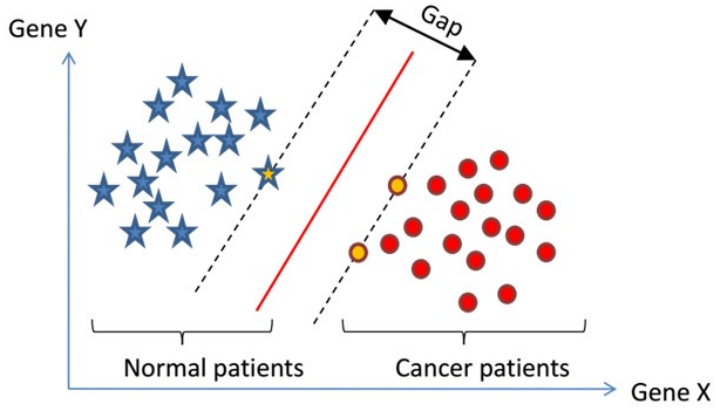

Figura 14. Hiperplano separante de margen máximo.

Se les denomina de esa forma porque son los que conforman o ayudan a delimitar los bordes del margen máximo entre las clases.

En caso de que sea imposible encontrar un hiperplano que linealmente separe las clases, entonces los datos se mapean a un espacio con una dimensionalidad mucho más alta llamado «Espacio de Rasgos» donde trata de encontrarse otro hiperplano que las separe. Este espacio de rasgos se construye por medio de lo que se conoce como «Kernel Trick» o truco de Kernel.

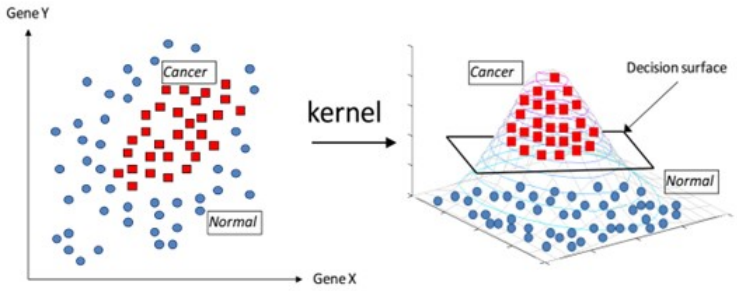

Figura 15. Efecto del mapeo de los datos a un espacio de una dimensión mayor usando el truco del Kernel.

Hiperplanos

La matemática detrás de este método de clasificación es trivial y de alguna manera es preciso que introduzcamos algunos conceptos vitales para entenderlo y el de Hiperplano es uno de ellos.

En un espacio de dos dimensiones el hiperplano es una recta. Pero, un hiperplano se puede definir para un espacio de p dimensiones. La ecuación general del hiperplano para un espacio de p dimensiones es:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + ...\beta_p x_p = 0$$

donde el vector:

Se llama vector normal, y es un vector unitario en el cual la suma de los cuadrados es 1. Este vector apunta a una dirección ortogonal a la superficie del hiperplano (vector de color rojo de la figura 1.).

Si proyectamos cada uno de los puntos sobre el vector normal, los puntos que caen sobre el hiperplano tienen valor 0 al proyectarse. Los puntos por encima del hiperplano tienen un valor positivo, los puntos por debajo del hiperplano un valor negativo y además estos valores son mayores cuanto más lejanos están del hiperplano. Es decir, el valor que se obtiene al proyectar los puntos sobre el vector normal es proporcional a la distancia de los puntos al hiperplano. Esta característica geométrica hace posible el uso de las máquinas de vector de apoyo para buscar los patrones necesarios.

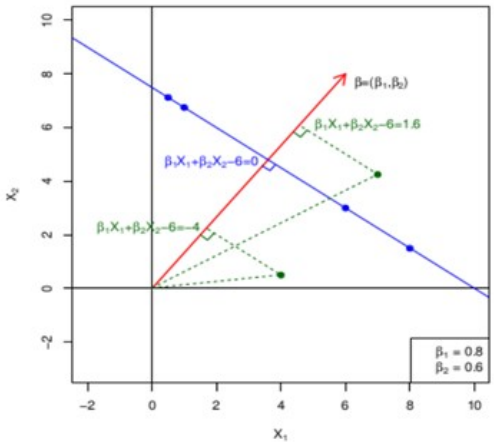

Figura 16. Ejemplo de un hiperplano

La cuestión es que el hiperplano busca separar dos conjuntos de puntos en dos regiones distintas. Por ejemplo, en un problema de clasificación el hiperplano puede separar aquellos puntos que corresponden a personas con un tumor determinado de aquellas personas sanas.

Sin embargo, para un conjunto de puntos determinados existen múltiples hiperplanos posibles que separan los puntos en dos regiones.

Separando por hiperplanos

Dado una serie de puntos en un espacio geométrico que se desea clasificar, se puede establecer que aquellos puntos que al proyectarse sobre el vector normal a un hiperplano determinado sean > 0, correspondan a una clase y aquellos que sean < 0, a otra clase.

Es decir, de forma matemática:

$$f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ...\beta_p x_p$$

$(X) > 0$ establece los puntos en un lado del hiperplano

$(X) < 0$ en el otro

Sin embargo, lo habitual es encontrarse en la situación en la cual existen múltiples hiperplanos posibles, y por tanto es necesario decidir ¿cuál de ellos establecer?

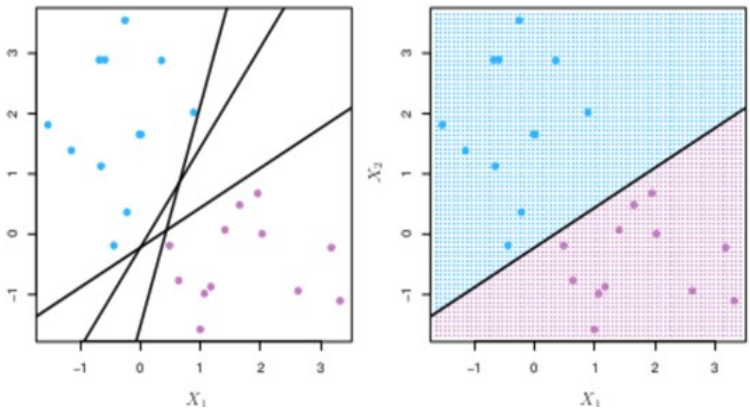

Figura 17. Ejemplo de tres hiperplanos posibles para separar dos conjuntos de clase (izquierda). En el caso de elegir un hiperplano los puntos se clasifican en función de cada una de las regiones sombreadas de rojo y azul.

Clasificador de Margen Máximo

En el caso de un problema de clasificación binaria, de todos los hiperplanos posibles, es necesario buscar aquel que nos proporciona la mayor diferencia entre las dos clases, lo cual se traduce en la mayor distancia entre los puntos que pertenecen a una clase y a otra. La hipótesis que hay detrás de esto, es porque suponemos que este hiperplano será el que tendrá una mayor distancia en el conjunto de test y en las predicciones futuras.

Esta situación puede modelarse como un problema de optimización con restricciones donde es necesario

maximizar el margen y, matemáticamente, se define:

Maximizar M

$$\beta_1,\beta_2,\ldots,\beta_p$$

Sujeto a las siguientes restricciones

$$\sum_{i=1}^{p}\beta_i{}^2 = 1$$

$$y_i\,(\beta_0+\beta_1x_1+\beta_2x_2+\ldots\beta_px_p)\geq M$$

Para todo i=1,…,N

En la siguiente imagen se muestra de forma gráfica el concepto donde la línea continua resaltada es el Margen Máximo, y se muestran dos bandas con líneas discontinúas que contienen la distancia del hiperplano a los primeros puntos de cada una de las clases, el objetivo es maximizar la distancia de estas dos bandas.

