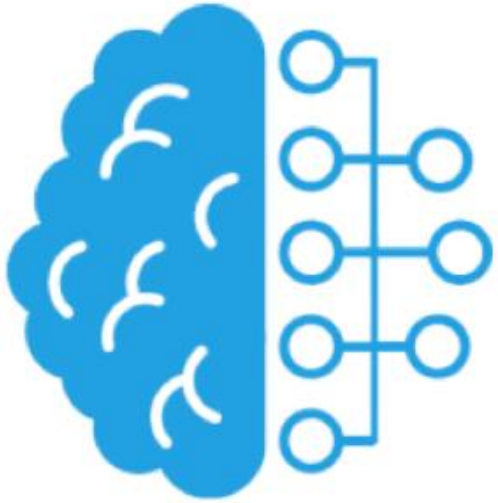
Aprendizaje de Máquina
Clase 4

Claudio Delrieux – DIEC - UNS
cad@uns.edu.ar



Cuántos Modelos de Clasificadores?

Maximum Likelihood
Linear Discriminant Analysis (y Fisher LDA)
Clasificador Bayesiano
K-NN
Regresión Logística
Árboles de decisión (y Random Forests)
Support Vector Machines
Perceptrones
etc. etc.

Clasificación Bayesiana

Definamos un costo de decisión ρ_{ij} $1 \leq i, j \leq n$ asociado con la decisión de asignar a la clase ω_j un patrón x que pertenece a la clase ω_i.

Notar que

ρ_{ii} costo de una decisión correcta, en general se define como 0

ρ_{ij} es en general diferente de ρ_{ji}

$\rho_{ij} \geq 0$

De modo que podemos concluir que el costo total será minimizado si

$$\sum_{i=1}^{m} \rho_{ij} P(\omega_i|\mathrm{x}) \leq \sum_{i=1}^{m} \rho_{ik} P(\omega_i|\mathrm{x}) \quad \forall\, k \neq j$$

Hemos llegado por lo tanto a la llamada *Regla de Decisión de Mínimo Costo de Bayes*, que establece

$$\text{Asignar } \mathrm{x} \rightarrow \omega_j \iff \sum_{i=1}^{m} \rho_{ij} p(\mathrm{x}|\omega_i) P(\omega_i) = \min_{1 \leq k \leq m} \sum_{i=1}^{m} \rho_{ik} p(\mathrm{x}|\omega_i) P(\omega_i)$$

donde en en la última ecuación hemos usado la identidad de Bayes: $P(\omega_i|\mathrm{x})p(\mathrm{x}) = p(\mathrm{x}|\omega_i)P(\omega_i)$.

Clasificación Bayesiana

Observar que el teorema de Bayes permite estimar la probabilidad a-posteriori a partir de la probabilidad a-priori, de las probabilidades condicionales (pdf o likelihood), y de la distribución marginal de los patrones *p(*x*) (o evidencia)*:

$$P(w_i|\mathbf{x}) = \frac{P(\mathbf{x}|w_i)P(w_i)}{P(\mathbf{x})} \qquad \text{a-posteriori} = \frac{\text{a-priori} \times \text{probabilidades}}{\text{evidencia}}$$

El clasificador Bayesiano *ingenuo* (naïve Bayes) asume que todos los eventos **x** y sus probabilidades condicionales y conjuntas son independientes, por lo que se puede computar el modelo más arriba simplemente con productorias.

Clasificación Bayesiana

Como los costos son constantes, pueden asimilarse a la productoria. Finalmente, la *evidencia* (el denominador, que es de difícil estimación), tiene el mismo efecto en todas las clases, por lo que puede eliminarse por lo que finalmente el modelo se conoce como *elegir el máximo a-posteriori* (MAP).

Si atributos son cuantitativos, se evalúa su distribución Gaussiana, con lo cual es posible determinar las probabilidades condicionales (lo que los hace sensibles a datos atípicos). Si es nominal, puede representarse en términos de frecuencias relativas.

Regresión Logística

Pese a su nombre, es un método de clasificación. La regresión es respecto a la probabilidad de que un patrón pertenezca a una determinada clase.

