RBF* se formula cómo:

$$B_j(x, w_j) = e^{-\frac{1}{2}\left(\frac{d(\boldsymbol{x}, \boldsymbol{w_j})}{r_j}\right)^2}$$

Siendo:

$$d(x, w_j) = \sqrt{\sum_{i=1}^{K} (x_i - w_{i,j})^2}$$

2.2. Entrenamiento y estimación de los parámetros

El procedimiento de aprendizaje en las redes neuronales consiste en estimar los valores $\boldsymbol{\theta}$ y la arquitectura de la red neuronal, es decir el número de nodos en capa oculta M, el número de conexiones entre los nodos e incluso el número de capas ocultas. Sin embargo, asumiendo una arquitectura predefinida invariable, el aprendizaje de los pesos de las conexiones se realiza comúnmente mediante un algoritmo de descenso de gradiente, como es el caso del algoritmo de retropropagación. Este proceso consta de los siguientes pasos:

1. Se propaga la entrada hacia la salida, obtiene el valor estimado por nuestra función: $\hat{y}(\boldsymbol{x}, \boldsymbol{\theta}) = \beta_0 + \sum_{j=1}^{M} \beta_j \, B_j(\boldsymbol{x}, \boldsymbol{w_j})$
2. Obtenemos el error cometido por la red $E = \left(y - \hat{y}(\boldsymbol{x}, \boldsymbol{\theta})\right)^2$.
3. Se calcula el gradiente del error:

$$\nabla E = \frac{\partial E}{\partial \boldsymbol{\theta}} = \left\{ \frac{\partial E}{\partial w_{0,1}}, \ldots, \frac{\partial E}{\partial w_{0,M}}, \ldots, \frac{\partial E}{\partial w_{N,M}}, \frac{\partial E}{\partial \beta_0}, \ldots, \frac{\partial E}{\partial \beta_M} \right\}$$

4. Una vez que hemos obtenido el gradiente, debemos actualizar los pesos. Siendo $\Delta w_{i,j} = \frac{\partial E}{\partial w_{i,j}}$, se tiene la siguiente expresión:

$$w_{i,j} = w_{i,j} - \eta \Delta w_{i,j} - \mu \left(\eta \Delta w_{i,j}(t-1) \right)$$

Dónde η y μ son la tasa de aprendizaje y el momento, respectivamente.

5. Repetir el procedimiento hasta que el algoritmo converja.

La tasa de aprendizaje controla que los pasos que se van dando no sean muy pequeños o muy grandes. Ajustar el valor de η es difícil, ya que si es demasiado grande podemos provocar oscilaciones, mientras que si es demasiado pequeño, necesitamos de muchas iteraciones. El concepto de momento mejora la convergencia y permite que los cambios anteriores afecten en la dirección del movimiento actual. La idea es que en un ejemplo como el siguiente, el momento haga que se pueda escapar del óptimo local.

Pese a que la derivada te diga que vayas a la izquierda, la “inercia” puede llevarte a seguir hacia la derecha.

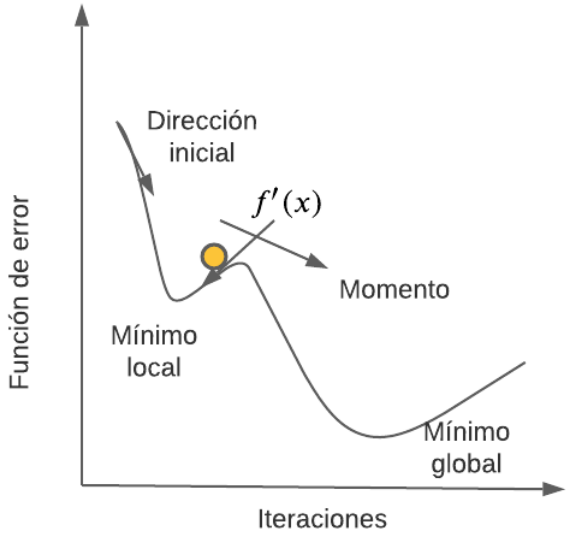

Figura 2.2: Ejemplo de momento.

