Tema 4: Predicción - Parte 1: Clasificación

1. El problema de la clasificación

¿Qué se hace en ciencia de datos?

Real world ➡ Raw Data ➡ Process and Analyze ➡ Meaningful Data ➡ Useful Insights

Núcleo del proceso: el aprendizaje automático (subrama de la IA cuyo objetivo es que las computadoras aprendan)

Tipos de variables

- Numérica (ordinal) que se divide en **continua** (temperatura, energía, peso, estatura…) y **discreta** (edad en años, nivel de educación, número de hijos…)

- Categórica o nominal (no hay un orden): se divide en **booleana o lógica** (padece o no una enfermedad, es o no mayor de edad) y **otras** (género, grupo sanguíneo, lugar nacimiento…)

Técnicas de aprendizaje automático: Clasificación

Las técnicas de aprendizaje se dividen en varios tipos. En este tema vamos a ver la **clasificación** (que pertenece a supervisado y categórico)

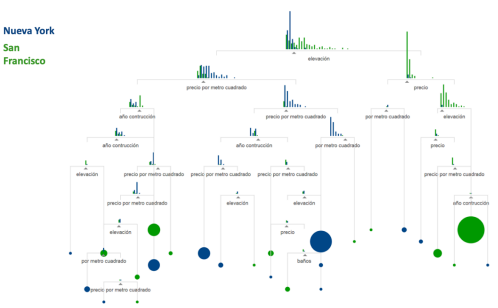
	Supervisado	**No supervisado**
Categórico	**CLASIFICACIÓN**	Reglas de asociación
Continuo	Regresión	Clustering

Clasificación: técnica de aprendizaje automático más conocida.
- **Problema**: predecir una clase determinada (categórica) para un objeto.
- **Tarea**: dado un conjunto de ejemplos ya clasificados, construir un modelo (entrenamiento) que permita clasificar nuevos casos (predicción)

Aprendizaje supervisado: se conoce la verdadera clase de cada uno de los ejemplos de entrenamiento que se utilizan para construir el *clasificador*.
Un clasificador o modelo puede ser un conjunto de reglas, un árbol de decisión, una red neuronal, etc.

Aplicaciones típicas: aprobación de créditos, marketing directo, detección de fraudes, diagnóstico medico, reconocimiento de imágenes...

Ejemplo de clasificación: digamos que debemos determinar si una casa esta en San Francisco o Nueva York en base a una serie de características como evaluación, precio, año de construcción… Lo que hacemos es ir construyendo el árbol eligiendo en cada modelo la variable que permite separar las clases de la mejor manera posible.



Deep Learning

Tipo de machine learning que entrena a una computadora para que realice tareas como lo hacemos los seres humanos, como el reconocimiento del habla, identificación de imágenes… Reconoce patrones mediante el uso de muchas capas de procesamiento.

1.1. Definición del problema de clasificación

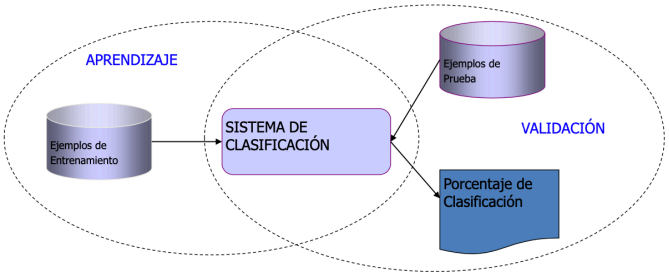
El problema de clasificación se trata de predecir una clase determinada (categórica) para un objeto. El problema fundamental de la clasificación está directamente relacionado con la separabilidad de las clases. Según el modelo, se consigue un tipo de separación u otro.

Un objeto se describe a través de un conjunto de características (variables o atributos), por ejemplo, para conceder un crédito pues tenemos las variables: ingresos, deudas, propiedades…

El **objetivo de la tarea de clasificación**: Clasificar el objetivo dentro de una de las categorías de la clase C={c1,…,ck}. Las características o variables elegidas dependen del problema de clasificación

$$f: X_1 \times X_2 \times \ldots \times X_n \rightarrow C$$

1.2. Etapas en el proceso de clasificación



1. **Construcción del modelo que describe el conjunto de clases (predeterminadas): Fase de Aprendizaje:**
   Cada ejemplo (supla) se sabe que pertenece a una clase (etiqueta del atributo clase). Se utiliza un conjunto de ejemplos para la construcción del modelo: conjunto de entrenamiento (training set). El modelo obtenido se representa como un conjunto de reglas de clasificación, árboles de decisión, fórmula matemática, ...

2. **Utilización del modelo: validación**
   Estimación de la precisión del modelo:
   Se utiliza un conjunto de ejemplos distintos de los utilizados para la construcción del modelo: conjunto de prueba (test set). Si el conjunto de test no fuese independiente del de entrenamiento ocurría un proceso de sobreajuste (*overfitting*). Para cada ejemplo de test se compara la clase determinada por el modelo con la clase real (conocida). El ratio de precisión es el porcentaje de ejemplos de test que el modelo clasifica correctamente

Training data ➡ Algoritmo de clasificación ➡ Modelo (classifier)
Testing Data ➡ Classifier ➡ Unseen Data

1.3. Criterios para evaluar un clasificador

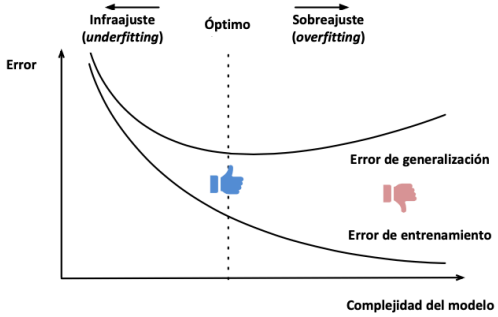
- Precisión o exactitud (s, ε): En ocasiones se debe considerar el costo de la clasificación incorrecta

