

Algoritmos de clasificación en minería de datos

David Charle

Contents

1	Introducción a la minería de datos	1
2	Definiciones	2
3	Problema de clasificación	2
3.1	Clasificación binaria	2
3.2	Clasificación multiclase	3
3.3	Clasificación multietiqueta	4
4	Transformación de problemas	4
4.1	Multiclase	5
4.2	Multietiqueta	5
5	Algoritmos de clasificación	6
5.1	K-nearest neighbors (kNN)	6
5.2	Árboles de decisión	8
5.3	Máquina de vectores de soporte (SVM)	8
5.4	RAkEL	10
5.5	HOMER	10

1 Introducción a la minería de datos

A día de hoy, se recopilan cantidades ingentes de datos en cada vez menos tiempo. Fotografías tomadas en teléfonos y publicadas, datos de estaciones meteorológicas, valores en bolsa, son simples ejemplos de la información que se produce constantemente y a la que queremos buscar un significado. Y mientras que algunos datos se podrán interpretar manualmente, para estudiar muchos otros conjuntos de datos necesitaremos de la ayuda de ordenadores y de la Ciencia de datos.

Para extraer conocimiento de los datos se realiza el proceso de KDD, o *Knowledge Discovery in Databases*, que incluye generalmente las siguientes fases¹:

- Selección de un conjunto de datos a analizar
- Preprocesamiento: limpieza de datos, eliminación de ruidos...
- Transformación: reducir número de variables, transformación del problema...
- Minería: búsqueda de patrones de interés
- Interpretación y visualización

¹Fayyad, Usama; Piatetsky-Shapiro, Gregory; Smyth, Padhraic - *From Data Mining to Knowledge Discovery in Databases*

Nos centraremos en la fase del análisis de los datos en sí para buscar significados, conocida como Minería de datos. Los patrones que busquemos servirán para predecir el comportamiento de nuevas observaciones y explicar las relaciones entre las ya conocidas, además de para visualizar la información que se extrae y resumirla en medidas estadísticas.

La Minería de datos abarca distintos problemas, generalmente divididos en aprendizaje no supervisado (*clustering*, detección de anomalías, reglas de asociación) y aprendizaje supervisado (clasificación, regresión). Aquí estudiaremos el problema de clasificación, que consiste en aprender de un conjunto de datos ya clasificados para ser capaz de clasificar nuevos datos.

2 Definiciones

- **Característica/Atributo:** En general, se trata de un conjunto, no vacío, de posibles valores. Por ejemplo, $F_1 = \{Intel, AMD\}$, $F_2 = \mathbb{R}_0^+$.
- **Espacio de características:** Es el producto cartesiano de las características. $F = F_1 \times F_2 \times \dots \times F_n$ para F_1, \dots, F_n características.
- **Dataset:** Un subconjunto finito $D \subset F$ espacio de características.
- **Instancia:** Cada $X \in D$ dataset.
- **Clasificador:** Una función capaz de predecir los atributos ausentes en nuevos datos. $c : F_1 \times F_2 \times \dots \times F_m \rightarrow F_{m+1} \times \dots \times F_n$
- **Algoritmo de clasificación:** Un procedimiento que toma un dataset y genera un clasificador *entrenado* para el mismo. Esto significa que el algoritmo encontrará posibles relaciones entre los atributos de los datos, de forma que cuando tengamos un dato nuevo con características ausentes, se le relacionará con unas instancias u otras a partir del resto de atributos.

3 Problema de clasificación

El problema al que nos enfrentamos consiste en, teniendo un dataset al que llamaremos *de entrenamiento*, generar un clasificador que sea capaz de predecir, con la mayor precisión posible, una o más características de cualquier nueva instancia incompleta.

Atendiendo al número de características que estarán ausentes en los nuevos datos, y a sus posibles valores, distinguiremos tres tipos de clasificación:

- **Binaria:** Implica clasificar en 2 clases (generalmente 0 o 1, *Verdadero o falso*), utilizando una característica que tome solo dos valores.
- **Multiclase:** Ahora habrá más de dos clases, pero cada instancia pertenecerá a una y solo una de ellas, por lo que se usará una característica que contenga tantos valores como clases.
- **Multietiqueta:** En este caso cada instancia puede asociarse a más de una etiqueta, por tanto se usarán tantos atributos como etiquetas, cada uno de ellos conteniendo dos valores.