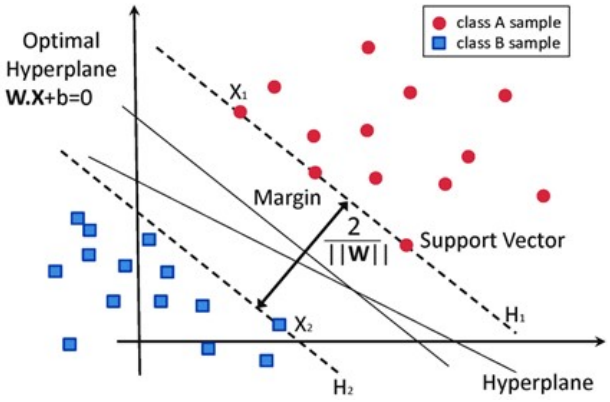

Figura 18. Ejemplo de Hiperplano separante con Margen Máximal.

El principal problema con esta separación óptima del hiperplano es que los datos habitualmente no son linealmente separables con una recta en el caso de un espacio de dos dimensiones, como se observa en la siguiente gráfica, donde es imposible separar los puntos de una clase de los de otra.

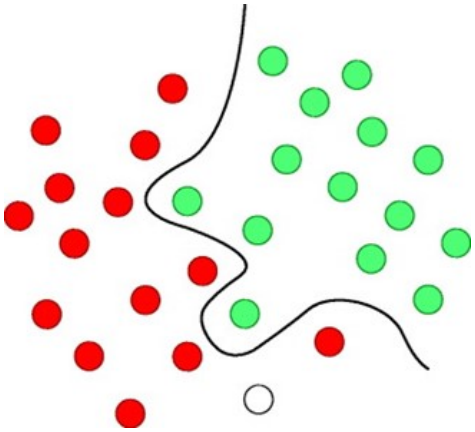

Figura 19. Ejemplo de dos clases que no son linealmente separables.

Esta situación produce una solución pobre para el clasificador de Margen Máximo. Por tanto, el clasificador con vectores de apoyo maximiza un Margen Suave.

Clasificador con Margen Suave

La solución al problema anterior en el cual los puntos no son linealmente separables es maximizar un llamado Margen Suave. Matemáticamente el problema se define de la siguiente forma:

Maximizar M

$$\beta_1,\beta_2,\dots,\beta_p,\zeta_1,\zeta_2,\dots,\zeta_n$$

Sujeto a las siguientes restricciones

$$\sum_{i=1}^{p}{\beta_i}^2=1$$

$$y_i\left(\beta_0+\beta_1x_1+\beta_2x_2+\dots\beta_px_p\right)\geq M\left(i-\zeta_i\right)$$

$$\zeta_i\geq0,\sum_{i=1}^{n}\zeta_i\leq C$$

De forma gráfica, en la siguiente figura se observa:

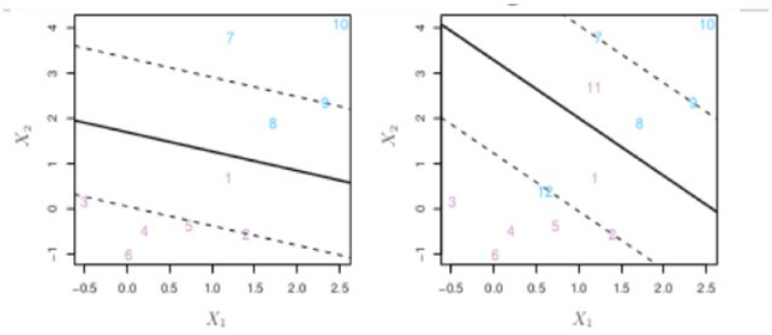

Figura 20. Ajuste de un clasificador con vectores de apoyo a pocos datos. El hiperplano se muestra con una línea sólida y los márgenes con una punteada. La imagen de la derecha muestra el efecto cuando aparecen dos nuevas observaciones (11 y 12) las cuales están en el lado incorrecto del hiperplano y de los márgenes.

Modificando el parámetro C de la ecuación anterior, el cual se conoce como función de coste se puede hacer el margen más grande o pequeño. En concreto un valor de C más grande hace el margen más pequeño y a la inversa un valor C más pequeño hace el margen más grande.

En la siguiente figura se muestra un hiperplano y los márgenes correspondientes en función de diferente valor de C.

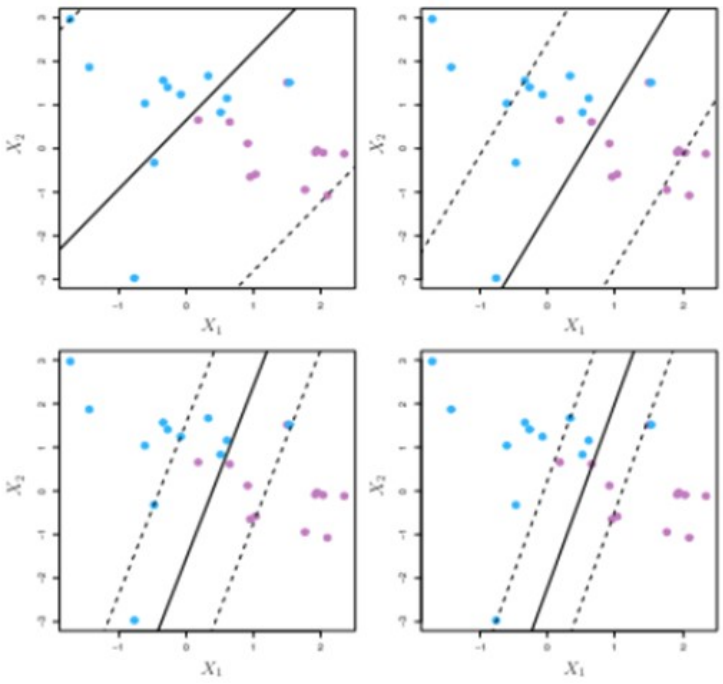

Figura 21. Diferentes fronteras de decisión en los márgenes de una soft-margin en función de diferentes valores de C y utilizando los mismos puntos y el mismo hiperplano. El margen más grande es el de la gráfica superior izquierda y el más pequeño el de la inferior derecha.

Problema con las fronteras lineales

Muchas veces las fronteras lineales siguen sin funcionar, independientemente del valor de coste C que se utilice. Por ejemplo, en la siguiente gráfica se muestra un problema donde la separación necesaria entre los puntos de una clase y la otra es no lineal.

Expansión de variables

Para solucionar este problema donde no es posible realizar una separación entre dos clases, una opción es incrementar el espacio de las variables por medio de transformaciones, es decir pasar de un espacio p-dimensional a un espacio de D dimensiones D > p. Una vez realizada esta transformación, se debería ajustar un clasificador con vectores de apoyo en este nuevo espacio. El objetivo es obtener fronteras de decisión no lineales sobre el espacio original.

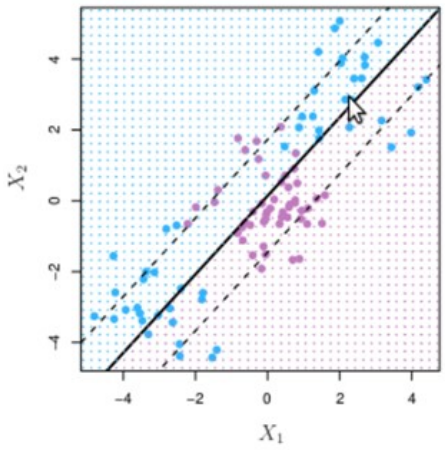

Figura 22. Ejemplo donde no es posible una separación lineal entre dos clases.