- Velocidad: Tiempo necesario para la construcción del modelo y para usarlo
- Robustez: capacidad para tratar con valores desconocidos
- Escalabilidad: Aumento del tiempo necesario (en construcción y evaluación) con el tamaño de la BD
- Interpretabilidad: comprensibilidad del modelo obtenido
- Complejidad del modelo: Tamaño del árbol de clasificación, número de reglas, antecedentes en las reglas,...
- Matriz de confusión: Dado un problema de clasificación con m clases, una matriz de confusión es una matriz m x m en la que una entrada ci,j indica el número de ejemplos que se han asignado a la clase cj, cuando la clase correcta es ci

Name	Gender	Height	Clase Correcta	Clase predicha
Kristina	F	1.6m	Short	Medium
Jim	M	2m	Tall	Medium
Maggie	F	1.9m	Medium	Tall
Martha	F	1.88m	Medium	Tall
Stephanie	F	1.7m	Short	Medium
Bob	M	1.85m	Medium	Medium
Kathy	F	1.6m	Short	Medium
Dave	M	1.7m	Short	Medium
Worth	M	2.2m	Tall	Tall
Steven	M	2.1m	Tall	Tall
Debbie	F	1.8m	Medium	Medium
Todd	M	1.95m	Medium	Medium
Kim	F	1.9m	Medium	Tall
Amy	F	1.8m	Medium	Medium
Wynette	F	1.75m	Medium	Medium

Matriz de confusión

	Asignación		
	Short	Medium	Tall
Short	0	4	0
Medium	0	5	3
Tall	0	1	2



Acierto $s = \frac{VP + VN}{VP + FP + FN + VN}$

Error $\varepsilon = 1 - s$

real \ predicho	P	N
P	VP verdadero positivo	FN falso negativo
N	FP falso positivo	VN verdadero negativo

1.4. Algunos ejemplos de clasificadores sencillos

Clasificador ZeroR: Es el clasificador más sencillo. Todas las instancias se clasifican como pertenecientes a la clase mayoritaria. Se usa como caso base para realizar comparaciones (cualquier algoritmo debería al menos igualar su rendimiento).

Clasificador OneR: Objetivo: crear un clasificador formado por reglas con una única variable en el antecedente. Se generan todas las reglas del tipo:

SI variable=valor ENTONCES clase=categoría

También se usa como algoritmo para realizar comparaciones. ***Nota**: Los valores perdidos se utilizan como un valor especial de las variables*

Clasificador del vecino más cercano (k-NN): es uno de los clasificadores más utilizados por su simplicidad. El proceso de aprendizaje de este clasificador consiste en almacenar una tabla con los ejemplos disponibles, junto a la clase asociada a cada uno de ellos. Ante un nuevo ejemplo a clasificar, se calcula su distancia (usaremos la Euclídea) con respecto a los *n* ejemplos existentes en la tabla, y se consideran los *k* más cercanos. El nuevo ejemplo se clasifica según la clase mayoritaria de los *k* ejemplos más cercanos. El caso más sencillo es cuando *k = 1 (1-NN)*. **IMPORTANTE:** Los atributos deben estar normalizados [0,1] para no priorizarlos sobre otros.

k-NN devuelve la clase más repetida de entre todos los k ejemplos de entrenamiento cercanos a *xq*.
Diagrama de Voronoi: superficie de decisión inducida por 1-NN para un conjunto dado de ejemplos de entrenamiento.

1.5. Evaluación de clasificadores

El objetivo de los métodos de validación es realizar una estimación honesta de la bondad del clasificador construido. Utilizar como bondad la tasa de acierto sobre el conjunto de entrenamiento no es realista (el porcentaje obtenido suele ser demasiado optimista debido a que el modelo estará sobreajustado a los datos utilizados durante el proceso de aprendizaje).
Existen distintas técnicas de validación de clasificadores, entre ellas:

Hold-out: Consiste en dividir la BD en dos conjuntos independientes: entrenamiento (CE) y test (CT). El tamaño del CE normalmente es mayor que el del CT. Los elementos del CE suelen obtenerse mediante muestreo sin reemplazamiento de la BD inicial. El CT está formado por los elementos no incluidos en el CE. Suele utilizarse en BBDD de tamaño grande

$$\text{Test set (\%)} + \text{Training set (\%)} = 100\%$$

Validación cruzada: Consiste en:

1. Dividir la BD en k subconjuntos (folds), {S1,...,Sk} de igual tamaño
2. Aprender k clasificadores utilizando en cada uno de ellos un CE distinto. Validar con el CT correspondiente

$$CE = S1 \cup ... \cup Si-1 \cup Si+1 \cup ...Sk$$
$$CT = Si$$

3. Devolver como tasa de acierto (error) el promedio obtenido en las k iteraciones

Validación cruzada estratificada: Los subconjuntos se estratifican en función de la variable clase. Valores típicos de k=5, 10. Suele utilizarse en BBDD de tamaño moderado.

Leaving-one-out: Es un caso especial de validación cruzada en el que *k* es igual al número de registros. Tiene la ventaja de que el proceso es determinista y de que en todo momento se utiliza el máximo posible de datos para la inducción del clasificador. Se utiliza en BBDD muy pequeñas, debido a su alto costo computacional

Bootstrap: Está basado en el proceso de muestreo con reposición. A partir de una BD con n registros se obtiene un CE con n casos. Como CT se utilizan los registros de la BD no seleccionados para el CE, ¿Cuántos casos habrá en CT? ¿qué porcentaje respecto a n?. La probabilidad de que se elija un registro es 1/n. La probabilidad de que no se elija es 1-1/n. Se hacen n extracciones, por tanto la probabilidad de que un ejemplo no sea elegido es ≃ 0.368. El CE tendrá aproximadamente el 63.2% de los registros de la BD y el CT el 36.8 %. Esta técnica se conoce como 0.632 bootstrap. El error sobre el CT suele ser bastante pesimista por lo que se corrige.