Adicionalmente, existe una generalización del problema de clasificación que reside en la clasificación **multidimensional**. En este tipo de clasificación, las características que se habrán de predecir serán de cualquier tipo, no necesariamente de dos valores.

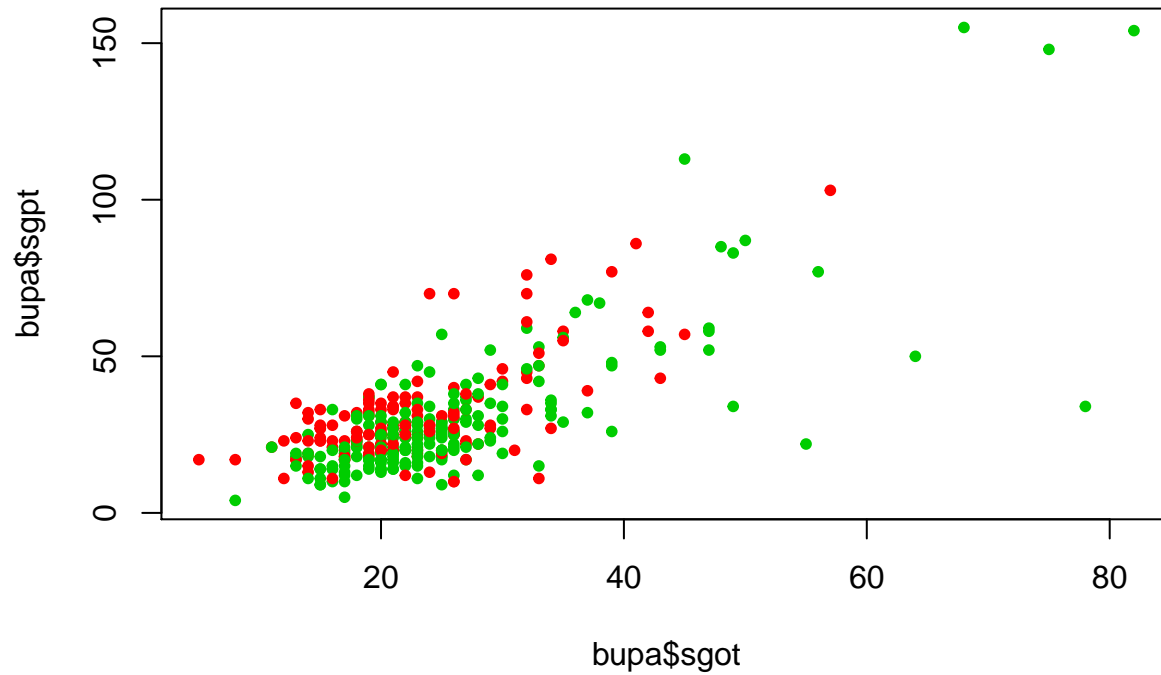
3.1 Clasificación binaria

En clasificación binaria solo usaremos una característica para contener información de las clases, por lo que $m = n - 1$, y como solo tendremos dos clases, tomaremos $F_n = \{0, 1\}$.

El siguiente es un ejemplo de gráfico generado a partir de las instancias del dataset *bupa*², que tiene 7

²UCI Machine Learning Repository - [Liver Disorders](#)

atributos en total (se han escogido dos para los ejes y el atributo de clase está representado por el color):