Por ejemplo, si tenemos datos de entrada en dos dimensiones es posible utilizar el siguiente espacio de 6 dimensiones: (X_(1,) X_2,X_1^(2,) X_2^2,X_(1,) X_2) siendo por tanto la frontera de decisión:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 {X_1}^2 + \beta_4 {X_2}^2 + \beta_5 X_1 X_2 = 0$$

Esta transformación conlleva fronteras de decisión no-lineales en el espacio original.

Polinomios cúbicos

Utilizando una transformación en forma de polinomios cúbicos se puede pasar de 2 a 9 variables. De esta forma el clasificador en el nuevo espacio soluciona el problema de buscar los patrones en el espacio con menor dimensiones.

En la siguiente gráfica se muestra el resultado de las fronteras de separación obtenidas por medio de una transformación en polinomios cúbicos., resultado del cálculo de la siguiente ecuación:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 {X_1}^2 + \beta_4 {X_2}^2 + \beta_5 X_1 X_2 + \beta_6 {X_1}^2 + \beta_7 {X_2}^3 + \beta_8 X_1 {X_2}^2 + \beta_9 X_1^2 X_2 = 0$$

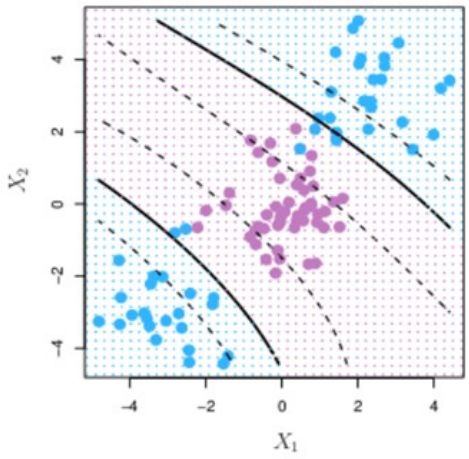

Figura 13. Ejemplo de las fronteras de decisión no lineales obtenidas por medio de una transformación obtenida con polinomios cúbicos.

El truco del Kernel (Kernel trick)

Las expansiones de las variables con polinomios, especialmente aquellos con grandes dimensiones, son computacionalmente costosos. Existe una solución más elegante y controlada de introducir no-linealidad que es utilizado en las máquinas con vectores de apoyo por medio del uso de kernels. El kernel de un operador A denotado por ker es el conjunto de todos los vectores cuya imagen sea el vector nulo:

$$KerA = \backslash\backslash\{\overrightarrow{v} \in V : A\overrightarrow{v} = 0 \backslash\backslash\}$$

Los kernels se apoyan en concepto del producto vectorial de los vectores de apoyo. Se trata de funciones que reciben dos vectores como parámetros. Uno de los kernels más utilizados es el de base radial que asume que el feature space es de altas dimensiones y se define con la siguiente función matemática:

$$Ker\left(x_i,\acute{x_i}\right)=\exp\left(-\gamma\sum_{i=1}^{p}\left(x_{ij}-\acute{x_{ij}}\right)^2\right)$$

$$yi\left(\beta_0+\beta_1x_{i1}+\beta_2x_{i2}+...+\beta_px_{ip}\right)\geq M$$

El uso de kernels permite obtener fronteras de decisión no lineales por medio de transformaciones matemáticas sin necesidad de tener que realizar transformaciones con polinomios. En la siguiente figura se muestran las fronteras de decisión no lineales obtenidas por medio del uso de un kernel de base radial.

Está más que claro que una comprensión completa de los principios e interpretación de los conceptos anteriores requerirá mucho más que una lectura a este material. Por eso sugerimos que para profundizar en este tema consulten: A Gentle Introduction to Support Vector Machines in Biomedicine, de los autores Alexander Statnikov, Douglas Hardin, Isabelle Guyon, Constantin F. Aliferis. Así como Support Vector Machines Succinctly de Alexandre Kowalczyk. Aunque hay mucha información disponible sobre el tema.

Hasta aquí el recorrido por la ideas básicas detrás de las Maquinas con Vectores de Apoyo o SVM. Veamos cómo se usan en la práctica y los detalles de los parámetros que son necesario ajustar.

Como a lo largo de estas lecciones, usaremos la implementación de SVM del módulo scikit-learn de Python, con el fin de explicar el uso práctico de este método para generar modelos de aprendizaje automáticos de clasificación.

Como ya mencionamos al principio de esta sección SVM se puede usar tanto para regresión como para clasificación y clustering. Este módulo, implementa tres versiones de SVM para clasificación:

- SVC (C-SVM)
- NuSVC (v-SVM)
- LinearSVC (SVM con kernel Lineal)

Todos los métodos soportan la clasificación binaria o multiclase. NuSVC es una reparametrización de C-SVC implementada en SVC y por tanto matemáticamente equivalentes, pero se incluye un parámetro v en vez de C.

LienarSVC es una implementación de SVM con kernel lineal.

Veremos los parámetros más importantes para cada uno de las implementaciones y luego un ejemplo.

Parámetros de SVC:

- C: toma valores de punto flotante. El valor por omisión es 1. Es un parámetro de regularización y controla la penalización. El efecto de la regularización en inversamente proporcional a C. Debe ser siempre positivo.
- Kernel: Define el tipo de kernel a usar por SVM y pueden ser los siguientes:
  - Lineal : ⟨x,x ´⟩
  - Polinomial(polynomial): (γ⟨x,x'⟩+r)^ddonde d es especificado por el parámetro degree.
  - Rbf (Función de Base Radial) : exp(-γ‖x-x ´ ‖^2) donde γes representado por el parámetro gamma. Debe ser mayor que cero.

Cuando se usa este tipo de kernel hay que tener en cuenta dos parámetros, C y gamma. El parámetro C, que es común a todos los kernel, controla el compromiso entre la calidad de la clasificación y la simplicidad de la superficie de decisión. Cuando C es bajo la superficie de decisión es suave. Cuando C es alta se busca clasificar de forma correcta todos los ejemplos del conjunto de entrenamiento y puede conducir a overffiting. Mientras que gamma define cuanta influencia puede tener un ejemplo de entrenamiento.

Elegir de forma adecuada el C y gamma es vital para SVM. Lo ideal es usar Validación Cruzada (GridSearchCV) espaciándolos de manera exponencial.

- Sigmoide(sigmoid): $(\gamma\langle x,x'\rangle + r)^{d}$
- degree: Toma valores enteros y define el grado en el kernel polinomial. El valor por omisión es 3. Es ignorado por el resto de los kernels.
- gamma: Toma valores float. Es el coeficiente de los kernels rbf, poly y sigmoid. También se le puede pasar los siguientes valores:
  - scale: si gamma='scale' entonces se usa como valor 1/(n de rasgos*x.var()) como su valor.
  - Si se le pasa gamma='auto' entonces usa 1/(n de rasgos) como valor. Para el resto de los parámetros y sus valores, ver la documentación del método.

En la imagen siguiente mostramos el uso de SVC para clasificar el iris dataset, comparando los resultados contra un clasificador basado en Gradiente descendente estocástico.

En este ejemplo se usa la estrategia Uno contra el resto One-vs-Rest de forma explícita, pero esto métodos lo hacen de manera interna y lo mostramos aquí solo a modo de ejemplo.

```python
import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import SGDClassifier
#*****************************************************************
iris = datasets.load_iris(as_frame=True)

#*****************************************************************
x_train, x_test, y_train, y_test = train_test_split(iris.data,iris.target, test_size = 0.25, random_state = 0)
#*****************************************************************
caler = MinMaxScaler()
X_train = scaler.fit_transform(x_train)
X_test = scaler.transform(x_test)
#*****************************************************************
ovr_clf = OneVsRestClassifier(SVC(kernel='rbf',  gamma='auto'))
ovr_clf.fit(x_train, y_train)
y_pred=ovr_clf.predict(x_test)
print("Datos para SVM sobre el Test Set")
print(classification_report(y_test,y_pred))
#*****************************************************************
sgd_clf = SGDClassifier(max_iter=1000, tol=1e-3, random_state=42)
sgd_clf.fit(x_train, y_train)
y_pred=sgd_clf.predict(x_test)
print("Datos para SGD sobre el Test Set")
print(classification_report(y_test,y_pred))
```

Figura 24. Ejemplo de uso de SVC con Kernel=rbf.