$$error = 0.632 * error_{CT} + 0.368 * error_{CE}$$

2. Clasificación con árboles y reglas
2.1 Árboles de decisión
2.1.1 Definición de árboles de decisión

Un **árbol de decisión** es un clasificador que en función de un conjunto de atributos permite determinar a qué clase pertenece el caso objeto de estudio. La estructura del un árbol de decisión es:
- Cada hoja es una categoría (clase) de la variable objeto de la clasificación
- Cada nodo es un nodo de decisión que especifica una prueba simple a realizar
- Los descendientes de cada nodo son los posibles resultados de la prueba del nodo

2.1.2 Construcción de árboles de decisión

El proceso de generación de un árbol de decisión consiste en dos fases:

- Construcción del árbol: Al principio todos los ejemplos de entrenamiento están en el nodo raíz. Se van dividiendo recursivamente los ejemplos en base a los atributos seleccionados.
- Poda del árbol: Identificar y quitar ramas que describen ruido o datos anómalos

Uso del árbol de decisión: Clasificar un ejemplo desconocido: Comprobar los valores de los atributos del ejemplo contra el árbol de decisión

Algoritmo básico (algoritmo voraz): se construye el árbol mediante la técnica divide y vencerás aplicada de forma recursiva. Al principio todos los ejemplos de entrenamiento están en el nodo raíz. Los atributos son categóricos (si son continuos, se discretizan previamente). Los ejemplos se dividen recursivamente basándose en atributos seleccionados. Los atributos de test se seleccionan en base a una medida heurística o estadística (por ejemplo, la ganancia de información).

Condiciones para terminar el particionamiento: Todos los ejemplos para un nodo dado pertenecen a la misma clase. No quedan más atributos para seguir particionando. En este caso se utiliza el voto de la mayoría para clasificar en el nodo hoja. NO quedan ejemplos.

Entrada: Sea T el conjunto de ejemplos, $A=\{A_1, \ldots, A_n\}$ el conjunto de atributos y $C=\{C_1, \ldots, C_k\}$ el conjunto de valores que puede tomar la clase

ConstruirArbol (T,C,A)

1. Crear un nodo RAIZ para el árbol
2. Si todos los ejemplos en T pertenecen a la misma clase C_i, devolver el nodo RAIZ con etiqueta C_i
3. Si $A=\Phi$ devolver el nodo RAIZ con etiqueta C_i donde C_i es la clase mayoritaria en T
4. a ← el atributo de A que mejor clasifica T
5. Etiquetar RAIZ con a
6. Para cada valor v_i de a hacer
    1. Añadir una nueva rama debajo de RAIZ con el test $a=v_i$
    2. Sea T_i el subconjunto de T en el que $a = v_i$
    3. ConstruirArbol (T_i,C,A-a)
7. Devolver RAIZ

Dependiendo del orden en el que se van tomando los atributos obtenemos clasificadores de distinta complejidad. Lo ideal sería tomar en todo momento el atributo que mejor clasifica.

2.1.3 Criterios de selección de variables

Definición de entropía. Dados: un problema con dos clases (positiva y negativa), S un conjunto de ejemplos, la entropía mide la incertidumbre en S.

$$Entropia(S) = H(S) = -p_+ \cdot \log_2 p_+ - p_- \cdot \log_2 p_- =$$
$$p_+ \cdot \log_2(1/p_+) + p_- \cdot \log_2(1/p_-)$$

Existen distintos criterios para seleccionar el atributo X* de test en cada momento. Algunos de los más conocidos son:

***InfoGain*: Ganancia de información (ID3)** ➡ Objetivo: Seleccionar el atributo con la mayor ganancia de información. Consideremos un problema con dos clases: *P* y *N:* Sea *S* el conjunto de ejemplos que contiene *p* elementos de la clase *P* y *n* elementos de la clase *N, la* cantidad de información necesaria para decidir si un ejemplo arbitrario dentro de *S* pertenece a *P* o a *N* viene definido por

$$H(C)=H(p,n)=-\frac{p}{p+n}\log_2\frac{p}{p+n}-\frac{n}{p+n}\log_2\frac{n}{p+n}$$

La entropía mide la aleatoriedad o sorpresa o incertidumbre al predecir una clase. Consideremos que utilizando un atributo A, el conjunto S se puede dividir en v conjuntos {S1, S2 , ..., Sv}. Si Si contiene pi ejemplos de P y ni ejemplos de N, la entropía o la información que se espera sea necesaria para clasificar los objetos en todos los subárboles Si es

$$H(C/A)=\sum_{i=1}^{v}\frac{p_i+n_i}{p+n}H(p_i,n_i)$$

La información que se podría ganar con una rama que considere *A* es Es la reducción en la entropía al considerar el atributo *A*

$$Gain(A)=H(p,n)-H(C/A)$$

Con la ganancia de información buscamos maximizar, cuánto más lejos del 0 mejor.

GINI (CART) ➡ Si un conjunto de datos T tiene ejemplos pertenecientes a n clases, el índice Gini, se define como la siguiente fórmula, donde pj es la frecuencia relativa de la clase j en T.

$$gini(T)=1-\sum_{j=1}^{n}p_j^2$$

Si se divide un conjunto de datos T en dos subconjuntos T1 y T2 de tamaños N1 y N2 respectivamente, el índice de Gini de los datos separados conteniendo ejemplos de las n clases se define como

$$gini_{split}(T)=\frac{N_1}{N}gini(T_1)+\frac{N_2}{N}gini(T_2)$$

Se elige para dividir el nodo, el atributo que proporciona el índice ginisplit(T) más pequeño (es necesario enumerar todos los posibles puntos de división para cada atributo).
Se busca minimizar la medida, no se puede llegar al 1 como mucho llegaremos al 0.5. Cuánto más cerca del 0 mejor.