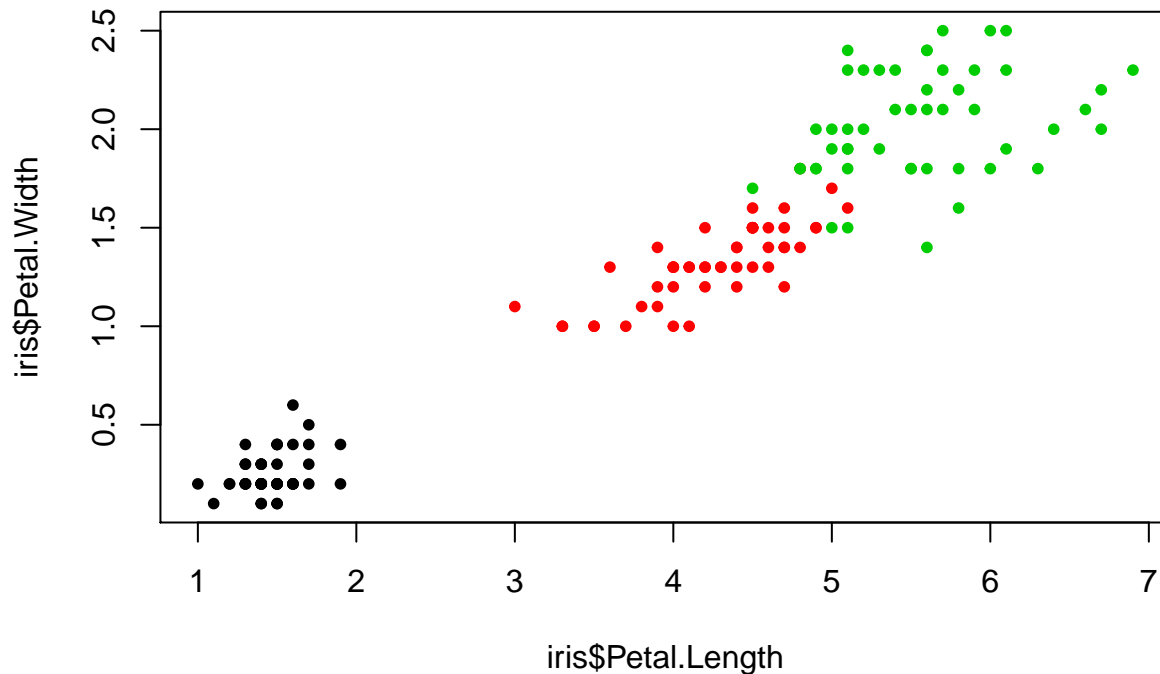


3.2 Clasificación multiclase

La clasificación multiclase añade la complejidad de que, en lugar de dos únicos valores para la característica que agrupa a las instancias, tendremos 3 o más. Esto significa que aunque m se mantenga igual a $n - 1$, tendremos $F_n = \{1, 2, 3 \dots\}$.

La nube de puntos siguiente representa los datos del conocido dataset `iris`³, que cuenta con 5 atributos, el último de los cuales se utiliza para separarlos en 3 clases. De nuevo, se representan dos de los atributos y el color diferencia las clases:

³UCI Machine Learning Repository - [Iris](#)



3.3 Clasificación multietiqueta

Una etiqueta se diferencia de una clase en que no es exclusiva, es decir, que una instancia puede pertenecer a varias etiquetas a la vez. Esto implica que cada etiqueta puede estar activada o desactivada en todas las instancias, y por tanto necesitaremos tantas características como etiquetas: $m = n - l \leq n$ para l etiquetas. Cada atributo correspondiente a una etiqueta tendrá dos valores: $F_{m+1} = \dots = F_n = \{0, 1\}$.

A continuación se muestran las etiquetas del dataset `emotions` obtenido del repositorio MULAN⁴. Este dataset, aun siendo de los más pequeños entre los multietiqueta, cuenta con 78 características, 6 de las cuales son etiquetas.

##	index	count
## amazed-suprised	73	173
## happy-pleased	74	166
## relaxing-calm	75	264
## quiet-still	76	148
## sad-lonely	77	168
## angry-aggressive	78	189

Uno de los problemas que presenta la clasificación multietiqueta es que suelen existir varias decenas de etiquetas, o incluso cientos (el dataset de MULAN con más etiquetas tiene 3993), aumentando significativamente la cantidad de información que un clasificador debe averiguar para cada dato nuevo. Además, es probable que se dé la situación en que algunas etiquetas estén relacionadas entre sí. Un buen algoritmo de clasificación deberá tener esto en cuenta, y no tratar cada etiqueta por separado.

4 Transformación de problemas

Las primeras estrategias a las que podemos recurrir cuando tratamos un problema de clasificación multiclase o multietiqueta son las de transformación del mismo a problemas de clasificación binaria.

⁴MULAN - [Multilabel datasets](#)

Supondremos que disponemos de un algoritmo de clasificación binaria que entrenará clasificadores para varios conjuntos de datos.