Los resultados son los siguientes:

Datos para SVM sobre el Test Set

```
precision recall f1-score support



0 1.00 1.00 1.00 13

1 1.00 0.94 0.97 16

2 0.90 1.00 0.95 9



accuracy 0.97 38

macro avg 0.97 0.98 0.97 38

weighted avg 0.98 0.97 0.97 38



Datos para SGD sobre el Test Set

precision recall f1-score support



0 1.00 1.00 1.00 13
```

```
1 1.00 0.38 0.55 16

2 0.47 1.00 0.64 9


accuracy 0.74 38

macro avg 0.82 0.79 0.73 38

weighted avg 0.88 0.74 0.72 38
```

El accuracy de SVM es del 97% sobre el conjunto de pruebas, que sin dudas es muy buen resultado.

Con esto terminamos nuestro recorrido por los métodos SVM.

En las dos secciones anteriores hemos visto dos métodos de clasificación que esencialmente usan enfoques muy diferentes para atacar el problema.

El que veremos a continuación es quizás el enfoque clásico, por decirlo de alguna forma y es el de clasificación probabilista.

Clasificación con Naives Bayes

En este tema se va a introducir el clasificador Naive Bayes, el cual está basado en el teorema de Bayes para calcular las probabilidades a posteriori de los eventos a predecir.

Vamos a ver los siguientes puntos:

- En primer lugar, veremos el teorema de Bayes.
- A continuación, se describe la forma de calcular las tablas de probabilidad condicionada.
- Posteriormente se verá por qué es importante asumir independencia condicional en el clasificador Naive Bayes.
- El clasificador Naive Bayes y finalmente cómo se puede utilizar con variables numéricas.

Teorema de Bayes

El Teorema de Bayes es una proposición planteada por el filósofo inglés Thomas Bayes (1702-1761) en el año 1763 en su artículo «An Essay towards solving a Problem in the Doctrine of Chances» publicado en la revista Philosophical Transactions of the Royal Society of London.

Este teorema expresa la probabilidad condicional de un evento aleatorio A dado B en términos de la distribución de probabilidad condicional del evento B dado A y la distribución de probabilidad marginal de solo A.

El teorema de Bayes es bastante relevante porque relaciona la probabilidad de dos eventos A y B utilizando la dependencia condicional de uno de ellos. Es decir, relaciona la probabilidad de que ocurra el evento A y sabemos de antemano que ha ocurrido B, utilizando la probabilidad de que ocurra el evento B sabiendo que ha ocurrido A.

Este teorema permite relacionar, entre otras cosas, síntomas y enfermedades. Por ejemplo, sabiendo la probabilidad de tener dolor de garganta dado que se conoce que se tiene gripe, se puede obtener la probabilidad de tener gripe dado que se tiene dolor de garganta.

El teorema de Bayes relaciona la comprensión de la probabilidad de aspectos causa-efecto dados los eventos dependientes observados. Un evento dependiente es aquel cuyo resultado se ve afectado por el resultado de otro evento o serie de eventos. Los eventos dependientes son la base del modelado predictivo puesto que se busca obtener la probabilidad de que ocurra un suceso teniendo en cuenta la existencia de una serie de eventos dependientes.

En el caso de dos eventos dependientes A y B, el teorema de Bayes describe su relación de la siguiente manera:

$$P(A \vee B) = \frac{P(B \vee A)\,P(A)}{P(B)}$$

Es decir, la probabilidad de A dado que ocurre B, es igual a la probabilidad de que ocurra B dado que ha ocurrido A, multiplicado por la probabilidad de que ocurra A, dividido por la probabilidad de que ocurra B.

Para poner esto en contexto de la clasificación transformemos la llamada regla de Bayes a una que nos permita clasificar un conjunto de casos.

Entonces tenemos:

$$P(C \vee X) = \frac{P(X \vee C)\,P(C)}{P(X)}$$

En lenguaje simple y para meros mortales, esto significa que la probabilidad de que el caso X pertenezca a la clase C, está dada por la probabilidad de X dado la clase C multiplicada por la probabilidad de la clase divida por la probabilidad del predictor X.

Dónde:

- P(C) son las probabilidades a priori de las clases.
- P(X v C) Se conoce como Likelihood la cual es la probabilidad del predictor dada la Calse.
- P(Ai v B) son las probabilidades a posteriori.
- P(B) es la probabilidad a priori del predictor.

La esencia de este método es que toma los datos del dataset, calcula primero las probabilidades a priori, construyendo tablas de frecuencias y luego tablas de likelihood a partir de ellas. Después calcula las probabilidades a posteriori aplicando la regla que acabamos de ver y elige la clase que más alta probabilidad a posteriori arroje como clase para el caso X.

Ese, de manera general, es el algoritmo del método y para ejemplificar tomemos un pequeño dataset que pos su simplicidad se presta para esto. Usaremos el Weather Dataset.

Este dataset permite entrenar generar un modelo que nos prediga si podemos jugar Golf a partir de cuatro variables o rasgos que definen el clima y lo asocian con la posibilidad de que se pueda jugar o no.

Las variables son:

- Outlook: toma los siguientes valores: Rainy, Sunny, Overcast.
- Temp: Toma valores: Hot, Mild, Cool.
- Humidity: Toma los valores: High, Normal.
- Windy: Toma solo los valores: True o False.
- Play Golf: Solo toma valores: Yes, No.

El dataset se muestra en la siguiente imagen:

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

Figura 25. Dataset Weather.

Construyamos, las tablas de frecuencia y las de Likelihood para poder entender cómo se calculan las probabilidades a priori de cada uno de los rasgos y luego como calculamos las de las clases y la posteriori.

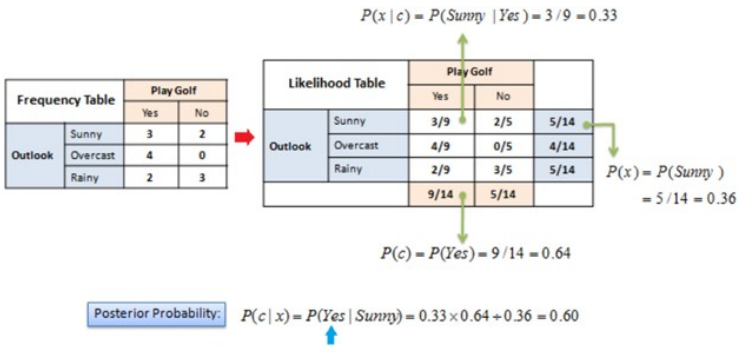

Figura 26. Tabla de frecuencia y Likelihood del rasgo OutLook contra la clase Yes.

Estas tablas se hacen contando las veces que coincide, por ejemplo, el valor sunny del rasgo Outllok con el valor yes de la variable independiente Play Golf. En la imagen anterior se muestran los cálculos para la clase Yes, veamos los de la clase NO.

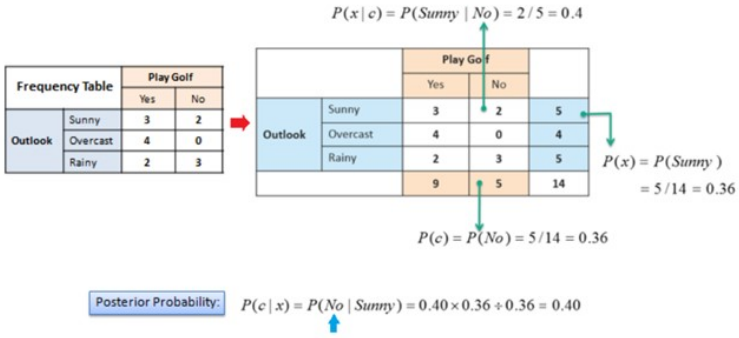

Figura 27. Tabla de frecuencia y Likelihood del rasgo OutLook contra la clase No.