2.3. Consideraciones importantes

Se pueden destacar las siguientes consideraciones:

- Se trata de un modelo probabilístico no lineal.
- No obstante, la capa de salida se puede formular como una función lineal de funciones base no lineales.
- Se puede utilizar tanto en regresión como clasificación.
- Pueden sobreentrenar.
- El coste de entrenamiento es alto.
- Además, se necesitan estimar varios hiperparámetros: número de capas ocultas, número de neuronas en capa oculta, número de iteraciones, tasa de aprendizaje y momento, entre otros.
- Normalmente, son difíciles de interpretar.

Importante

Es importante conocer el concepto de *Deep Learning*. Para ello, consultar el enlace: **https://www.ticportal.es/glosario-tic/deep-learning-dl**

2.4. Ejemplo de aplicación

Haciendo uso de la base de datos Iris vista anteriormente, reducida a dos dimensiones, se ha ejecutado un algoritmo para estimar la siguiente red neuronal de unidades sigmoide:

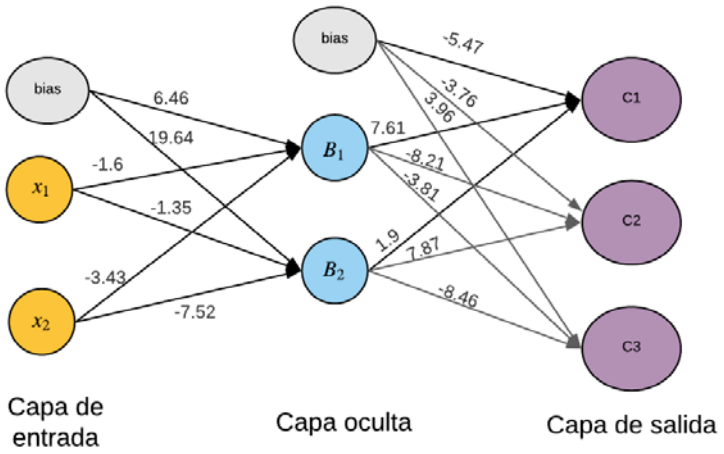

Figura 2.3: Red neuronal para base de datos IRIS.

1. Hallar las predicciones del modelo para el siguiente conjunto de *test*.
2. Evaluar el rendimiento del clasificador en dicho conjunto.

Patrón	X_1	X_2	Clase
1	1.4	0.2	1
2	1.5	0.2	1
3	1.5	0.4	1
4	1.6	0.6	2
5	4.0	1.3	2
6	4.2	1.2	2
7	4.1	1.3	3
8	5.1	1.9	3
9	5.0	1.9	3
10	5.1	1.8	3

Solución:

1. Calculamos para cada patrón la función de transferencia B_1 y B_2, y la salida para cada una de las clases. En la capa de salida también se ha utilizado una función sigmoidal, ya que, de este modo tenemos la probabilidad de pertenencia de cada patrón a cada clase.

Patrón	X_1	X_2	B_1	B_2	$p(C1)$	$p(C2)$	$p(C3)$	Predicha	Real
1	1.4	0.2	0.97	1.00	0.98	0.02	0.00	1	1
2	1.5	0.2	0.97	1.00	0.98	0.02	0.00	1	1
3	1.5	0.4	0.94	1.00	0.97	0.03	0.00	1	1
4	1.6	0.6	0.86	1.00	0.95	0.05	0.00	1	2
5	4.0	1.3	0.01	0.99	0.03	0.98	0.01	2	2
6	4.2	1.2	0.01	0.99	0.03	0.98	0.01	2	2
7	4.1	1.3	0.01	0.99	0.03	0.98	0.01	2	3
8	5.1	1.9	0.00	0.18	0.01	0.09	0.92	3	3
9	5.0	1.9	0.00	0.20	0.01	0.10	0.91	3	3
10	5.1	1.8	0.00	0.32	0.01	0.22	0.78	3	3