InfoGain: Ganancia de información (ID3)	*GainRatio*: Ganancia de información modificada (C4.5)	*GINI (CART)*
$X^*=\max_X(H(C)-H(C\|X))$	$X^*=\max_X\frac{H(C)-H(C\|X)}{H(X)}$	$X^*=\max_X(G(C)-G(C\|X))\quad con\quad G=1-\sum_{i=1}^{n}p_i^2$

2.1.4 Particionamiento del espacio con un árbol de decisión

Los árboles particionan el espacio de soluciones de forma exhaustiva: variables discretas o continuas

2.1.5. Ventajas e inconvenientes del uso de árboles de decisión en clasificación

Ventajas:
- Los árboles de decisión son fáciles de utilizar y eficientes
- Las reglas que generan son fáciles de interpretar
- Escalan mejor que otros tipos de técnicas
- Tratan bien los datos con ruido

Inconvenientes:
- No manejan de forma fácil los atributos continuos
- Tratan de dividir el dominio de los atributos en regiones rectangulares y no todos los problemas son de ese tipo
- Tienen dificultad para trabajar con valores perdidos
- Pueden tener problemas de sobreaprendizaje
- No detectan correlaciones entre atributos

2.1.6. Algunos algoritmos de minería de datos basados en árboles de decisión

Elección de atributos de prueba: La forma en que se eligen las variables de prueba tiene una gran influencia en el rendimiento del árbol de decisión

Divisiones: Para algunos atributos y dominios las divisiones son obvias (por ejemplo, variables categóricas), mientras que para otros es difícil determinar (por ejemplo, variables numéricas)

Estructura del árbol: Para mejorar el rendimiento al aplicar el árbol de decisión, es preferible un árbol equilibrado

Criterio de detención: En muchos problemas puede ser necesario detenerse antes para evitar árboles muy grandes. Es un equilibrio entre precisión y rendimiento

Poda: A veces es necesario llevar a cabo un proceso de poda para mejorar el rendimiento del árbol durante la clasificación: puede haber arboles balanceados o de profundidad

ID3 ➡ Crea el árbol utilizando conceptos de teoría de información. Intenta reducir el número de comparaciones. ID3 elige el atributo test con máxima ganancia de información (basada en la entropía que se utiliza como una medida de la cantidad de incertidumbre o sorpresa en un conjunto de datos). ID3 elige el atributo con mayor ganancia de información. La **ganancia de información** diferencia entre la cantidad de información que se necesita para hacer una clasificación antes de hacer la división y después.
Se calcula determinando la diferencias entre la entropía del conjunto de datos de partida y la suma ponderada de las entropías una vez dividido el conjunto de ejemplos

$$Gain(S,A) = Entropia(S) - \sum_{v \in valores(A)} \frac{|S_v|}{|S|} Entropia(S_v)$$

Esquema del algoritmo ID3:
1. Seleccionar el atributo A que maximice la ganancia G(S,A)
2. Crear un nodo para ese atributo con tantos sucesores como valores tenga
3. Introducir los ejemplos en los sucesores según el valor que tenga el atributo A
4. Por cada sucesor:
   Si sólo hay ejemplos de una clase, Ck
   Entonces etiquetarlo con Ck

Si no, llamar a ID3 con un conjunto de ejemplos formado por los ejemplos de ese nodo, eliminando la columna del atributo A

Termina cuando todos los datos del nodo son de la misma clase y la entropía es cero.
Se usa en árboles cortos frente a largos, con los atributos que producen una mayor ganancia de información cerca de la raíz. Refinamiento de ID3: cuándo parar en la construcción del árbol, atributos continuos, otras medidas de selección de atributos, atributos con coste diferente y ejemplos incompletos

C4.5 ➡ Mejora a ID3 en los siguientes aspectos:

- Datos perdidos: Cuando se construye el árbol, los datos perdidos se ignoran (el árbol de decisión se construye mirando sólo los registros que tienen valor para ese atributo). Para clasificar un ejemplo con valor perdido, éste se predice en base a lo que se sabe sobre los valores del atributo para otros registros.

- Datos continuos: Se divide en rangos en base a los valores encontrados en el conjunto de entrenamiento

- Propone soluciones para el sobreaprendizaje. Posibilidades

  - **Pre-poda**: se decide cuándo dejar de subdivivir el árbol. No se divide un nodo si se tiene poca confianza en él (no es significativa la diferencia de clases), o se valida con un conjunto de test independiente y se para cuando la curva del conjunto de test empieza a subir.

    Hay dos estrategias de post-poda en C4.5

    - Reemplazamiento de subárboles: Se reemplaza un subárbol por una hoja si al hacerlo el error es similar al original

    • Elevación de subárbol: Reemplaza un subárbol por su subárbol más utilizado (un subárbol se mueve de su localización a un nodo superior en el árbol)

  - **Post-poda:** se construye el árbol y después se poda

Reglas: C4.5 permite clasificación mediante el árbol o a través de las reglas que se generan a partir de él.

Selección de atributos: D3 favorece atributos con muchas divisiones, EN C4.5 se utiliza como criterio de selección el ratio de ganancia de información que tiene en cuenta la cardinalidad de cada división

2.2 Clasificadores basados en reglas
2.2.1 Algoritmos de minería de datos basados en reglas

Una regla de clasificación está formada por: un antecedente, que contiene un predicado que se evaluará como verdadero o falso con respecto a cada ejemplo de la base de ejemplos y un consecuente, que contiene una etiqueta de clase.
Se puede utilizar un árbol de decisión para generar reglas pero no son equivalentes: el árbol tiene implícito un orden en el que se van comprobando los atributos. Con las reglas, en general, no hay orden. El árbol se crea mirando todas las clases. Cuando se genera una regla sólo se examina una clase. Líneas principales en los algoritmos de minería de datos para la obtención de reglas:

- Generación de reglas mediante árboles de decisión
- Generación de reglas mediante cobertura

2.2.2 Generación de reglas mediante árboles de decisión

Podemos extraer las reglas de un árbol de decisión y usarlas como un SBR. El algoritmo para obtener el SBR, para cada camino de la raíz del árbol a un nodo hoja, genera una regla y la añade al conjunto final (SBR).