Veamos las tablas del resto de los rasgos.

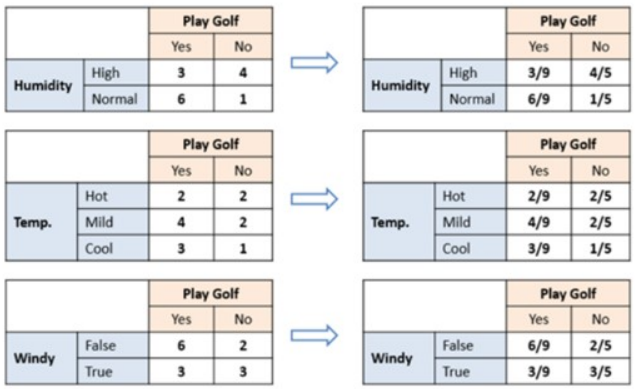

Figura 28. Tabla de frecuencia y Likelihood de los rasgos contra la clase.

Con estos datos a mano ya podemos hacer un ejemplo para clasificar un caso.

Veamos, tenemos el siguiente caso y la pregunta es:

Si, Outlook = rainy, Temp = cool, Humidity= High y Windy=true, ¿podemos jugar Golf?

Si lo formulamos por medio del algoritmo Naives Bayes, tenemos que calcular las probabilidades a posteriori de la siguiente forma:

$$P(yes \vee rainy, cool, high, true) = \frac{P(rainy, cool, high, true \vee yes)\, P(yes)}{P(rainy, cool, high, true)}$$

Calculamos primero los numeradores de la siguiente forma:

$$P(yes \vee rainy, cool, high, true) = P(rainy \vee yes) * P(cool \vee yes) * P(high \vee yes) * P(true \vee yes) * P(yes)$$

Lo iremos haciendo paso a paso:

P(Outlook=rainy|yes)=2/9=0,22 P(Temp=cool|yes)=3/9=0,33

P(Humidity=high|yes)=3/9=0,33 P(Windy=true|yes)=3/9=0,33

P(yes)=9/14=0,64

$$P\left(rainy, cool, high, true \vee yes\right) P\left(yes\right) = 0{,}22 * 0{,}33 * 0{,}33 * 0{,}33 * 0{,}64 = 0{,}005059$$

Ahora hacemos lo mismo para la clase no:

P(Outlook=rainy|no)=3/5=0,6 P(Temp=cool|no)=1/5=0,2

P(Humidity=high|no)=4/5=0,8 P(Windy=true|no)=3/5=0,6

P(no)=5/14=0,35

Por tanto

$$P\left(rainy, cool, high, true \vee no\right) P\left(no\right) = 0{,}6 * 0{,}2 * 0{,}8 * 0{,}6 * 0{,}35 = 0{,}02016$$

Ahora calculamos las probabilidades a posterioris de la siguiente manera

$$P\left(yes \vee rainy, cool, high, true\right) = \frac{0{,}005059}{0{,}005059 + 0{,}02016} = 0{,}2$$

$$P\left(no \vee rainy, cool, high, true\right) = \frac{0{,}02016}{0{,}02016 + 0{,}005059} = 0{,}8$$

Ya calculamos las dos probabilidades y ahora escogemos la que mayor probabilidad nos arroja y es la correspondiente a Play Golf = No.

Y sería consistente, pues, si el día está lluvioso, la temperatura fría, la humedad relativa alta y hace mucho viento, bueno lo lógico es que no se pueda jugar golf.

Si los rasgos fueran numéricos, habría que transformar los rasgos antes de construir las tablas de frecuencias o usar las distribuciones de las variables numéricas para estimar las probabilidades; pero no lo vamos a revisar en esta lección.

De esta forma terminamos de dar un recorrido por las ideas básicas detrás de este tipo de clasificador y veremos ahora algunos detalle de como aplicarlo en la práctica usando el módulo scikit-learn de Python.

Tema 3. Algoritmos de aprendizaje automático supervisado

Naives Bayes en scikit-learn incluye varios algoritmos de aprendizaje supervisado. Cada una con asunciones diferentes. Estos métodos son:

- Naives Bayes Gausiano. (Gaussian Naives Bayes)
- Naives Bayes Multinomial. (Multinomial Naive Bayes)
- Complement Naive Bayes
- Bernoulli Naive Bayes.
- Categorical Naive Bayes.
- Out-of-core naive Bayes model fitting.

Gaussian Naives Bayes

Este método asume que el likelihood de los datos es Gaussiano, es decir, presupone que:

$$P\left(x_i \vee y\right) = \frac{1}{\sqrt{2\pi\sigma_y{}^2}} e^{\frac{-\left(x-\mu_y\right)^2}{2\sigma y^2}}$$

En la siguiente imagen se muestra un ejemplo.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
gnb = GaussianNB()
y_pred = gnb.fit(X_train, y_train).predict(X_test)
print("Número de puntos incorrectamente clasificados del total %d punto : %d"
      % (X_test.shape[0], (y_test != y_pred).sum()))
```