2. Una vez se han realizado las predicciones, se calcula la matriz de confusión, y las métricas CCR y $Kappa$.

$$\begin{pmatrix} 3 & 0 & 0 \\ 1 & 2 & 0 \\ 0 & 1 & 3 \end{pmatrix}$$

$$CCR = \frac{1}{n}\sum_{j=1}^{J} n_{jj} = 80\%$$

$$Kappa = \frac{p_0 - p_e}{1 - p_e} = \frac{CCR - \frac{1}{n^2}\sum_{j=1}^{J} n_{j\cdot}\, n_{\cdot j}}{1 - \frac{1}{n^2}\sum_{j=1}^{J} n_{j\cdot}\, n_{\cdot j}} = \frac{0.8 - 0.33}{1 - 0.33} = 0.70$$

Para cada una de las clases consideramos como positiva dicha clase y negativa el resto. Así, se forman las siguientes matrices de confusión:

Clase 1: $\begin{pmatrix} 3 & 0 \\ 1 & 6 \end{pmatrix}$ Clase 2: $\begin{pmatrix} 2 & 1 \\ 1 & 6 \end{pmatrix}$ Clase 3: $\begin{pmatrix} 3 & 1 \\ 0 & 6 \end{pmatrix}$

A partir de aquí, calculamos las métricas para cada una de las clases:

Clase	$\boldsymbol{Sensibilidad}$	$\boldsymbol{FP\ Rate}$	$\boldsymbol{Especifidad}$	$\boldsymbol{Precision}$	$\boldsymbol{F-Score}$
1	1	0.143	0.857	0.750	0.857
2	0.667	0.143	0.857	0.667	0.667
3	0.750	0	1	1	0.857
Promedio	0.806	0.095	0.905	0.806	0.794

3. MÁQUINAS DE VECTORES SOPORTE

El modelo SVM (por sus siglas en inglés *Support Vector Machines)* surgió como una generalización del clasificador de margen máximo. Su buen rendimiento en una gran variedad de problemas ha hecho que sea considerado como uno de los mejores clasificadores.

3.1. Fundamentos y modelo

En un problema binario, dado un conjunto de entrenamiento, donde cada ejemplo pertenece a una de dos categorías, un algoritmo de entrenamiento de SVM construye un modelo que asigna etiquetas a un conjunto de datos no visto.

Concretamente, un modelo de SVM es una representación de ejemplos como puntos en el espacio, mapeados de forma que dichos ejemplos pertenecientes a distintas categorías estén divididos por un claro espacio o margen que debe ser lo más ancho posible. Los nuevos patrones son después mapeados en el mismo espacio y la predicción se basa en determinar en qué lado del margen caen.

La idea básica de las máquinas de vector soporte es encontrar el hiperplano que define la máxima separación entre patrones de dos clases diferentes. Formalmente:

$$f(x) = \hat{y} = sgn(< w \cdot \phi(x) > + b)$$

donde $\hat{y} = +1$ si x corresponde a la clase positiva y $\hat{y} = -1$ en otro caso.

De una forma muy intuitiva se puede observar el fundamento de este tipo de modelo de aprendizaje automático.