Lista de decisión: Las reglas están ordenadas. En estos clasificadores lo que se intenta es buscar las reglas que cubran el mayor número de ejemplos positivos (o el menor de ejemplos negativos). No se realiza un particionamiento exhaustivo del espacio de soluciones.

2.2.3 Generación de reglas mediante cobertura

Los algoritmos de cobertura intentan generar reglas que cubran exactamente una clase. Habitualmente generan la mejor regla posible optimizando la probabilidad de clasificación deseada. A diferencia de los algoritmos de generación de árboles (seleccionan el mejor atributo), suelen elegir el mejor par (atributo,valor).

Ejemplo: Si se quiere generar una regla para clasificar personas altas, se busca una regla:

Si ???? Entonces clase = alto

El objetivo para algoritmos de cobertura es reemplazar ??? por el predicado que obtenga la mejor probabilidad de que la clase sea alto.

Ejemplo básico de algoritmo de cobertura: **1R** (genera un conjunto de reglas equivalente a un árbol de decisión de primer nivel). Elige el mejor atributo para realizar la clasificación basándose en datos de entrenamiento.

Otro ejemplo **PRISM**: genera reglas para cada clase mirando en el conjunto de entrenamiento y añadiendo reglas que describan todos los ejemplos de dicha clase

Para seleccionar la mejor regla utilizaremos la exactitud **s** y, en caso de empate, la regla que representa a más ejemplos (la más general).

Entonces el algoritmo basado en cobertura:
Genera reglas para todas las categorías (puede modificarse para incluir regla por defecto). No garantiza obtener el conjunto óptimo de reglas, debido a que actúa de forma voraz incluyendo una condición en cada paso. A no ser que haya inconsistencias en la BD consigue una clasificación perfecta→sobreajusta a los datos

¿Qué ocurre con los atributos numéricos? Condiciones del tipo {<,≤,≥,>}
Existen algoritmos basados en cobertura/recubrimiento mucho más avanzados que el descrito

3. Clasificación con otras técnicas

3.1 Clasificadores basados en métodos bayesianos

3.3.1 Teorema de Bayes e hipótesis MAP

1. Clasificadores basados en métodos bayesianos

Los métodos probabilísticos/bayesianos representan la incertidumbre asociada a los procesos de forma natural. Esto supone una gran ventaja sobre otros métodos.

1.1. Teorema de Bayes e hipótesis MAP

El **teorema de Bayes** orientado a un problema de clasificación con n variables tiene la siguiente expresión:

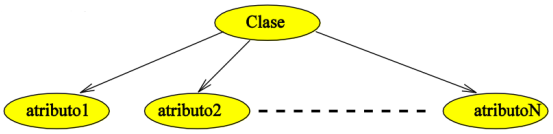
$$P(C|A_1,\ldots,A_n)=\frac{P(A_1,\ldots,A_n|C)P(C)}{P(A_1,\ldots,A_n)}$$

Hipótesis MAP (máxima a posteriori): si queremos clasificar un nuevo caso (a1,…,an) y la variable clase C tiene k posibles categorías Ωc={c1,…,ck}, lo que nos interesa es identificar la más probable y devolverla como clasificación. Problema: que hay que trabajar con la distribución conjunta y eso normalmente es inmanejable/

$$\begin{aligned} c_{MAP} &= \arg\max_{c\in\Omega_C} P(c|a_1,\ldots,a_n) \\ &= \arg\max_{c\in\Omega_C} \frac{P(a_1,\ldots,a_n|c)P(c)}{P(a_1,\ldots,a_n)} \\ &= \arg\max_{c\in\Omega_C} P(a_1,\ldots,a_n|c)P(c) \end{aligned}$$

3.3.1 Clasificador Naïve Bayes

Es el modelo de red bayesiana orientada a clasificación más simple. Supone que todos los atributos son independientes conocida la variable clase. Gráficamente



En un Naïve Bayes (NB) la hipótesis MAP queda como:

$$c_{MAP} = \arg\max_{c\in\Omega_C} P(c|a_1,\ldots,a_n) = \arg\max_{c\in\Omega_C} P(c)\prod_{i=1}^{n} P(a_i|c)$$

A pesar de la suposición poco realista realizada en el NB, este algoritmo se considera un estándar y sus resultados son competitivos con la mayoría de los clasificadores.

¿Cómo se estiman estas probabilidades? Estimación de parámetros

Pues encontramos:

- Las **variables discretas**: *P(x|ci)* se estima como la frecuencia relativa de ejemplos que teniendo un determinado valor de *x* pertenecen a la clase *ci*. Estimación por máxima verisimilitud (EMV). Sea n(xi) el no de veces que aparece xi=xi en la BD y n(xi,xj) el no de veces que aparece el par (xi=xi,xj=xj) en la BD, entonces:

$$p(x_i|x_j) = \frac{n(x_i,x_j)}{n(x_j)}$$

  Suavizando por la corrección de Laplace:

$$p(x_i|x_j) = \frac{n(x_i,x_j)+1}{n(x_j)+|\Omega_{X_i}|}$$

  Cuando hay muchos casos, tiende a la EMV, si hay pocos casos tiende a la uniforme

- Las **variables numéricas**: *P(x|ci)* se estima mediante una función de densidad gaussiana. Es decir, se asume que los valores numéricos siguen una distribución normal, es decir, para cada categoría de la variable clase se estima una distribución normal (de media μ y desviación estándar σ). Evidentemente, en unos casos la aproximación realizada será mejor que en otros

$$P(x|c_i) = N(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{(x-\mu)^2}{2\sigma^2}}$$

Ventajas:
- Es fácil de implementar
- Obtiene buenos resultados en gran parte de los casos

Desventajas:
- Asumir que las variables tienen independencia condicional respecto a la clase lleva a una falta de precisión
- En la práctica, existen dependencias entre las variables. P. ej.: en datos hospitalarios:
  Perfil: edad, historia familiar, etc.
  Síntomas: fiebre, tos, etc.
  Enfermedad: cáncer de pulmón, diabetes, etc.