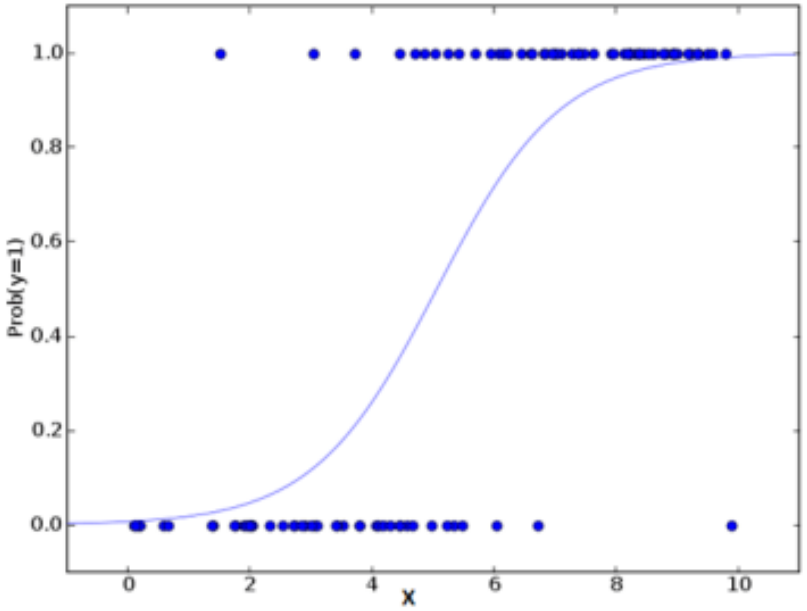
Al igual que con el caso paramétrico, se elige la clase más probable dada la observación, pero la probabilidad no se calcula en base a parámetros estadísticos sino con una función de regresión.

La regresión lineal es inadecuada para probabilidades, dado que éstas no tienen sentido o significado para valores fuera del intervalo [0, 1]. Por ello se usa la denominada función logística.

Regresión Logística

La función logística se basa en las «chances» (*odds*) de x (el cociente entre la probabilidad de que x ocurra sobre la probabilidad de que no). Para simetrizar se aplica el logaritmo de las chances (*log-odds*).

$$\ell = \log \frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$



Regresión Logística

A diferencia del clasificador Bayesiano, en la regresión logística el vector de atributos es cuantitativo (continuo). Si tenemos variables nominales es necesario cuantificarlas.

Regresión logística y Bayesiano son los clasificadores más sencillos y económicos en producción, y por eso son la línea de base respecto de los demás clasificadores. Se prefiere Bayesiano cuando la mayoría de los atributos es nominal (o nominalizable razonablemente) y regresión logísitica cuando la mayoría de los atributos es cuantitativa.

Árboles de Decisión

La inducción de árboles de decisión es una de las aplicaciones más antiguas y exitosas del ML. Un árbol de decisión toma una determinada situación como entrada, y produce una “decisión” como salida, luego de realizar un grupo de tests.

Cada nodo no hoja del árbol tiene asociado un atributo, mientras que cada hoja tiene asociado un resultado. Cada arco tiene asociado un posible valor del atributo del nodo del cual parte.

A partir de un conjunto de ejemplos (valores de los atributos, y decisión correcta esperada) se puede “inducir” un árbol de decisión.

Árboles de Decisión

Encontrar el árbol de decisión más pequeño es un problema NP, por lo que se busca aplicar alguna heurística que permita encontrar árboles suficientemente pequeños en tiempos razonables. Una estrategia es la construcción top-down del árbol, encontrando para el nodo raíz el atributo «mejor» (en algún sentido).

Para ello, se particiona el conjunto de ejemplos en tantos grupos como posibles valores tenga ese atributo, y se construyen sub-árboles con cada partición, asignando cada uno de ellos a un hijo de la raíz.

Luego, aplicar recursivamente el mismo criterio sobre los nodos hijo, hasta llegar a una situación donde la decisión queda cerrada (nodos hoja).

¿Cuántas preguntas se requieren para adivinar en qué elemento estoy pensando en un conjunto de ejemplos S cuyo tamaño es $|S|$?

Cada pregunta si/no elimina como máximo $1/2$ de los elementos. Podríamos entonces definir como **información** a la cantidad de preguntas requeridas para determinar un elemento en un conjunto.

$$I(S) = \log_2 |S|. \tag{1}$$

Supongamos por un momento que nuestro conjunto de ejemplos son o bien positivos o bien negativos, $S = Y \cup N$, Y y N disjuntos. Entonces, cuántas preguntas necesito para determinar un elemento x?

$$P(x \in Y) \log_2 |Y| + P(x \in N) \log_2 |N|,$$

o, en forma equivalente,

$$I(Y) + I(N) = \frac{|Y|}{|Y+N|} \log_2 |Y| + \frac{|N|}{|Y+N|} \log_2 |N|. \quad (2)$$

¿Cuántas preguntas debo hacer para determinar un elemento x en S una vez que sé que el elemento está en Y o N?

Debería ser (1) - (2), lo cual se traduce en

$$I(Y,N) = \log_2 |Y+N| - \frac{|Y|}{|Y+N|} \log_2 |Y| - \frac{|N|}{|Y+N|} \log_2 |N|,$$

lo que es equivalente a

$$I(y,n) = -y \log_2 y - n \log_2 n, \tag{3}$$

donde $y = \frac{|Y|}{|Y+N|}$ y $n = \frac{|N|}{|Y+N|}$

$I(y, n)$ mide el **contenido informacional** o **entropía** en bits (es decir, la cantidad de preguntas por si o por no que se necesitan) asociada con un conjunto S de ejemplos que consisten en un subconjunto Y de casos positivos y otro subconjunto disjunto N de casos negativos.