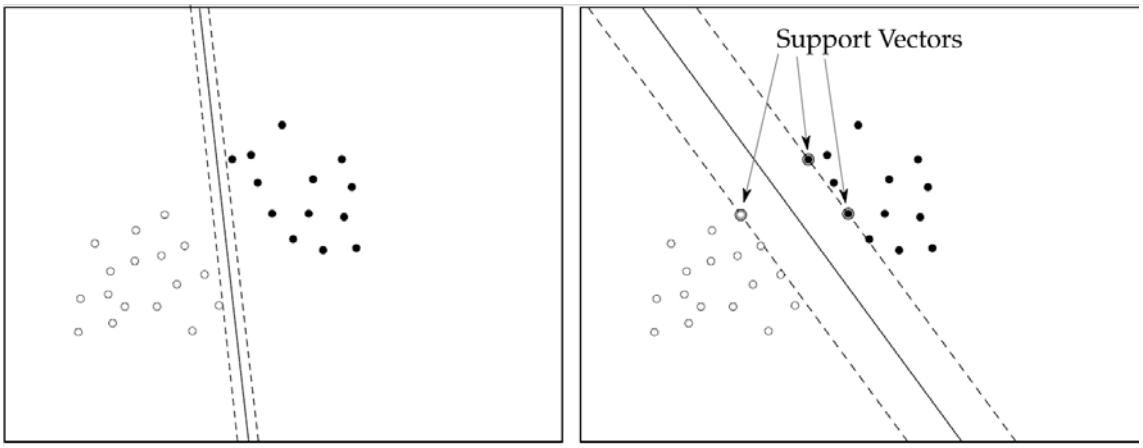

Figura 3.1: Márgenes pequeño y más grande entre los patrones de dos clases.

En la Figura 3.1 se observa cómo pueden existir una infinidad de márgenes que separan a ambas clases, sin embargo, la idea de SVM es elegir el margen más ancho entre ambas clases. Aquellos puntos que están en la frontera del margen son los denominados vectores soporte.

El ejemplo anterior muestra una separación en un espacio lineal. No obstante, esta idea se puede expandir a datos no lineales separables. De esta forma se pueden crear márgenes que separan datos que no son linealmente separables.

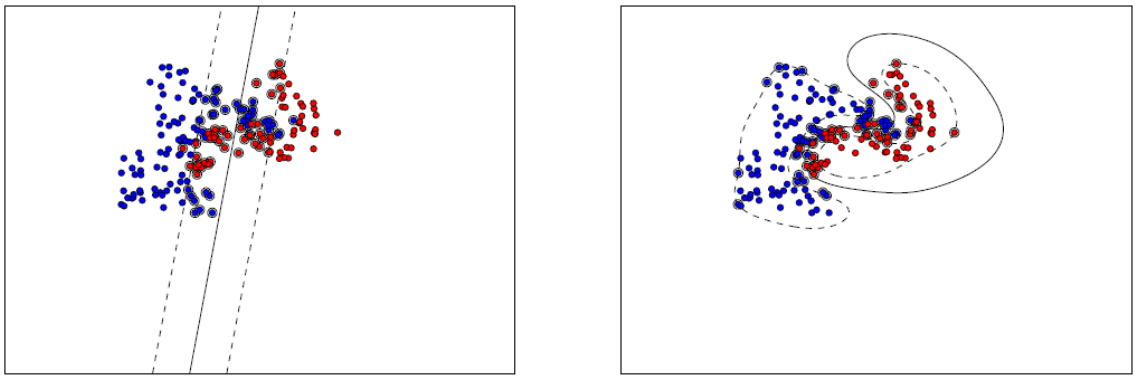

Figura 3.2: SVM lineal vs SVM no lineal.

3.2. Entrenamiento y estimación de los parámetros

En resumen, SVM intenta resolver un problema de optimización intentando aumentar la distancia del margen de decisión entre las clases y maximizando el número de puntos correctamente clasificados. El modelo de cálculo del hiperplano para el caso no lineal es complejo y requiere de habilidades matemáticas de optimización. Por tanto, en esta asignatura nos centraremos en la idea que subyace y fundamenta a los SVM, así como los hiperparámetros que son necesarios ajustar y que efecto provocan en nuestro clasificador.