```
Número de puntos incorrectamente clasificados del total 75 punto : 4
```

Figura 29. Ejemplo de uso de Naives Bayes Gaussiano.

Los detalles del resto de los modelos, por favor, revisarlos en la documentación de SciKit-Learn.

Clasificación con arboles de decisión

En esta sección veremos los modelos de clasificación y regresión basados en los árboles de decisión.

- En primer lugar, veremos una introducción a los árboles de decisión.
- A continuación, se hará énfasis en cómo se decide el mejor corte, cubriendo para ello los conceptos de índice Gini y ganancia de información.

- Más adelante, se describe la necesidad de realizar la poda de los árboles.
- Posteriormente, se describen brevemente las diferencias con los árboles utilizados para clasificación.

Arboles de Decisión

Los árboles de decisión son una de las técnicas más populares dentro del campo del aprendizaje automático. Se llevan utilizando durante muchos años en el área de la minería de datos. Se trata de métodos simples y fáciles de interpretar.

Estos modelos dividen o segmentan el espacio de las variables predictoras en una serie de regiones. Una vez creado el árbol de decisión es utilizado para predecir observaciones futuras. Para este propósito se utiliza la moda en el caso de que la variable a predecir sea categórica o bien la media en el caso de que sea numérica.

Como el conjunto de las reglas para separar las variables predictoras se pueden resumir en forma de árbol, a estos métodos se les conoce popularmente con el nombre de árboles de decisión.

Los árboles de decisión dividen el espacio en rectángulos que minimizan el error de la predicción. Debido a que es computacionalmente imposible considerar cualquier posible partición del espacio de entrada se utiliza un enfoque top-down greedy conocido como recursive binary splitting.

En la siguiente imagen se muestra un árbol de decisión sencillo para obtener el logaritmo del salario de los jugadores de béisbol. Este valor se obtiene por medio de las variables de años en la liga (years) y de hits y utilizando unas reglas de decisión en base a los umbrales de estas variables. Así, por tanto, aquellos jugadores con más de 4,5 años de experiencia y más de 117,5 hits estarían en 6,74, mientras que si tuvieran menos de 117,5 hits estarían en 6,0.

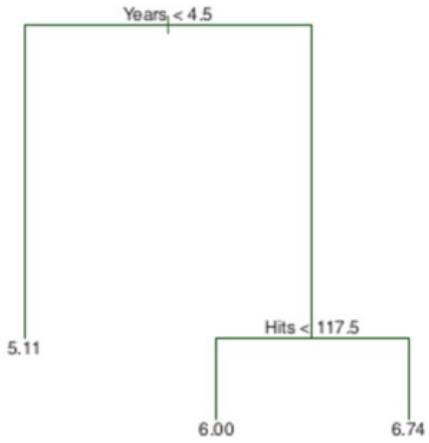

Figura 30. Árbol de Decisión sencillo para estimar el salario de un jugador de béisbol en función de los años y los hits.

El árbol de decisión anterior se obtiene por medio de la representación en un espacio de dos dimensiones de los puntos donde están el eje de años (years) y el de los hits. A cada uno de esos puntos le corresponde un salario determinado y el objetivo es agrupar aquellos puntos que tienen un salario similar utilizando la información de las otras dos variables. (Hits y años de experiencia).

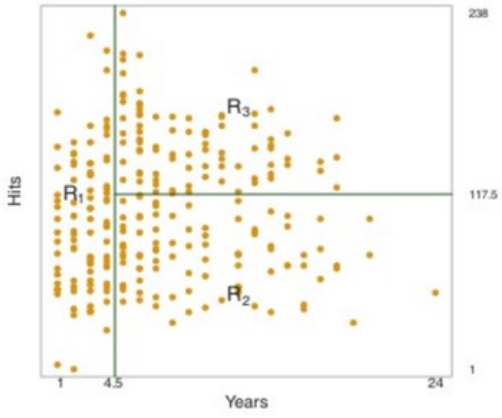

Figura 31. Representación bidimensional de la distribución de Hits y años de los jugadores de la liga.

El árbol de decisión se obtiene por medio de un algoritmo que elige primero aquella variable que es más predictiva de la variable objetivo. A continuación, los ejemplos de entrenamiento se dividen en grupos con distintos valores para las clases de esta primera variable. El algoritmo continúa dividiendo los nodos con la elección de la mejor variable en cada iteración hasta que se alcance el criterio de parada. En cada iteración el algoritmo elige aquella variable que mejor predice la variable objetivo, por tanto, se trata de un

algoritmo de tipo greedy. El criterio de parada puede venir dado por algunas de estas situaciones:

- Todos, o casi todos, los ejemplos del nodo son de la misma clase.
- No existen variables para distinguir entre los ejemplos.
- El árbol ha alcanzado un tamaño predefinido.

Existen numerosas implementaciones de los árboles de decisión disponibles en el mercado, no obstante, las más famosas son:

- C.5.0 desarrollado por Ross Quinlan.
- C 4.5.
- ID3 (Iterative Dichotomiser 3).
- J48, una alternativa basada en Java open source del C.4.5.

De todos ellos el algoritmo C.5.0 está considerado como el estándar en la industria.

Best split: entropía, Gini index, ganancia de información

Cada una de las divisiones del árbol da como resultado dos grupos. Es decir, los datos resultantes de la división por la variable elegida y con el punto de corte determinado están en uno de los dos grupos. Cuando los segmentos de una división contienen valores de una única clase se considera que es una división pura.

A la hora de identificar los criterios para realizar el split o corte existen varias métricas de pureza. Muchos algoritmos, por ejemplo, el C.5.0 utilizan el concepto de entropía.

En este caso un valor de 0 indica que la muestra es completamente homogénea, mientras que un valor de 1 indica desorden completo.

La entropía se define con la siguiente fórmula matemática:

$$Entropia\,(S) = \sum_{i=1}^{n} -p_i \log_2 (p_i)$$

Por ejemplo, si se ha realizado una partición utilizando una variable y umbral determinado y los datos de dos clases se dividen en un 60 % en un lado de la rama y en un 40 % en otro lado de la rama, tendríamos el siguiente valor de entropía:

$$Entropia(s) = -0{,}60 * \log_2(0{,}60) - 0{,}40 * \log_2(0{,}40) = 0{,}9709506$$

Utilizando la medida de pureza el algoritmo tiene que decidir con que variable hacer el corte. Una opción es utilizar la entropía para calcular el cambio resultante de hacer el corte en esa variable.

A continuación, se calcula la ganancia de información (information gain o IG) que es la diferencia entre la entropía en el segmento antes de hacer el corte (S1) y la partición resultante de hacer el corte (S2), es decir:

Donde IG es la Ganancia de Información.

Después de un corte los datos se pueden dividir en más de una partición, por tanto, se considera la entropía a lo largo de las N particiones, ponderando la entropía de cada partición con el número de instancias de esa partición:

$$InfoGain(f) = entropia(s_1) - entropia(s_2)$$

Poda de los árboles

El proceso anterior de generación de cortes recursivos denominado top-down greedy puede generar buenas predicciones en el conjunto de entrenamiento, pero también sufre el problema del sobre ajuste.

Esto se debe a que el árbol resultante puede llegar a ser muy complejo y ajustarse demasiado bien a los datos de entrenamiento. Un árbol mucho más pequeño puede dar lugar a una menor varianza y mejor interpretación con el coste de un pequeño sesgo.

La estrategia para generar estos árboles más pequeños y con una menor varianza suele ser generar un árbol muy grande y después podarlo para obtener un sub-árbol (post-prunning).

Ahora bien, la cuestión es, ¿cómo seleccionamos el sub-árbol? Una buena solución es utilizar aquel sub-árbol que proporcione un menor error de test en validación cruzada (cross validaiton) o bien en el conjunto de validación.

Por ejemplo, en la gráfica 21 se muestra la obtención de un árbol sin podar.

Árboles para clasificación

Los árboles se pueden utilizar para realizar una predicción numérica en el caso de regresión o bien para realizar una clasificación. Los árboles de clasificación (clasification trees) son similares a los regression trees pero predicen la respuesta de una variable categórica.

En regression trees la predicción corresponde a la media de las observaciones de ese nodo terminal. En clasification trees se trata de la clase que tiene un mayor número de ocurrencias en ese nodo terminal.

En el caso de clasificación, el criterio que se suele utilizar para realizar los cortes o splits es el Gini index, que se define:

$$G = \sum_{k=1}^{k} \widehat{pmk}\left(1 - \widehat{pmk}\right)$$

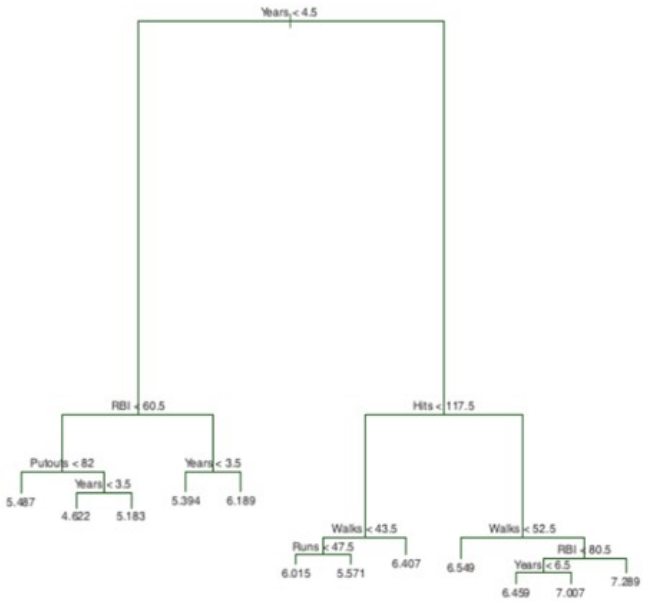

Figura 32. Ejemplo de un árbol sin podar.

O bien la entropía cruzada, la cual se define como:

$$D = -\sum_{k=1}^{k} \widehat{pmk} \log \widehat{pmk}$$

Árboles vs. Modelos lineales

Una regresión lineal asume un modelo que tiene la siguiente forma:

$$f(x) = \beta_0 \sum_{j=1}^{p} X_j \beta_j$$

Mientras que un árbol de regresión, asume un modelo:

$$f(x) = \beta_0 \sum_{j=1}^{p} X_j \beta_j$$

¿Qué modelo es mejor? Depende de lo que se esté modelando. Si las relaciones entre las variables de entrada y la variable respuesta se pueden aproximar por un modelo lineal, una regresión lineal funciona bien y supera a un árbol de regresión. Esto se debe a que un árbol de regresión no es capaz de modelar esta estructura.

Sin embargo, si la naturaleza del problema es no lineal y con relaciones complejas entre las variables, los árboles funcionan mejor.

Hasta aquí hemos hecho un breve recorrido por los conceptos básicos de los árboles de decisión y como en todos los casos ahora examinaremos como se emplean en la práctica, usando el módulo Scikit-Learn de Python.

En scikit-lean la clase que permite implementar un árbol de decisión para clasificación multiclase es DecisionTreeClassifier.

Los parámetros que este permite son los siguientes:

- criterion: Se le pueden pasar dos valores “gini” o “entropia”. El valor por omisión es gini. Estos son las medidas de la calidad de split.
- Splitter: Los valores que permite son “best” y “random”. El valor por omisión es “best”.
- max_depth: Acepta valores enteros y el valor por omisión es “None”. Este parámetro define la máxima profundidad del árbol. Si se define None o no se le pasa otro valor, entonces los nodos serán expandidos hasta que todas las hojas sean puras o hasta que todas las hojas contengan menos que min_samples_split muestras.

Para el resto de los parámetros de esta clase por favor revise la documentación.

A continuación le mostramos un ejemplo de cómo usar DecisionTreeClassifier, para clasificar el iris Dataset.

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.datasets import load_iris
#**************************************************
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
Dtree = DecisionTreeClassifier()
y_pred= Dtree.fit(X_train,y_train).predict(X_test)
print("Datos para DT sobre el Test Set")
print(classification_report(y_test,y_pred))
plot_tree(Dtree)
```

```
Datos para DT sobre el Test Set
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        21
           1       0.94      0.97      0.95        30
           2       0.96      0.92      0.94        24

    accuracy                           0.96        75
   macro avg       0.96      0.96      0.96        75
weighted avg       0.96      0.96      0.96        75
```