$0 \leq I(y, n) \leq 1$, donde 0 significa que no hay información (completo desorden), y 1 significa máxima información.

Ejemplo 1: Supongamos que $|Y| = |N|$. En ese caso $y = n = \frac{1}{2}$. Entonces

$$I(\tfrac{1}{2}, \tfrac{1}{2}) = -\tfrac{1}{2}\log_2 \tfrac{1}{2} - \tfrac{1}{2}\log_2 \tfrac{1}{2} = 1.$$

Ejemplo 2: $|Y| = |S|$. En ese caso

$$I(1, 0) = -1\log_2 1 - 0\log_2 0 = 0.$$

La ec. (3) puede generalizarse a

$$I = -\sum_{c \in C}(-P(c)\log_2 P(c)),$$

donde C el conjunto de posibles resultados. Por ejemplo, la entropía de tirar un dado es

$$I = -\sum_{i \in 1.,6}(-P(i)\log_2 P(i)) = 2{,}58,$$

Volviendo a nuestro conjunto de ejemplos S, luego de preguntar el valor de un atributo A, el conjunto se particiona en v subconjuntos, de acuerdo a los valores que asumió A en cada uno. La información remanente es ahora

$$Rem(A) = \sum_{i=1}^{v} \frac{y_i + n_i}{y + n} I(\frac{y_i}{y_i + n_i}, \frac{n_i}{y_i + n_i}),$$

lo cual puede generalizarse para S con más de dos valores (y, n):

$$Rem(A) = - \sum_{v \in A} -P(v) \sum_{c \in S} P(c|v) \log_2 P(c|v).$$

La ganancia de información que se obtiene al utilizar un atributo, es la diferencia entre la información remanente, menos la información en el nodo antes de determinar del valor de A.

Árboles de Decisión

Finalmente, el algoritmo para la inducción de árboles de decisión es:

```
Construir árbol de decisión T para el ejemplo S:

1. Si todos los ejemplos en S pertenecen a la misma clase C,
   entonces retornar un nodo hoja etiquetado con C.

2. Caso contrario
   2.1 Elegir el atributo más informativo A,
   2.2 Particionar S en S1, ..., Sn según los valores de A
   2.3 Construír árboles T1, ... Tn para S1, ... Sn
   2.4 El árbol final T tiene raíz A y subárboles T1, ... Tn
```

Árboles de Decisión

Los AD son probablemente el producto más transparente e intuitivo (caja blanca) dado que son autoexplicativos.

Se parecen a la manera en la que los humanos realizan toma de decisiones.

Requieren poca preparación previa de los datos y son robustos frente a atípicos.

Robustos frente a variables co-dependientes.

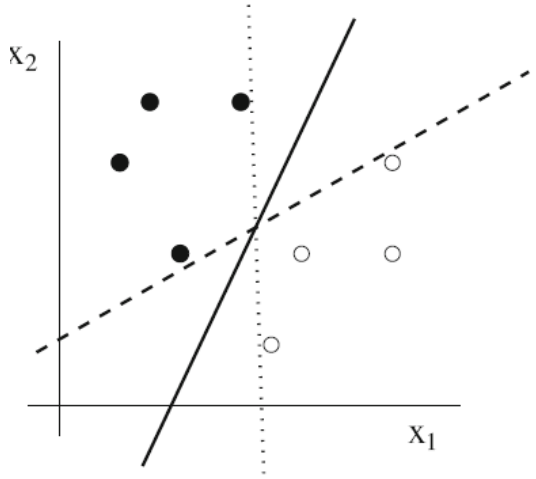
Adecuados para generar ensambles de modelos (*boosting* y *bagging*).

Adecuados para selección de atributos en otros modelos.

Frágiles respecto del conjunto de entrenamiento y susceptibles al overfitting y a modelos super complejos (que requieren *regularización*).

Support Vector Machines

Se basa en dos ideas fundamentales:

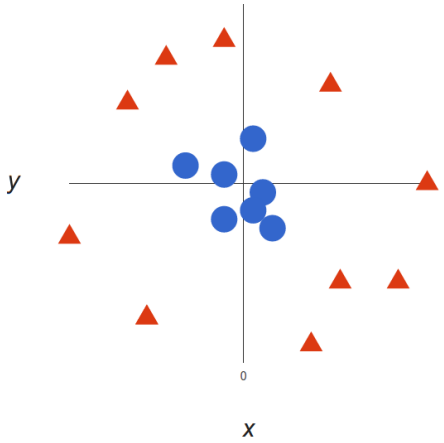


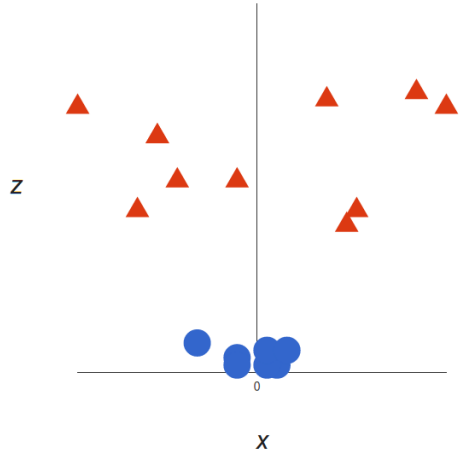
Puntos etiquetados que no sean linealmente separables se pueden mapear a un espacio de más dimensiones (usando una función *kernel*) donde tal separación es posible. El *kernel* a su vez puede facilitar la representación.

Dado un conjunto de puntos etiquetados linealmente separables, es posible encontrar un hiperplano de separación óptima (que maximice el margen de separación) y por lo tanto que sea más inmune al ruido y al *overfitting*.

Support Vector Machines

Un ejemplo arbitrariamente sencillo. Ningún modelo lineal puede separar los puntos rojos de los azules, pero si mapeamos los datos a una tercera coordenada $z=x^2+y^2$, entonces es posible encontrar una separación fácil.





Support Vector Machines

Si el conjunto de datos es separable, entonces existe un hiperplano tal que $\omega\mathbf{x}-b = 0$, (donde ω es el vector normal al plano y **x** es un dato dado. La búsqueda del modelo lineal óptimo es relativamente sencilla (veremos un caso particular con los perceptrones).

El además del "kernel trick", se pueden utilizar diferentes funciones en vez del producto escalar (polinomiales, Gaussianas, hiperbólicas, etc.). Eso implícitamente "*warpea*" el espacio de atributos de manera de maximizar la separación.

Perceptrones

El punto de partida histórico fueron las funciones discriminantes lineales, las cuales dieron origen al primer modelo de aprendizaje, conocido como **perceptrón**.

Luego fueron agregándose diferentes niveles de complejidad, hasta llegar finalmente al actual deep learning.

Cada «capa» de la red neuronal realiza tareas sencillas de clasificación, de nivel de abstracción creciente.

Perceptrones

Las fronteras entre regiones son (hiper)superficies, llamadas *superficies de decisión.*

En el caso más sencillo (pero suficientemente poderoso), las hipersuperficies consisten en hiperplanos, cuya ecuación genérica en el espacio de los patrones es

$$w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + w_{n+1} = 0$$

De esa manera, cada clase tendrá una una *función discriminante lineal*

$$d_k(\mathbf{x}) = w_{1,k} x_1 + w_{2,k} x_2 + \cdots + w_{n,k} x_n + w_{n+1,k}$$

La expresión

$$d_k(\mathbf{x}) = w_{1,k}x_1 + w_{2,k}x_2 + \cdots + w_{n,k}x_n + w_{n+1,k}$$

puede pensarse como

$$\mathbf{w}_k^T\mathbf{x}$$

donde $\mathbf{w}_k^T = [w_{1,k}\ w_{2,k}\ \cdots\ w_{n,k}\ w_{n+1,k}]$ es el *vector de pesos (aumentado)*,

y $\mathbf{x}^T = [x_1\ x_2\ \cdots\ x_n\ 1]$ es el *patrón aumentado.*

Un patron $\mathbf{x}$ pertenece a la clase ω_j cuando la distancia entre $\mathbf{x}$ y $\mathbf{z}_j$ es menor que al centro de cualquier otra clase.

Una primera aproximación para este clasificador consiste en utilizar distancia Euclídea al cuadrado:

$$d(\mathbf{x}, \mathbf{z}_k)^2 = |\mathbf{x} - \mathbf{z}_k|^2 = (\mathbf{x} - \mathbf{z}_k)^T(\mathbf{x} - \mathbf{z}_k) = \mathbf{x}^T\mathbf{x} - 2\mathbf{x}^T\mathbf{z}_k + \mathbf{z}_k^T\mathbf{z}_k$$

El término $\mathbf{x}^T\mathbf{x}$ es independiente de la clase k y por lo tanto puede ser eliminado.

Queremos ahora encontrar el $\mathbf{z}_k$ tal que $-2\mathbf{x}^T\mathbf{z}_k + \mathbf{z}_k^T\mathbf{z}_k$ sea mínimo.