El primer hiperparámetro es el coste c. Si c es pequeño, la penalización por puntos mal clasificados es baja. Esto supondría la creación de un margen más ancho a pesar de un mayor número de errores de clasificación. En el caso contrario, si c es grande, el clasificador trataría de reducir los errores de clasificación con un margen más pequeño. La penalización de cada ejemplo es directamente proporcional a la distancia con respecto al límite de decisión.

En el caso de los SVM no lineales con funciones RBF, existe otro parámetro denominado gamma (γ) que controla la distancia de la influencia de un solo punto de entrenamiento. Valores bajos implican un radio grande de similitud por lo que se agrupan más puntos. Para valores altos, los puntos deben estar muy cerca para ser considerados del mismo grupo. Así, un valor muy alto de γ hace que el SVM pueda sobreentrenar.

Otro de los parámetros a ajustar es determinar el tipo de *kernel* a utilizar. Los más conocidos son el lineal, el polinomial, el radial (RBF) o sigmoide. La utilización de uno u otro dependerá de los datos del problema.

Una visualización del efecto que produce aumentar el valor de c (fijando el parámetro γ) es el siguiente, donde podemos ver como para el primer valor, el modelo está infraentrenado y para el último, sobreentrenado.

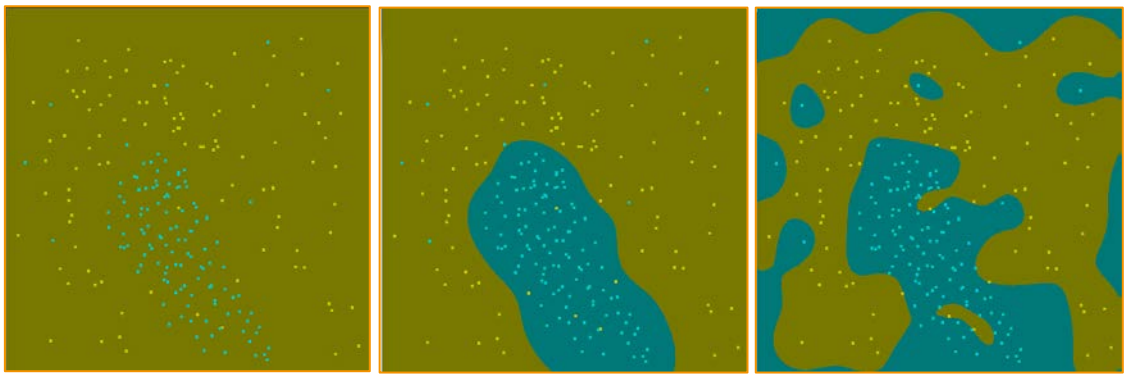

Figura 3.3: Efecto de variar el valor de c.

De la misma forma, el efecto visual de modificar el parámetro γ se puede observar en el ejemplo siguiente:

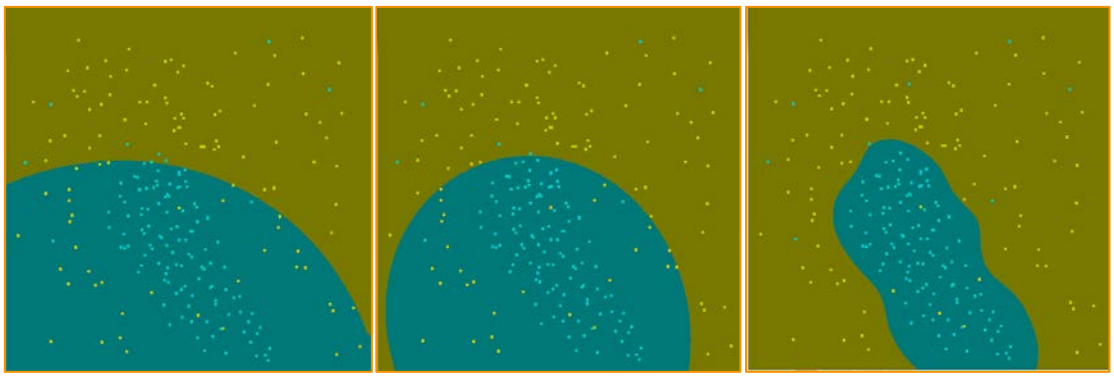

Figura 3.4: Efecto de variar el valor de γ.