- Con un clasificador Naïve Bayes no se pueden modelar estas dependencias
- Solución: Redes de creencia bayesianas, que combinan razonamiento bayesiano con relaciones causales entre los atributos

3.2 Clasificadores basados en instancias

Están basados en el aprendizaje por analogía. Se encuadran en el paradigma perezoso de aprendizaje, frente al voraz al que pertenecen los paradigmas anteriores.

> *Perezoso (lazy): El trabajo se retrasa todo lo posible. No se construye ningún modelo, el modelo es la propia BD o conjunto de entrenamiento. Se trabaja cuando llega un nuevo caso a clasificar: Se buscan los casos más parecidos y la clasificación se construye en función de la clase a la que dichos casos pertenecen*

Los algoritmos más conocidos están basados en la regla del vecino más próximo.

Regla del vecino más próximo o *Nearest neighbour* (1-NN)

Si tenemos *m* instancias *{e1, ..., em}* en nuestra base de datos, para clasificar un nuevo ejemplo e' se hará lo siguiente:

1. c_{min} = clase (e_1)
2. d_{min} = d (e_1, e')
3. Para *i=2* hasta *m* hacer
   d=d(e_i,e')
   Si ($d<d_{min}$)
   Entonces c_{min} = clase (e_i), d_{min} = d
4. Devolver c_{min} como clasificación de e'

$$d_h(a,b) = \begin{cases} 0, & si\ a = b \\ 1, & si\ a \neq b \end{cases}$$

d(·,·) es una función de distancia. En el caso de variables nominales se utiliza la distancia de *Hamming*.

Distancias para las variables numéricas

Las variables numéricas se suelen normalizar al intervalo [0,1]. Si ei^j es el valor de la variable j en ei, es decir $ei=(ei^1,...,ei^n)$ entonces algunas de las distancias más utilizadas son:

- Euclídea $$d_e(e_1,e_2)=\sqrt{\sum_{i=1}^{n}(e_1^i-e_2^i)}$$
- Manhattan: $$d_m(e_1,e_2)=\sum_{i=1}^{n}\left|e_1^i-e_2^i\right|$$
- Minkowski $$d_m^k(e_1,e_2)=\left(\sum_{i=1}^{n}\left|e_1^i-e_2^i\right|^k\right)^{1/k}$$

Como se puede observar, dm1=dm y dm2=de.

Por tanto, la distancia entre dos instancias *e1* y *e2*, utilizando p. ej. *de*

para las variables numéricas, siendo *i* el índice que se utiliza para recorrer las variables numéricas y *j* el índice que se utiliza para recorrer las variables nominales, sería: $$d_e(e_1,e_2)=\sqrt{\sum_i(e_1^i-e_2^i)^2+\sum_j d_h(e_1^j,e_2^j)}$$

Tratamiento de valores desconocidos: si *ej1* =? Y *ej2*=? Entonces la distancia asociada a la j-ésima componente es la máxima, es decir, 1.
Una crítica a esta regla es que todas las variables se consideran con igual importancia. La solución es asignar pesos a los atributos de forma que se pondere su importancia dentro del contexto

$$d_e(e_1,e_2)=\sqrt{\sum_i w_i\cdot(e_1^i-e_2^i)^2+\sum_j w_j\cdot d_h(e_1^j,e_2^j)}$$

La extensión a la regla del vecino más próximo, es considerar los k vecinos más próximos: k-NN (NN = 1-NN). Su funcionamiento es el siguiente: dado e el ejemplo a clasificar es:

1.Seleccionar los k ejemplos con $K = \{e1,..., ek\}$ tal que, no existe ningún ejemplo e' fuera de K con $d(e,e')<d(e, ej)$, $i=1,...,k$

2. Devolver la clase que más se repite en el conjunto *{clase(e1), ..., clase (ek)}* (la clase mayoritaria)

El algoritmo k-NN es robusto frente al ruido cuando se utilizan valores de k moderados (k>1). Es bastante eficaz, puesto que utiliza varias funciones lineales locales para aproximar la función objetivo.
Es válido para clasificación y para predicción numérica (devolviendo la media o la media ponderada por la distancia).
Es muy ineficiente en memoria ya que hay que almacenar toda la BD.
La distancia entre vecinos podría estar dominada por variables irrelevantes (Selección previa de características).
Su complejidad temporal (para evaluar un ejemplo) es O(dn2) siendo O(d) la complejidad de la distancia utilizada (una forma de reducir esta complejidad es mediante el uso de prototipos).
El algoritmo k-NN está disponible en WEKA bajo el nombre de IBk. Permite voto por mayoría o voto ponderado por la distancia (1/d y 1-d). No permite ponderar las variables.

3.3 Clasificadores basados en redes neuronales

Redes neuronales: Surgieron como un intento de emulación de los sistemas nerviosos biológicos. Actualmente: computación modular distribuida mediante la interconexión de una serie de procesadores (neuronas) elementales.