Minimizar $-2\mathbf{x}^T\mathbf{z}_k + \mathbf{z}_k^T\mathbf{z}_k$ es lo mismo que maximizar $\mathbf{x}^T\mathbf{z}_k - \frac{1}{2}\mathbf{z}_k^T\mathbf{z}_k$

Esto puede expresarse como una función discriminante

$$d_k(\mathbf{x}) = \mathbf{x}^T\mathbf{z}_k - \frac{1}{2}\mathbf{z}_k^T\mathbf{z}_k = \mathbf{x}^T\mathbf{z}_k - \frac{1}{2}|\mathbf{z}_k|^2 = \mathbf{x}^T\mathbf{w}_k$$

donde el vector aumentado de pesos $\mathbf{w}_k$ es $\mathbf{w}_k = \left[z_{k,1}\ z_{k,2}\ \cdots\ z_{k,n}\ \frac{1}{2}|\mathbf{z}_k|^2 \right]$

El hiperplano de decisión entre las clases j y k, entonces, queda definida como

$$d_{jk}(\mathbf{x}) = \mathbf{x}^T(\mathbf{z}_j - \mathbf{z}_k) - \frac{1}{2}(|\mathbf{z}_j|^2 - |\mathbf{z}_k|^2) = 0$$

Es fácil ver que este hiperplano es perpendicular al vector que une $\mathbf{z}_j$ con $\mathbf{z}_k$, y que cruza dicho vector exactamente en el punto medio entre ambos centros de clase.

Perceptrones

Dado un discriminante lineal $d_k(\mathbf{x}) = w_{1,k}x_1 + w_{2,k}x_2 + \cdots + w_{n,k}x_n + w_{n+1,k} = \mathbf{w}^T\mathbf{x}$, el cual puede pensarse como el producto escalar entre un patrón aumentado y un vector de pesos aumentado.

El problema del entrenamiento consiste en encontrar un vector de pesos adecuado.

Es más adecuado representar este problema en el *espacio de los pesos*, donde cada valor de $\mathbf{w}$ es representado con un punto, y viceversa.

El espacio de los pesos es un espacio $(n+1)$-dimensional en el cual las coordenadas son $w_1, w_2, \cdots, w_{n+1}$.

La región en este espacio donde $\mathbf{w}^T\mathbf{x} = 0$ es también un hiperplano, que siempre pasa por el origen (sin traslación).

Perceptrones

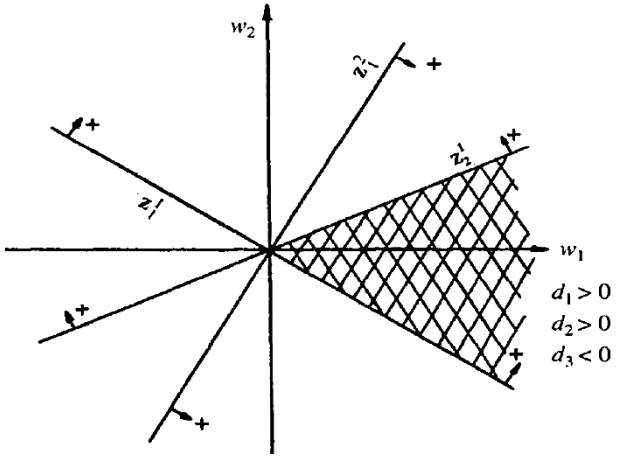
Dado un prototipo $\mathbf{z}_k^i$ de la clase k, todo hiperplano $\mathbf{w}$ tal que $\mathbf{w}^T\mathbf{z}_k^i > 0$ probablemente clasificará a $\mathbf{z}_k^i$ como perteneciente a ω_k.

Supongamos tener dos clases ω_1, ω_2 linealmente separables, cada una con N_1, N_2 prototipos.

En ese caso queremos encontrar un vector de pesos $\mathbf{w}$ tal que

$$\mathbf{w}^T\mathbf{z}_1^i > 0,\ \forall \mathbf{z}_1^i \in \omega_1, i = 1, \cdots, n_1$$

$$\mathbf{w}^T\mathbf{z}_2^i < 0,\ \forall \mathbf{z}_2^i \in \omega_2, i = 1, \cdots, n_2$$



Cuando ocurre un error en el entrenamiento, ello se debe a que dado el vector de pesos $\mathbf{w}_k$, para alguna clase, con dicho vector algún prototipo queda mal clasificado.

Corregir el error implica colocar a dicho prototipo en la clasificación adecuada, es decir, encontrar un nuevo $\mathbf{w}'_k$ en la zona adecuada del espacio de los pesos.

El procedimiento habitual es mover $\mathbf{w}_k$ en dirección perpendicular al $\mathbf{w}_k$ mal clasificado (hacia la zona positiva si fue un falso negativo, o hacia la zona negativa si fue un falso positivo).

Eso se logra simplemente sumando (restando) el prototipo mal clasificado al vector de pesos. Se utiliza un *factor de corrección* c para regular la convergencia del procedimiento.

$$\mathbf{w}'_k = \mathbf{w}_k + (-)c\mathbf{z}_k$$

Si más de un prototipo fue mal clasificado, se elige uno entre todos al azar.