4.1 Multiclase

4.1.1 One vs. All

Consiste en entrenar un clasificador para cada clase, de forma que sea capaz de distinguir cuándo un dato nuevo tiene o no esa clase, y aportar un valor de confianza (probabilidad) del resultado. Para decidir la clase que llevará un nuevo dato, se le aplican todos los clasificadores y se queda con la clase que tenga el mayor valor de confianza:

$$\begin{aligned}\phi_i : F &\rightarrow F_1 \times \dots \times F_{n-1} \times F_n^{(i)} \\ \phi_i(x_1, x_2, \dots, x_n) &= (x_1, x_2, \dots, x_{n-1}, \delta_{i, x_n})\end{aligned}$$

donde δ_{i, x_n} es la Delta de Kronecker para i y x_n . Obtenemos clasificadores c_i para $\phi_i(D)$, $\forall i \in F_n$, y generamos el clasificador completo:

$$\begin{aligned}c : F_1 \times \dots \times F_{n-1} &\rightarrow F_n \\ c(x) &= \operatorname{argmax}_{i \in F_n} \operatorname{conf}(c_i(x))\end{aligned}$$

donde $\operatorname{conf}(c_i(x))$ es la confianza del clasificador c_i de que la instancia x pertenezca a la i -ésima clase.

Una desventaja de esta estrategia es que al darle a cada clasificador todas las instancias, se desequilibran fácilmente las clases, ya que se le informará de una de las clases y el resto de las instancias aparecerán como de una única clase. Cuando esto suceda, el clasificador podría dar siempre valores muy pobres de confianza.

4.1.2 One vs. One

Para cada pareja de clases, se entrenará un clasificador que aprenda a distinguirlas. Para ello, se restringirá el dataset en cada caso a instancias de alguna de las dos clases que se vayan a usar. En total se entrenarán $\frac{l(l-1)}{2}$ clasificadores (para l clases). De esta forma, cuando tengamos que decidir la clase de un nuevo dato, cada uno de los clasificadores dará una de dos clases, y la clase más votada será la que se le asigne al dato.

Uno de los problemas de esta estrategia es la posible ambigüedad cuando haya dos clases con el mismo número de votos y haya que decidir entre ellas. Además, cada clasificador que se entrene no dispondrá de todas las instancias, sino de la restricción a las dos clases elegidas, por lo que no será capaz de obtener toda la información.

4.2 Multietiqueta

4.2.1 Binary Relevance

La idea de Binary Relevance es tratar cada etiqueta como un par de clases, y separar un problema de l etiquetas en l problemas de clasificación binaria, generando tantos clasificadores como etiquetas tengamos:

$$\begin{aligned}c_i : F_1 \times \dots \times F_m &\rightarrow F_{m+i} \quad \forall i = 1, \dots, l \\ c = (c_1, c_2, \dots, c_l) : F_1 \times \dots \times F_m &\rightarrow F_{m+1} \times \dots \times F_n\end{aligned}$$

El problema principal de esta estrategia es que no tiene en cuenta las relaciones entre las distintas etiquetas, por lo que el aprendizaje obtenido es menor.

4.2.2 Label Powerset

Label Powerset pretende solventar el problema de Binary Relevance, estudiando en vez de cada etiqueta por separado, sus posibles combinaciones:

$$\begin{aligned}\forall L \in F_{m+1} \times \cdots \times F_n, \quad F_L &= \{0, 1\} \\ \phi_L : F &\rightarrow F_1 \times \cdots \times F_m \times F_L \\ \phi_L((x, L)) &= (x, 1); \quad \phi_L((x, L')) = (x, 0) \forall L' \neq L\end{aligned}$$

Esta transformación de los datos nos deja una única característica, F_L , para averiguar, se trata de un problema binario. Con el dataset restringido a cada uno de esos espacios de características, entrenaremos un clasificador (en total, 2^l):

$$\begin{aligned}c_L : F_1 \times \cdots \times F_m &\rightarrow F_L \\ c : F_1 \times \cdots \times F_m &\rightarrow F_{m+1} \times \cdots \times F_n \\ c(x) &= \arg \max_{L \in F_{m+1} \times \cdots \times F_n} \text{conf}(c_L(x))\end{aligned}$$

donde, de nuevo, $\text{conf}(c_L(x))$ será la confianza que dé el clasificador correspondiente de que la instancia x tenga la combinación L de etiquetas.