3.3. Consideraciones importantes

A modo de resumen, y como consideraciones más importantes tenemos:

- SVM trata de encontrar el hiperplano que mejor separa dos clases.
- Se pueden utilizar en clasificación (SVM) o regresión (SVR).
- En general, su rendimiento es muy bueno para bases de datos con pocas instancias y muchas características.
- Se dice que para pocos patrones (< 2000 – 10000 dependiendo de la dimensionalidad) los modelos no lineales son muy potentes, previo ajuste de los hiperparámetros c y γ.
- Para bases de datos mayores, el modelo lineal es bastante competitivo.
- El modelo es no lineal y es más robusto al problema de alta dimensionalidad.
- Hay que ajustar bien los parámetros para evitar el problema del sobreentrenamiento.
- No son interpretables.
- Aunque existen propuestas para la resolución distribuida del problema de optimización, suelen ser ineficientes para grandes volúmenes de datos.

Importante

Se ha tratado el fundamento de los SVM de forma bastante superficial. Como se ha comentado, el fundamento matemático es complejo por lo que su aprendizaje puede llevar bastante tiempo. Se aconseja, si estáis interesados, hacer una búsqueda bibliográfica acerca de estos modelos.

4. ENSEMBLES

En aprendizaje automático definimos un ensemble como un conjunto de modelos diferentes entre sí que unen sus predicciones con el propósito de hacer una predicción más robusta.

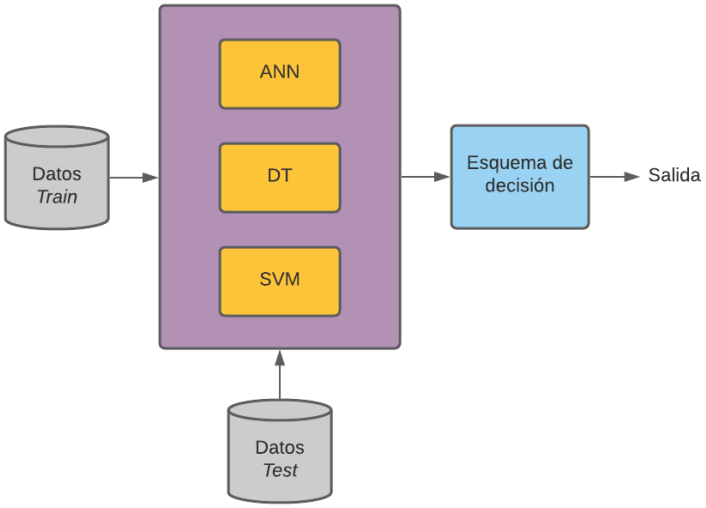

Figura 4.1: Ejemplo de ensemble con tres clasificadores base.

4.1. Fundamentos

Podemos justificar el uso de ensembles con las siguientes ideas:

- Se basan en la premisa de que cuantos más elementos decidan más robusta será la decisión.
- Los clasificadores o regresores suelen entrenar con procedimientos que pueden caer en óptimos locales. La agregación de sus predicciones podría acercarnos al óptimo global.
- Si distinguimos entre modelo fuerte como aquél que tiene una precisión deseada y modelo débil como aquél ligeramente superior a un modelo aleatorio, la idea es hacer ensembles de modelos débiles para construir un modelo fuerte.

4.2. Diversidad

Se considera que un ensemble aporta más que un modelo único en base a la **diversidad** de modelos. La intuición dice que los predictores base deben ser precisos y no cometer errores coincidentes.