El procedimiento se *itera* hasta que todos los prototipos queden correctamente clasificados.

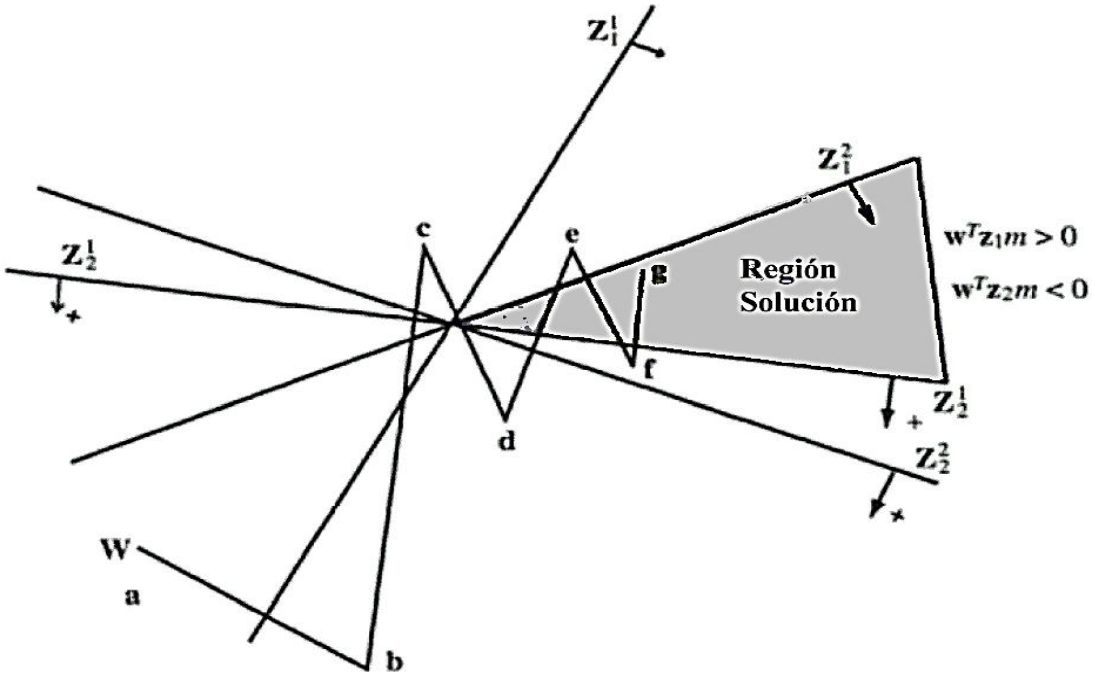
Es importante observar que existe un teorema que demuestra la convergencia del procedimiento anterior, siempre que c sea positivo, y las clases sean separables.

Modificar el valor de c equivale a multiplicar los prototipos sin alterar su separabilidad.

Los diferentes criterios para la elección de c (fijo, fraccional, absoluto, etc.) determinan la velocidad de convergencia.

Perceptrones

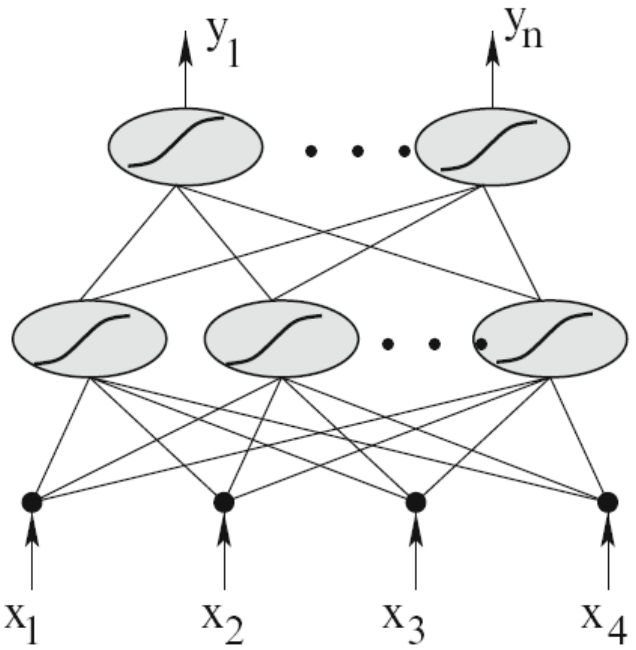
Un ejemplo de aprendizaje por medio de la regla de la corrección del error. El valor de c se va decrementando linealmente.



Perceptrones

Para resolver problemas de complejidad y abstracción mayores, se utilizan redes de perceptrones multicapa, donde a su vez cada capa intermedia implementa una función de activación logística (o no lineal sigmoidal).