5 Algoritmos de clasificación

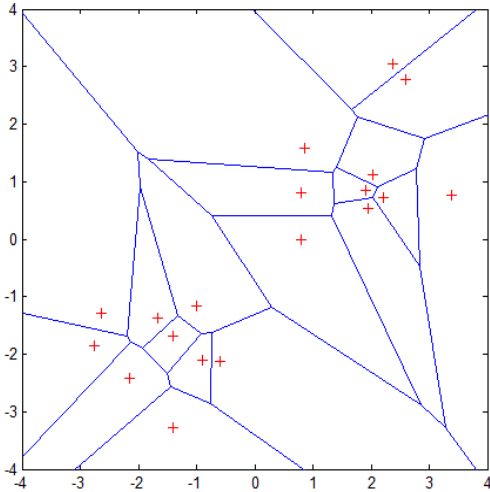
5.1 K-nearest neighbors (kNN)

Introducido en 1951⁵, el método de los k vecinos más cercanos consiste en hallar los primeros k puntos de $F_1 \times \cdots \times F_k$ a menor distancia (para cierta distancia d , generalmente la euclídea) del nuevo dato a clasificar. Una vez encontrados estos puntos, se le otorga al nuevo dato la clase de la mayoría (esquema de votación).

En el caso $k = 1$, expresar matemáticamente el algoritmo es muy sencillo; para un dato a clasificar X , la clase será la de la instancia más cercana a él por la distancia d :

$$NN(X) = \arg \min_{Y \in D} d(X, (Y_1, \dots, Y_{n-1})); \quad c(X) = (NN(X))_n \quad \forall X \in F_1 \times \cdots \times F_{n-1}$$

Además, se puede visualizar fácilmente el método mediante un diagrama de Voronoi⁶, que divide el plano en regiones que identifican al punto más cercano:



⁵Fix, E.; Hodges, J.L. - *Discriminatory analysis, nonparametric discrimination: Consistency properties*

⁶Diagrama de Voronoi obtenido de [Scholarpedia](#) (CC BY-NC-SA)

En el siguiente ejemplo⁷ utilizamos el algoritmo kNN para clasificar datos del dataset `iris`.

Nota: En el ejemplo se utiliza *validación cruzada* para comprobar la precisión del clasificador generado. La validación cruzada consiste en dividir el dataset en varias partes del mismo tamaño, y utilizar todas menos una como dataset para el algoritmo, y la restante como conjunto de test, es decir, para comprobación de la capacidad de clasificación del algoritmo. El procedimiento se repite utilizando una parte distinta como conjunto de test en cada iteración.

```
# Cargamos dataset
iris <- read.arff(system.file("arff", "iris.arff", package = "RWeka"))
# Generamos clasificador kNN con k = 10
classifier <- IBk(class ~., data = iris, control = Weka_control(K = 10))
# Evaluamos con validación cruzada
evaluate_Weka_classifier(classifier, numFolds = 5)
```

```
## === 5 Fold Cross Validation ===
##
## === Summary ===
##
## Correctly Classified Instances      145           96.6667 %
## Incorrectly Classified Instances    5             3.3333 %
## Kappa statistic                     0.95
## Mean absolute error                 0.0418
## Root mean squared error             0.1254
## Relative absolute error             9.3984 %
## Root relative squared error         26.5963 %
## Coverage of cases (0.95 level)     100           %
## Mean rel. region size (0.95 level)  41.7778 %
## Total Number of Instances          150
##
## === Confusion Matrix ===
##
##   a  b  c  <-- classified as
## 50  0  0 | a = Iris-setosa
##  0 49  1 | b = Iris-versicolor
##  0  4 46 | c = Iris-virginica
```

5.1.1 Tipificación

Generalmente en el espacio de características habrá atributos en distintas unidades de medida. Esto puede provocar un sesgo importante en el cálculo de la distancia entre los puntos, ya que algunos de los atributos tendrán más peso que otros. Para contrarrestar este efecto se tipifican las características, es decir, a cada valor de la característica se le resta la media de todos los valores del dataset y se divide el resultado entre la desviación típica.

5.1.2 Distance-weighted kNN⁸

Con el objetivo de afinar más la elección de la clase para nuevos datos, se le aplican pesos a los puntos más cercanos según la distancia a la que estén. Por tanto, la clase del punto más cercano tendrá más importancia en la elección final que la del k -ésimo punto más cercano.

⁷Wikibooks - [Data Mining Algorithms in R/Classification/kNN](#)

⁸Dudani, Sahibsingh A. - *The Distance-Weighted k-Nearest-Neighbor Rule*

5.1.3 ML-kNN⁹

ML-kNN es una adaptación del algoritmo kNN para problemas de clasificación multietiqueta. En este método, una vez localizados los k puntos más cercanos a una instancia de test Y , construimos un vector de cuentas como sigue. Si $kNN(Y) = \{X^{(1)}, X^{(2)}, \dots, X^{(k)}\}$ son los vecinos más cercanos a Y , definimos la cuenta:

$$C(Y) = \sum_{X \in kNN(Y)} (X_{m+1}, X_{m+2}, \dots, X_n)$$

De esta forma obtenemos un vector de l elementos para los cuales, cuanto mayor sea el valor, mayor es la confianza de que la instancia T tenga las etiquetas asociadas. Para discriminar cuáles de las etiquetas se asignarán y cuáles no, ML-kNN utiliza una estimación *Maximum a posteriori*.