Ventajas:
- Habitualmente gran tasa de acierto en la predicción
- Son más robustas que los árboles de decisión por los pesos
- Robustez ante la presencia de errores (ruido, outliers,...)
- Gran capacidad de salida: nominal, numérica, vectores,...
- Eficiencia (rapidez) en la evaluación de nuevos casos
- Mejoran su rendimiento mediante aprendizaje y éste puede continuar después de que se haya aplicado al conjunto de entrenamiento

Desventajas:
- Necesitan mucho tiempo para el entrenamiento
- Entrenamiento: gran parte es ensayo y error
- Poca (o ninguna) interpretabilidad del modelo (caja negra)
- Difícil de incorporar conocimiento del dominio
- Los atributos de entrada deben ser numéricos
- Generar reglas a partir de redes neuronales no es inmediato
- Pueden tener problemas de sobreaprendizaje

¿Cuándo utilizarlas?
- Cuando la entrada tiene una dimensión alta
- La salida es un valor discreto o real o un vector de valores
- Posibilidad de datos con ruido
- La forma de la función objetivo es desconocida
- La interpretabilidad de los resultados no es importante

- Las señales que llegan a las dendritas se representan como x1, x2, ..., xn
- Las conexiones sinápticas se representan por unos pesos wj1, wj2, wjn que ponderan (multiplican) a las entradas. Si el peso entre las neuronas j e i es:
  a) positivo, representa una sinapsis excitadora
  b) negativo, representa una sinapsis inhibidora
  c) cero, no hay conexión

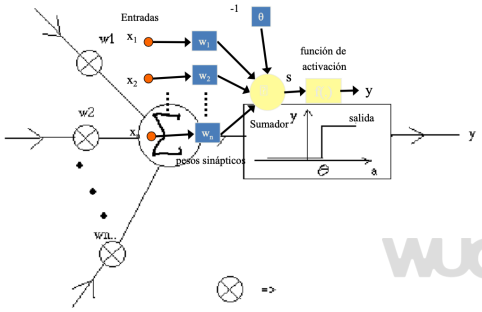
La acción integradora del cuerpo celular (o actividad interna de cada célula) se presenta por:

$$Net_j = w_{j1} \cdot x_1 + w_{j2} \cdot x_2 + \ldots + w_{jn} \cdot x_n = \sum_{i=1}^{n} w_{ji} \cdot x_i$$

La salida de la neurona se representa por yj. Se obtiene mediante una función que, en general, se denomina **función de salida, de transferencia o de activación**. Esta función depende de Netj y de un parámetro ∅j que representa el umbral de activación de la neurona

$$y_j = f(Net_j - \theta_j) = f\left(\sum w_{ji} \cdot x_i - \theta\right)$$

El umbral se puede interpretar como un peso sináptico que se aplica a una entrada que vale siempre -1. Con función de activación o transferencia tipo salto o "escalón"



Función de escalón. Representa una neurona con sólo dos estados de activación: activada (1) y inhibida (0 ó -1)

Función lineal: $y_j = Net_j - \theta_j$

Función lineal a tramos: $y_j = H(Net_j - \theta_j) = \begin{cases} 1, & si\ Net_j >= \theta_j \\ -1, & si\ Net_j < \theta_j \end{cases}$ $y_j = \begin{cases} 1, & si\ Net_j >= \theta_j + a \\ Net_j - \theta_j, & si\ |Net_j - \theta_j| < a \\ -1, & si\ Net_j < \theta_j - a \end{cases}$

Función sigmoidal: $y_j = \frac{1}{1+e^{-\lambda(Net_j-\theta_j)}}$ $y_j = \frac{2}{1+e^{-\lambda(Net_j-\theta_j)}} - 1$

Función base radial: $y_j = e^{-\left(\frac{Net_j-\theta_j}{\sigma}\right)^2}$

En definitiva, los componentes básicos de una red neuronal son:

- Vector de pesos (wij), uno por cada entrada desde a la neurona j
- Un sesgo (Θj) asociado a la neurona
- Los datos de entrada {x1,...,xn} a una neurona Nj se hacen corresponder con un número real utilizando los componentes anteriores
- Una función de activación f. Puede ser muy sencilla (función escalón) aunque normalmente es una función sigmoidal

Resolver un problema de clasificación con una red neuronal implica

- Determinar (habitualmente con conocimiento experto)
  - el número de nodos de salida,
  - el número de nodos de entrada (y los atributos correspondientes,
  - el número de capas ocultas

- Determinar pesos y funciones a utilizar

- Para cada tupla en el conjunto de entrenamiento propagarlo en la red y evaluar la predicción de salida con el resultado actual.
  - Si la predicción es precisa, ajustar los pesos para asegurar que esa predicción tendrá un peso de salida más alto la siguiente vez
  - Si la predicción no es precisa, ajustar los pesos para que la siguiente ocasión se obtenga un valor menor para dicha clase

- Para cada tupla en el conjunto de test propagarla por la red para realizar la clasificación

El aprendizaje de la estructura de una red neuronal requiere experiencia, aunque existen algunas guías

Entradas: Por cada variable numérica o binaria/booleana se pone una neurona de entrada. Por cada variable nominal (con más de dos estados) se pone una neurona de entrada por cada estado posible.

Salidas: Si es para predicción se pone una única neurona de salida por cada valor de la variable clase

Capas ocultas. Hay que indicar cuántas capas y cuantas neuronas hay que poner en cada capa. En Weka,

- 0: No se pone capa oculta (¡sólo particiones lineales!)
- Números enteros separados por comas: cada número representa la cantidad de neuronas de esa capa. P.e. 4,3,5: representa una red con tres capas ocultas de 4, 3 y 5 neuronas respectivamente
- Algunos comodines:
  - i/o: neuronas en la entrada/salida
  - t:i+o

- a: t/2 (es el valor por defecto)

Cuando la red neuronal está entrenada, para clasificar una instancia, se introducen los valores de la misma que corresponden a las variables de los nodos de entrada.
- La salida de cada nodo de salida indica la probabilidad de que la instancia pertenezca a esa clase
- La instancia se asigna a la clase con mayor probabilidad

Todas las variables numéricas se normalizan [-1,1].
Algoritmo de Backprogation o retropropagación:
- Basado en la técnica del gradiente descendiente
- No permite conexiones hacia atrás (retroalimentación) en la red