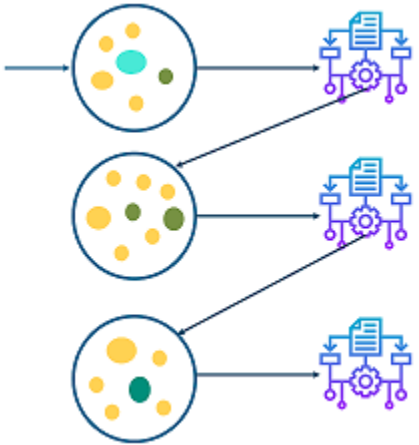
Finalmente, el algoritmo de aprendizaje por la regla de corrección del error deviene en el método conocido como *backpropagation*.



Ensambles

La idea es combinar varios clasificadores (débiles) en uno más fuerte. Esto se puede hacer de varias maneras, pero las más usuales son *boosting* y *bagging*.

La idea subyacente al *boosting* es la de tener un modelo donde los casos de entrenamiento tienen peso asignado (igual al principio), luego iterar el aprendizaje dándole mayor peso a los casos falsos. Cada modelo resultante tiene un determinado índice de éxito. Los modelos más exitosos tienen mayor incidencia en el resultado final.

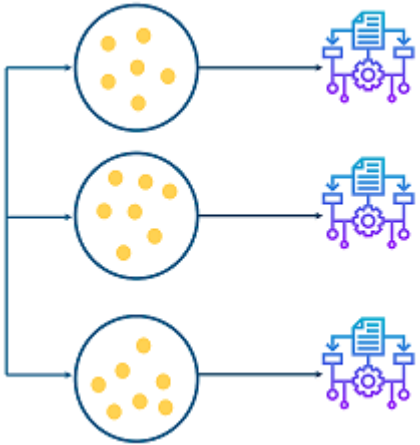


El *boosting* permite corregir el sesgo (*underfitting*).

Ensambles

El *bagging*, en cambio, computa en paralelo varios modelos independientes (elegidos con un muestreo insesgado), para luego finalmente decidir por mayoría simple. El *bagging* permite corregir la variancia (*overfitting*).

El ejemplo más usual son los random forests, que consisten en computar cientos de árboles de decisión definidos a partir de decisiones aleatorias, para luego clasificar de acuerdo al voto de la mayoría.



Notebooks de ejemplo

En el siguiente NB que ya vimos, tenemos un ejemplo más complejo de clasificador utilizando SVM:
https://github.com/manlio99/Materia-de-aprendizaje/blob/master/5_DataMining/2_classification.ipynb
La implementación de SciKitLearn utiliza kernel radial (Gaussiano) y el parámetro gamma es la inversa de la variancia. Valores pequeños warpean alrededor de los puntos elegidos para entrenamiento con mayor alcance y viceversa.

Práctico 4

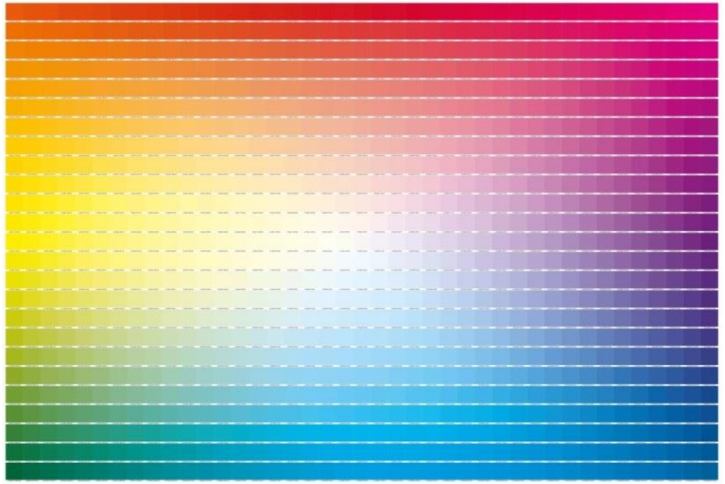
Elegir uno de entre los siguientes proyectos (se agregan links a las NB correspondientes) y aplicar clasificación para resolverlo. Cada proyecto tiene datasets de diferentes características (texto, imágenes, audio, etc.) por lo que tiene diferente dificultad (indicada con un semaforito).

Para los más sencillos trabajar individualmente y aplicar más de un clasificador, utilizar diferentes parámetros, comparar evaluaciones.

Para los más complejos, trabajar en grupos de dos personas y aplicar un único modelo.