5.1.4 Problemas en alta dimensionalidad

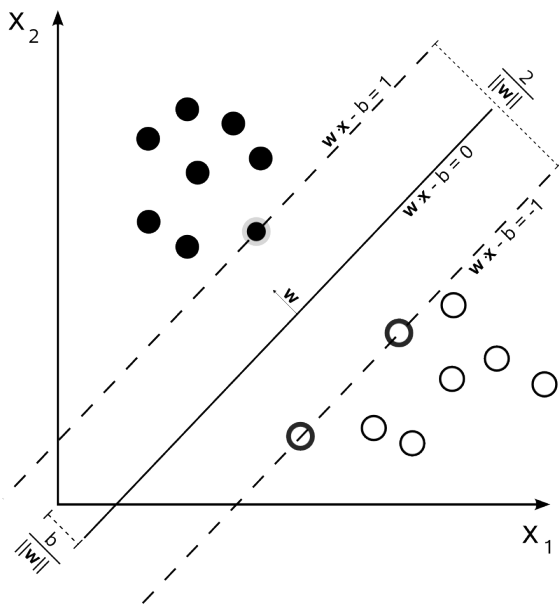
Se puede comprobar que, bajo ciertas condiciones, al aumentar el número de dimensiones, dada una instancia, la diferencia entre la distancia de la instancia más cercana y la distancia a la más lejana tiende a cero¹⁰.

5.2 Árboles de decisión

Binaria+multiclase

5.3 Máquina de vectores de soporte (SVM)

Las máquinas de vectores de soporte tienen su origen en la teoría de aprendizaje estadístico. En principio solo son capaces de tratar el problema de clasificación binaria. La idea básica de estos algoritmos es tratar de separar las instancias de las dos clases mediante un hiperplano de forma que la distancia que las separe sea la máxima posible. Los nuevos datos para los que haya que predecir la clase serán asignados a la que corresponda a la mitad del espacio (determinada por el hiperplano) a la que pertenezcan.



⁹Min-Ling Zhang, Zhi-Hua Zhou - *ML-kNN: A Lazy Learning Approach to Multi-Label Learning*

¹⁰Beyer, K.; Goldstein, J; Ramakrishnan, R; Shaft. U - *When Is "Nearest Neighbor" Meaningful?*

Por tanto, para un algoritmo de este tipo, cada instancia de un dataset representa un punto en un espacio de $n - 1$ dimensiones (si tenemos n atributos), y se pretende encontrar un hiperplano afín de la forma

$$H = \{x \in F_1 \times \cdots \times F_{n-1} : \langle w, x \rangle - b = 0\}$$

donde w sea un vector ortogonal al hiperplano y b un desplazamiento, tal que tenga el máximo margen de distancia a cualquier punto. Este margen se define como

$$\delta^* = \min_{X=(x,y) \in D} \{d(x, H)\} = \min_{X=(x,y) \in D} \|\overrightarrow{x - \pi_H(X)}\|$$