Figura 33. Ejemplo de uso de DecisionTreeClassifier, con el iris Dataset.

Clasificación con random forests

Random forests es un método basado en ensembles de árboles de decisión que fue inventado por Leo Breiman y Adele Cutler en el año 2001.

En este tema se describe como los modelos de random forests explotan la diversidad y son capaces de generalizar y predecir mejor que un árbol de decisión. Esta diversidad se consigue por medio de entrenar modelos con el método bagging y de realizar una selección de variables aleatoria.

A continuación, se discute la interpretación del out-of-bag error. Posteriormente nos enfocamos en determinar el número de árboles y en evaluar la importancia de las variables.

Explotando la diversidad: bagging y selección de variables

Uno de los inconvenientes principales de los árboles de decisión es su baja capacidad predictiva. Este inconveniente se puede solventar por medio de utilizar un ensemble o combinación de modelos de árboles de decisión. El modelo random forests es en esencia un ensemble de árboles de decisión.

Los ensembles de árboles se pueden combinar utilizando los métodos de bagging o boosting. Los modelos de random forests se basan en combinar los árboles por medio de los métodos de bagging.

En el caso de problemas de clasificación en cada observación de test se almacena la clase predicha por cada uno de los B árboles y la clase final se obtiene por medio del voto de la mayoría. Es decir, la predicción global es la clase que más veces ocurre a lo largo de las B predicciones. Por otro lado, en el caso de los problemas de regresión la predicción global es la media de las predicciones de cada uno de los árboles.

El modelo de random forests combina los principios de bagging con selección de variables aleatorias para añadir diversidad a los árboles de decisión. Una vez es generado el ensemble de árboles (forest) el modelo utiliza el mecanismo de votación o la media para generar las predicciones.

Se trata de un modelo que combina versatilidad y potencia en un enfoque. A la hora de construir cada uno de los árboles se utiliza una porción pequeña y aleatoria de las variables de entrada disponibles, por lo general, este número se define como √p siendo p el número de variables de entrada.

Al ser un modelo que genera cada árbol con un subconjunto de los registros de entrada y una selección aleatoria de las variables, puede trabajar con conjuntos de datos bastantes grandes y no se encuentra afectado por los problemas de curse of dimensionality. Además, debido a que los árboles son modelos con mucho ruido y muy inestables se pueden beneficiar de forma notoria de ser promediados.

En cada uno de los cortes de cada uno de los árboles se elige una serie de variables candidatas de entre todas las posibles. El valor por defecto, $\sqrt{p}$ variables aleatorias de entre todas las posibles, suele dar buenos resultados.

Este modelo se basa en la utilización de un gran número de árboles. La razón de utilizar un gran número de árboles es para que cada variable, de entre todas las posibles, tenga la oportunidad de aparecer en varios modelos.

La ventaja principal de los modelos de random forests frente a un modelo bagged de árboles es debido a que las variables de cada split de los árboles se obtienen por medio de seleccionar un subconjunto de todas las variables de forma aleatoria. De esta forma, se consigue decorrelacionar los árboles generados. Además, se reduce la varianza porque el resultado se obtiene a partir de calcular el promedio de los árboles y se controla el sobreajuste.

Por tanto, cada árbol de decisión se construye utilizando bootstrapped training samples y en cada split se utiliza una variable a partir de una selección aleatoria de m predictores del total p. Finalmente, los distintos árboles se combinan utilizando el voto de la mayoría para clasificación y la media para regresión.

En los random forests debido a la selección de variables aleatorias sobre el conjunto de entrada completo, el sesgo de los árboles se suele incrementar (con respecto al sesgo un único árbol de decisión); no obstante, debido a promediar los árboles la varianza se reduce y, por lo general, en una proporción mayor que el incremento del sesgo dando lugar a un mejor modelo por lo general. En muchos problemas el rendimiento de los random forests es similar a los modelos boosting a pesar de ser más sencillos de entrenar. Como consecuencia de este hecho, los random forests se han convertido en una técnica muy

popular.

Interpretación de out-of-bag error

Una característica importante de los modelos random forests es el error out-of-bag. El out-of-bag error es una forma sencilla de estimar el error de test en un modelo bagged. Se puede demostrar que de media cada árbol construido con un modelo bagged utiliza 2/3 de las observaciones del conjunto de entrenamiento. El 1/3 restante de las observaciones de entrenamiento no se utilizan para generar el árbol bagged y son llamadas out-of-bag.

Para cada observación se puede obtener la respuesta de los bagged tres, donde esta observación era out-of-bag. Esto proporciona alrededor de B/3 predicciones para cada una de las observaciones.

Si el valor de B (número de árboles) es grande esta estimación es equivalente a un modelo de tipo leave-one-out de validación cruzada. Este modelo de leave-one-out es la forma más rigurosa y extrema de evaluar un modelo de aprendizaje automático.

Un modelo random forests se puede entrenar de forma secuencial, donde no es necesario utilizar validación cruzada, pues las muestras out-of-bag proporcionan una estimación similar. Para ello, es necesario observar la evolución del out-of-bag error y una vez que este error se haya estabilizado el algoritmo puede parar el entrenamiento.

Si el valor de B (número de árboles) es grande esta estimación es equivalente a un modelo de tipo leave-one-out de validación cruzada. Este modelo de leave-one-out es la forma más rigurosa y extrema de evaluar un modelo de aprendizaje automático.

Un modelo random forests se puede entrenar de forma secuencial, donde no es necesario utilizar validación cruzada, pues las muestras out-of-bag proporcionan una estimación similar. Para ello, es necesario observar la evolución del out-of-bag error y una vez que este error se haya estabilizado el algoritmo puede parar el entrenamiento.