Si tenemos un predictor que no tiene errores, entonces no necesitamos un ensemble. Si por el contrario sí tiene errores, buscamos otro que tenga errores en patrones diferentes. Por tanto, debe haber diversidad en las predicciones de los predictores base.

La diversidad se puede conseguir:

- Utilizando diferentes modelos (árboles, ANN, SVM, ...). Por lo general, a mayor número de predictores base, mejor será la predicción del ensemble, sin embargo, esto supone un gran coste computacional.
- Manipulando los predictores. Por ejemplo, teniendo configuraciones iniciales distintas de los predictores o manipulando los parámetros de entrenamiento.
- Diferencia en la población de los datos. Seleccionar un conjunto de datos distinto a cada predictor base.
- Selección de características. Los diferentes predictores pueden entrenarse sobre diferentes atributos de los patrones.

4.3. Generación de ensembles

Una de las maneras de generar ensembles es la denominada *Bagging*. La forma de conseguir que los errores se compensen entre sí es que cada modelo se entrena con un subconjunto de entrenamiento. Así, se generan tantos subconjuntos como clasificadores/regresores base tenga el ensemble. Cada muestra se selecciona aleatoriamente con reemplazamiento.

La salida del ensemble se obtiene a partir de la combinación de la salida de cada modelo base. Por ejemplo, para clasificación se puede utilizar el voto mayoritario, es decir, la salida del ensemble será la que haya predicho un mayor número de veces en los predictores base. Para regresión, se puede utilizar la media aritmética. No obstante, existen variaciones que tratan de ponderar cada una de las salidas en función de la probabilidad de pertenencia del patrón a cada clase o bien optimizando una combinación lineal de las salidas para regresión.

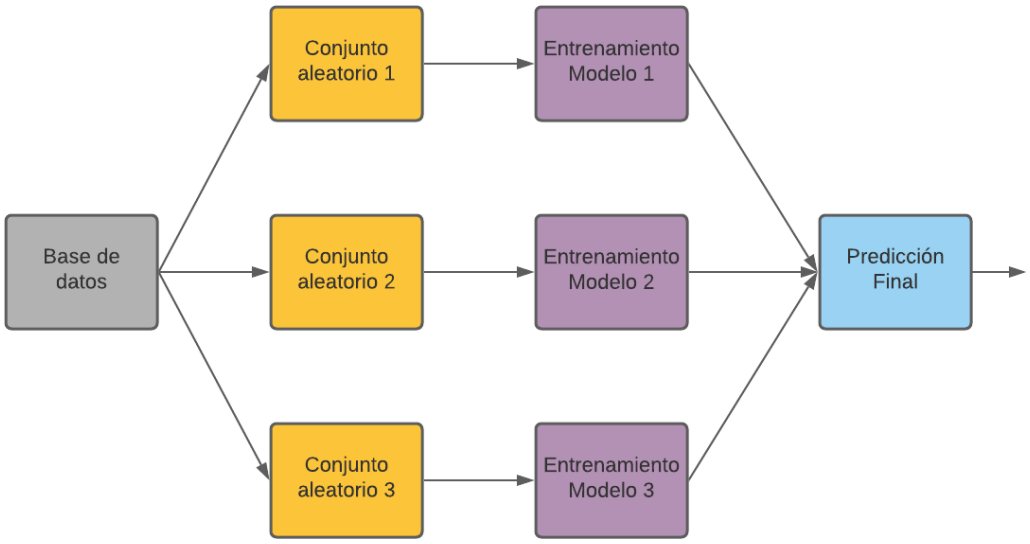

Figura 4.2: *Bagging* con tres predictores base.