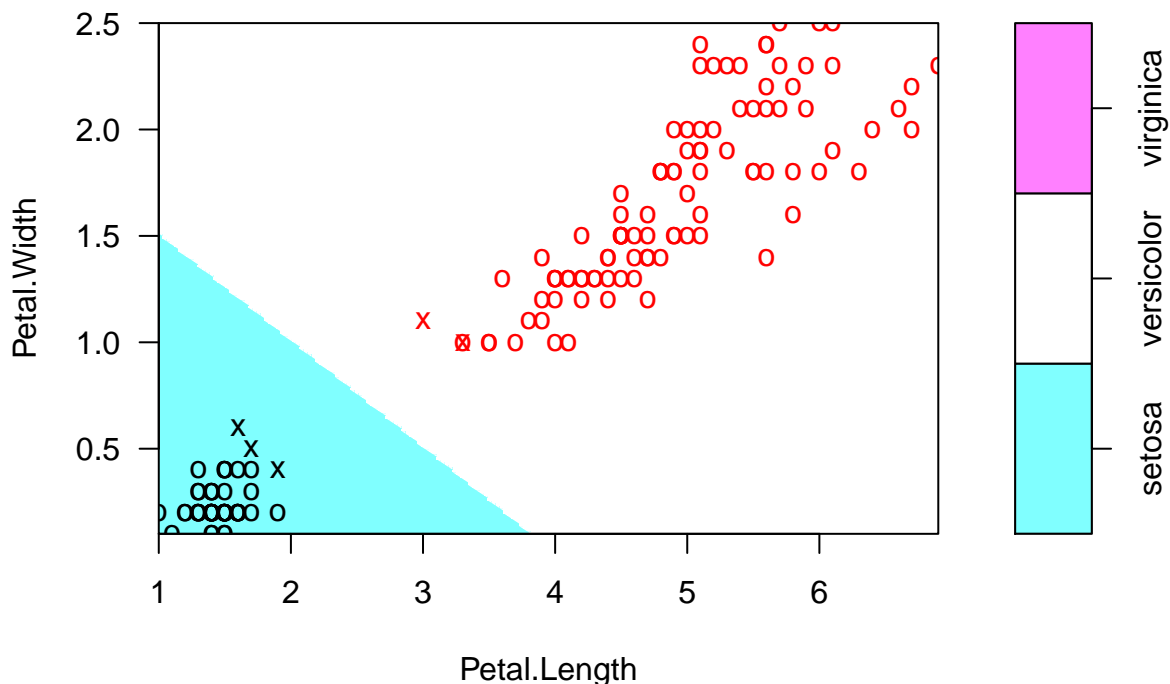
Los puntos que realizan este mínimo de distancia se denominan *vectores de soporte*.

En lugar de tratar de optimizar el margen mediante las distancias a todos los puntos, se puede utilizar el desplazamiento para medir la separación entre los hiperplanos $\{x : \langle w, x \rangle - b = 1\}, \{x : \langle w, x \rangle - b = -1\}$ (ver figura superior), de forma que solo tenemos que tratar de maximizar $\frac{1}{\|w\|}$, o equivalentemente, minimizar $\|w\|$, con las restricciones de separación de instancias de distintas clases.

En el siguiente ejemplo utilizamos la función `svm` del paquete `e1071` de R para generar un modelo SVM lineal que separe una clase del dataset `iris` de las demás, atendiendo a dos de las características:

```
# Cargamos dataset integrado en R
data(iris)
# Nos quedamos con 2 características
iris2 <- data.frame(Petal.Width = iris$Petal.Width,
                    Petal.Length = iris$Petal.Length,
                    Species = iris$Species)
# Eliminamos una de las clases
iris2$Species[iris2$Species == "virginica"] <- "versicolor"
# Generamos un modelo de máquina de vectores de soporte
model <- svm(Species~., data = iris2, kernel = "linear")
# Mostramos un gráfico del modelo
plot(model, iris2, grid = 200)
```

SVM classification plot



Los puntos marcados como \mathbf{x} son los vectores de soporte utilizados para el modelo.

5.3.1 Funciones *kernel*¹¹

El algoritmo básico para SVM tiene la restricción de que genera únicamente modelos lineales para separar los datos. Para evitarla, se pueden llevar los puntos del dataset a un espacio de más dimensiones, donde puedan ser linealmente separados, y utilizar la SVM en ese espacio. Una vez encontrado el hiperplano que los separa en ese espacio, se puede volver a transformar al espacio original, convirtiendolo en una superficie no lineal.

Para llevar esto a cabo se utilizan funciones *kernel*, que son capaces de hallar los productos escalares en distintos espacios, sin necesidad de crearlos y transformar todo el dataset. Algunos ejemplos de funciones *kernel* son:

- Kernel polinomial de grado q : $K(x_i, x_j) = (\langle x_i, x_j \rangle + 1)^q$
- Kernel gaussiano: $K(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$
- Kernel sigmoide de desplazamiento b : $K(x_i, x_j) = \tanh(a \langle x_i, x_j \rangle + b)$

5.4 RAKEL

Multietiqueta

5.5 HOMER

Multietiqueta



Esta obra se distribuye bajo una licencia [Creative Commons Atribución-CompartirIgual 4.0](#).

¹¹Wikibooks - [Data Mining Algorithms In R/Classification/SVM](#)