Evolución del número de árboles e importancia de variables

Los modelos basados en random forests principalmente tienen dos parámetros para optimizar: el número de árboles y el número de variables que se evalúan en cada tirada. Existen otros parámetros como la profundidad de los árboles, pero en la práctica no presentan muchos cambios.

En la siguiente gráfica se observa la evolución del error en el conjunto de test en función del número de árboles y del número de variables que se prueban en cada split. Se observa que el valor de $\sqrt{p}$

proporciona un error menor que los otros valores.

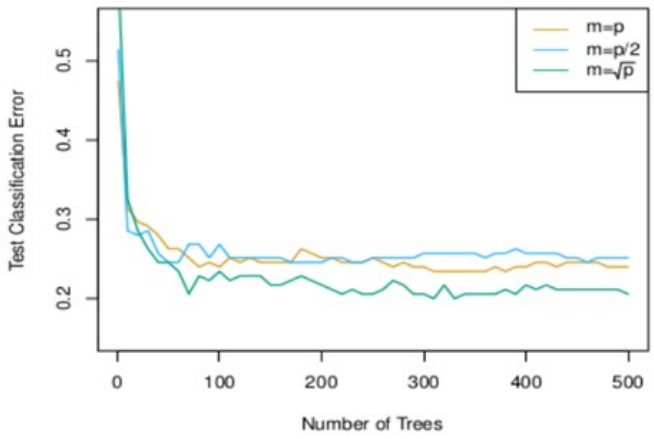

Figura 34. Resultado de tres random forest, en un problema de clasificación con 15 clases y 500 predictores.

En principio se puede pensar que cuanto más grande sea el número de árboles mejor. Sin embargo, por lo general existe un punto óptimo donde a pesar de añadir más árboles el error no se reduce de forma significativa.

Una de las ventajas de los random forests es que pueden generar buenos resultados sin necesidad de muchos ajustes manuales. Además, permiten construir de forma sencilla gráficos de importancia de variables.

En cada decisión de corte de cada árbol la mejora en el criterio de corte es la medida de importancia que se atribuye al corte de esa variable y se agrega para todos los árboles del forest para cada variable.

Random forest también utiliza las muestras fuera del bag (OOB) para construir una medida de importancia de variables que mide la capacidad predictiva de cada una de las variables.

En SciKit-Learn la clase RandomForestClassifier del módulo ensemble, es la implementación de este método.

Los parámetros más importantes son:

- n_estimators: Espera un entero. Determina la cantidad de árboles en el bosque. El valor por omisión es 100.
- criterion: Puede ser establecido a «Gini» o «entropía». El valor por defecto es «Gini».

- max_depth: Espera valores enteros y el valor por defecto es None. Establece la profundidad máxima del árbol. Si el valor pasado es None, o no se le establece valor, los nodos son expandidos hasta que todas las hojas sean puras o hasta que todas las hojas contengan una cantidad menor de muestras que las especificadas en min_samples_split.

- min_samples_split: Espera un valor entero. El valor por defecto es 2. Especifica el número mínimo de casos o muestras requeridos para dividir un nodo interno.

Si este número contiene un valor entero, entonces min_samples_split es el mínimo. Si por el contrario este contiene un valor float, entonces min_samples_split es una fracción y ceil(min_samples_split*n_samples) son los números mínimos de muestras.

Este parámetro y el anterior es importante que se tengan en cuenta para optimizar el uso de la memoria, cuando esto es importante.

Ahora veremos un ejemplo de cómo se usa en Python.

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
#*****************************************************************
#Lee el dataser de diabetes que está en formato csv y retorna un DataFrame
df = pd.read_csv("diabetes.csv")
dataset = df
columns_incorrect = ["Glucose","BloodPressure","SkinThickness","Insulin","BMI"]
#Sustituye todos los ceros en las variables
for column in columns_incorrect:
    #Separa el dataset en los casos que se sabén que son diabéticos de los que no lo son
    df1 = dataset.loc[dataset['Outcome'] == 1]
    df2 = dataset.loc[dataset['Outcome'] == 0]
    #Reemplaza todos los ceros de las variables con las medianas no con las medias
    df1 = df1.replace({column:0}, np.median(df1[column]))
    df2 = df2.replace({column:0}, np.median(df2[column]))
    dataframe = [df1, df2]
    dataset = pd.concat(dataframe)
#En esta parte tratamos los Outliers
#Ahora utilizaremos un escaler para normalizar los calores de las variables
Y = dataset.Outcome
x = dataset.drop("Outcome", axis = 1)
columns = x.columns
scaler = StandardScaler()
X = scaler.fit_transform(x)
data_x = pd.DataFrame(X, columns = columns)
#Ahora separamos el dataset en dos partes, un conjunto de entrenamiento y otro de prueba
#El conjunto de prueba es de 15% del total.
x_train, x_test, y_train, y_test = train_test_split(data_x, Y, test_size = 0.15, random_state = 45)
#Como es evidente que los datos son invalanceados, se usa la técnica SMOTE para valanceārlos
#Esta técnica genera datos a partir de los existentes, pero solo en el conjunto de entrenamiento
smt = SMOTE()
x_train, y_train = smt.fit_resample(x_train, y_train)
#*****************************************************************************************************
# Ahora aplicamos Random Forest
Forest = RandomForestClassifier()
Forest.fit(x_train, y_train)
y_pred = Forest.predict(x_test)
print("Accurace de RF", classification_report(y_test,y_pred))
```

Figura 35. Ejemplo de aplicación de Random Forest al dataset Pima Indian Diabetes.

Con este ejemplo terminamos lo referente a Random Forest e iniciamos con aquellos métodos se forman por medio de combinaciones de otros métodos.

Tema 3. Algoritmos de aprendizaje automático supervisado

Clasificación con ensembles de clasificadores: bootstrapping, bagging y boosting

En esta sección se van a estudiar los métodos de ensemble. Estos métodos combinan las predicciones de varios estimadores base, que han sido construidos con un algoritmo de aprendizaje automático, para mejorar la generalización/robustez de un único modelo.

En primer lugar, veremos lo correspondiente a bootstraping. A continuación, se describe el método bagging, el cual se apoya en la técnica de bootstrapping para ensamblar modelos.

Finalmente, veremos el método de boosting, uno de los métodos más populares para llevar a cabo la combinación de modelos débiles para construir un modelo más robusto y con mayor precisión.

Por lo general los métodos de ensemble o combinación de clasificadores se pueden dividir en dos grandes familias:

- **Averaging methods:** los cuales se basan en construir varios estimadores de forma independiente y luego promediar las predicciones. En media, el estimador combinado se suele comportar mejor que cualquier estimador individual porque la varianza se reduce. Los métodos de bagging funcionan de esta forma.
- **Boosting methods:** los estimadores base se construyen de forma secuencial con el objetivo de reducir el sesgo del estimador combinado. La motivación es combinar varios modelos débiles para producir un modelo combinado potente.

Técnica de bootstrapping

Bootstrapping es una técnica estadística que consiste en cualquier test o métrica sobre muestreo aleatorio con reemplazo. Bootstrapping permite asignar medidas de precisión definidas en términos de sesgo, varianza, intervalos de confianza o cualquier otra métrica sobre estimaciones muestrales. Esta técnica permite la estimación de la distribución de la muestra de cualquier estadístico utilizando métodos de muestreo. Generalmente, se suele clasificar como una clase de método de muestreo.

Bootstrapping es la práctica de estimar las propiedades de un estimador (así como su varianza) por medio de medir las propiedades al muestrear una distribución aproximada. Una elección habitual para aproximar la distribución es utilizar la función de distribución empírica de los datos observados. En el caso de que las observaciones se puedan asumir que se generan a partir de una población independiente e idénticamente distribuidas, pueden ser obtenidas por medio de muestreos con reemplazo del conjunto de datos observados.