Ejercicio 4.1

Rainbow Guru: El dataset es un archivo CSV con 865 nombres de color y sus correspondientes valores RGB (red, green, and blue) y el mismo valor en hexadecimal. El objetivo es predecir el nombre de un color dado a partir de su RGB o de su hexadecimal.



https://github.com/manlio99/Materia-de-aprendizaje/blob/master/3_MidtermProjects/ProjectRGB

Ejercicio 4.2

COVID-19 Detection in Chest X-Ray Images: El dataset tiene 600+ imágenes en tres carpetas (COVID-19, normal, neumonía). Desarrollar un extractor de características de las imágenes, y luego entrenar un clasificador.

https://github.com/manlio99/Materia-de-aprendizaje/blob/master/3_MidtermProjects/ProjectCXR

Ejercicio 4.3

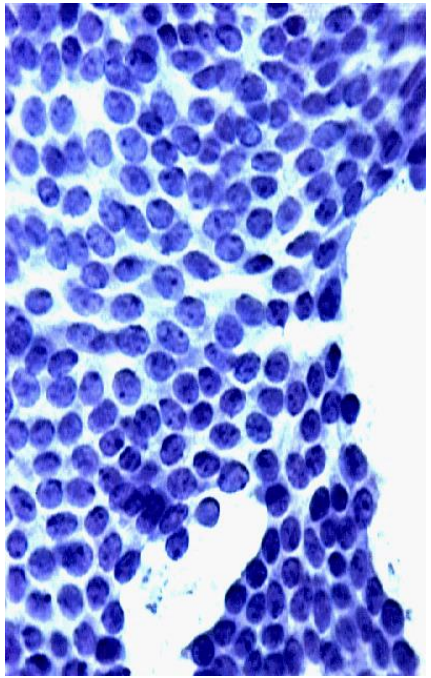
Fake News: El dataset tiene dos archivos, cada uno con noticias verdaderas o con noticias falsas. Los archivos tienen cuatro columnas (título, texto, asunto, fecha). Entrenar un modelo que dada una noticia prediga si es falsa o no.

https://github.com/manlio99/Materia-de-aprendizaje/blob/master/3_MidtermProjects/ProjectFN

Ejercicio 4.4

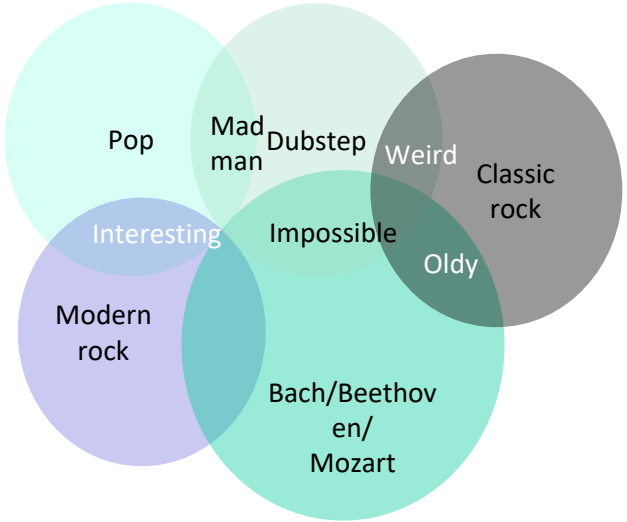
Winsconsin Breast Cancer dataset. Contiene una tabla con 569 datos de histología mamaria humana, 357 casos de tejido tumoral benigno y 212 tejido maligno. Los datos son 32 atributos radiómicos (tamaños, formas, texturas, etc.). El objetivo es desarrollar un clasificador que a través de los atributos permita predecir la benignidad o malignidad de un tumor.

https://github.com/manlio99/Materia-de-aprendizaje/blob/master/3_MidtermProjects/ProjectBCD



Ejercicio 4.5

Géneros musicales a oído. El dataset contiene 1,000 audios de 30 segundos cada uno, en 10 géneros musicales diferentes (100 tracks de cada uno, en formato .wav 16 bit mono). Se provee también una notebook para abrir y manipular estos tracks. Extraer features y desarrollar un clasificador que prediga el género musical a partir del audio.



https://github.com/manlio99/Materia-de-aprendizaje/blob/master/3_MidtermProjects/ProjectGTZ

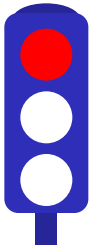
Ejercicio 4.6

Transacciones fraudulentas: El dataset contiene 280.000+ transacciones de tarjetas de crédito de usuarios en Europa durante 2013, de las cuales 492 fueron etiquetadas como fraudulentas. Se presentan 28 variables que corresponden al PCA (por razones de anonimización), además del timestamp y el monto de la transacción. El objetivo es desarrollar un clasificador que detecte cuándo una transacción es fraudulenta.

https://github.com/manlio99/Materia-de-aprendizaje/blob/master/3_MidtermProjects/ProjectCCF

Ejercicio 4.7

Señales camineras: El objetivo es obtener un modelo que determine el tipo de señal de tráfico, donde la dificultad es que las tomas fueron realizadas en condiciones realistas. Se provee un dataset etiquetado y dividido en las tres partes (train, validate, test).



https://github.com/manlio99/Materia-de-aprendizaje/blob/master/3_MidtermProjects/ProjectRTS