El uso de ***Boosting*** consiste en aplicar un aprendizaje secuencial. El algoritmo funciona entrenando el modelo con todo el conjunto de entrenamiento y consiste en crear varios predictores sencillos en secuencia. De esta forma, el segundo modelo ajusta bien lo que el primero no ajustó, el tercero lo que el segundo no pudo ajustar y así sucesivamente. Esto se consigue otorgando más peso a las muestras mal clasificadas y menos peso a las correctamente clasificadas. En problemas de regresión, las predicciones con un mayor error cuadrático medio tendrán más peso para el siguiente modelo.

Un ejemplo de implementación de *Boosting* es el algoritmo ***AdaBoost*** cuya secuencia de pasos es la siguiente:

1. Entrenar un predictor (ajustar sus parámetros).
2. Usar el predictor para identificar los casos incorrectamente predichos.
3. Construir un nuevo predictor que mejore la predicción en los casos incorrectamente predichos del punto 2.
4. Repita los pasos 2 y 3 hasta que se satisface la condición de parada (número de clasificadores base utilizados o patrones correctamente predichos).
5. Asignar un peso a cada predictor y unirlos para obtener un modelo con mejor desempeño.

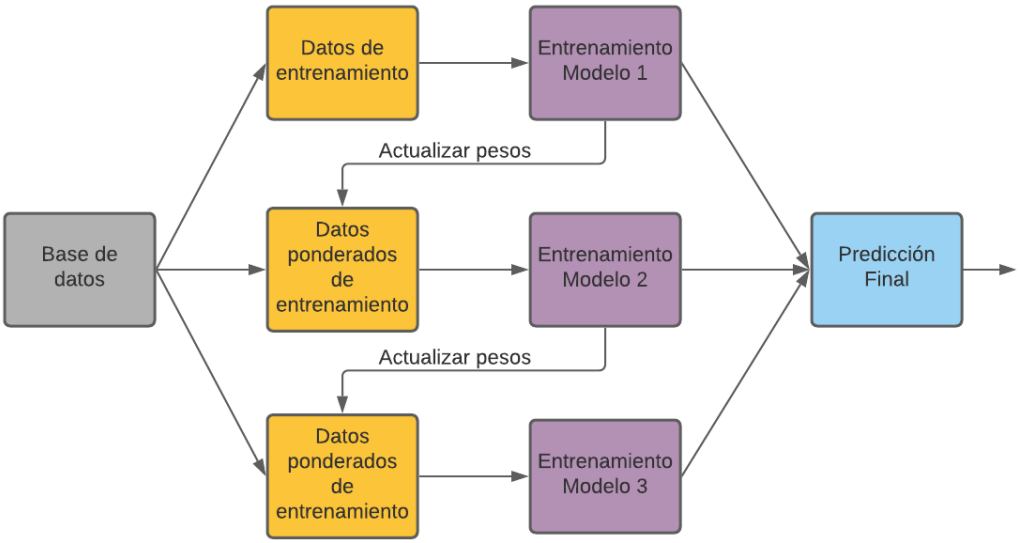

Figura 4.3: *Boosting* con tres predictores base.

4.4. Consideraciones importantes

Podemos destacar:

- Los ensembles unen las predicciones de un conjunto de modelos base.
- Funcionan tanto en regresión como clasificación.
- Debe haber diversidad en cada modelo base.
- Dos de las técnicas de generación de ensembles más utilizadas son *Bagging* y *Boosting*.
- Tanto el tiempo de entrenamiento como el tiempo de *test* son obviamente mayores que en modelos únicos. Dependerá de los requisitos de nuestro problema elegir este tipo de técnicas.

5. PUNTOS CLAVE

Una vez que hemos desarrollado los puntos más importantes de la segunda sesión de los modelos de regresión y clasificación, estaremos capacitados para:

- Ajustar los parámetros de un modelo de red neuronal, realizar predicciones en el conjunto de generalización e interpretar sus resultados.
- Comprender y saber aplicar las máquinas de vectores soporte.
- Entender el concepto de ensemble y los tipos de generación existentes.