Esquema del algoritmo de retropropagación
1. Inicializar los pesos y sesgos aleatoriamente
2. Para r=1 hasta número de epoch hacer
   a) Para cada ejemplo e de la BD hacer
   b) Lanzar un proceso forward de propagación en la red neuronal para obtener la salida asociada a e usando las expresiones vistas anteriormente
   c) Almacenar el valor Oj producido en cada neurona Nj
   d) Lanzar un proceso backward para recalcular los pesos y los sesgos asociados a cada neurona

1 epoch = procesamiento de todos los ejemplos de la BD

Algoritmo de retropopagación

1. Inicializar todos los pesos **W** a valores pequeños y fijar un ratio de aprendizaje η a un valor pequeño positivo

2. Seleccionar un ejemplo de entrenamiento (par entrada-salida) y calcular en propagación hacia delante los valores de salida de cada i-ésima neurona de la k-ésima capa
$$s_i^k(t) = \sum_{j=0}^{N_{k-1}} w_{ij}^k(t) \cdot x_j^k(t) \qquad y_i^k(t) = f(s_i^k(t))$$

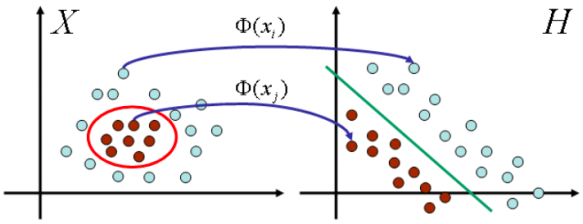
3. Aplicar los valores obtenidos en el paso previo para calcular los errores de la última capa (diferencia entre el valor predicho y el valor esperado)
$$\varepsilon_i^L(t) = \hat{y}_i^L(t) - y_i^L(t)$$

4. Calcular las cantidades de delta de la última capa a partir del error obtenido y de las salidas lineales
$$\delta_i^L(t) = \varepsilon_i^L(t) \cdot f'(s_i^L(t))$$
5. Calcular el valor de las deltas de las capas precedentes mediante retro-propagación de los errores
$$\delta_i^k(t) = \varepsilon_i^k(t) \cdot f'(s_i^k(t)) \qquad \varepsilon_i^k(t) = \sum_{q=1}^{N_{k+1}} \delta_q^{k+1}(t) \cdot w_{qi}^{k+1}(t)$$
6. Actualizar todos los pesos para cada capa
$$w_{ij}^k(t+1) = w_{ij}^k(t) - \eta \cdot \delta_i^k(t) \cdot x_j^k(t)$$
7. Volver al paso 2 y repetir el proceso con un nuevo dato de entrenamiento

3.4 Clasificadores basados en Máquina Soporte Vectorial (SVM)

"Una SVM (support vector machine) es un modelo de aprendizaje que se fundamenta en la Teoría de Aprendizaje Estadístico. La idea básica es encontrar un hiperplano canónico que maximice el margen del conjunto de datos de entrenamiento, esto nos garantiza una buena capacidad de generalización." ➡ **Representación dual de un problema + Funciones Kernel + Teoría de Aprendizaje Estadística + Teoría Optimización Lagrange**

Una **SVM** es un **máquina de aprendizaje lineal** (requiere que los datos sean linealmente separables). Estos métodos explotan la información que proporciona el **producto interno (escalar)** entre los datos disponibles. La idea básica es:



Los **problemas** de esta aproximación es que no sabemos cómo encontrar la función **Φ** y que el espacio de características incluido en H es de alta dimensión. También hay problemas de cómputo y memoria.

Solución: Uso de funciones kernel. La función kernel **[k: X × X ➡ ℝ]** es el producto interno de dos elementos en algún espacio de características inducido (**potencialmente de gran dimensionalidad**). Si usamos una función kernel no hay necesidad de especificar la función **Φ**.

Polinomial: $k(\vec{x_i}, \vec{x_j}) = (\vec{x_i} \cdot \vec{x_j})^d$

Gausiano (RBF): $k(\vec{x_i}, \vec{x_j}) = \exp(-\gamma \|\vec{x_i} - \vec{x_j}\|^2)$ $\quad \gamma = 1/(2\sigma^2)$

Sigmoide: $k(\vec{x_i}, \vec{x_j}) = \tanh(\kappa \vec{x_i} \cdot \vec{x_j} + c)$ $\quad \kappa > 0 \quad c < 0$

Para resolver el problema de optimización planteado se usa el **método de Lagrange**. Cada una de las variables ai es un multiplicador de Lagrange y existe una variable por cada uno de los datos de entrada.

Minimizar $L(w,b) = \frac{1}{2}\langle w, w\rangle - \sum_{i=1}^{l} \alpha_i [y_i(\langle w, x_i\rangle + b) - 1]$

Condicionado a $\alpha_i \geq 0$

Originariamente el modelo de aprendizaje basado en SVMs fue diseñado para problemas de clasificación binaria. La extensión a multi-clases se realiza mediante combinación de clasificadores binarios (se estudia en la siguiente sección, modelos One vs One).

4. Multiclasificadores

4.1. Combinación de clasificadores (*ensemble*)

La idea es inducir *n* clasificadores en lugar de uno solo. Para clasificar se utilizará una combinación de la salida que proporciona cada clasificador. Los clasificadores pueden estar basados en distintas técnicas (p.e.

árboles, reglas, instancias,…). Se puede aplicar sobre el mismo clasificador o con diferentes. Modelos específicos:
Bagging: cada clasificador se induce independientemente incluyendo una fase de diversificación sobre los datos
Boosting: cada clasificador tiene en cuenta los fallos del anterior

4.1.1 Bagging

Bagging ➡ definido a partir de "**b**ootstrap **agg**rega**ting**". Es un método para combinar múltiples predictores/ clasificadores. Bagging funciona bien para algoritmos de aprendizaje "inestables", aquellos para los que un pequeño cambio en el conjunto de entrenamiento puede provocar grandes cambios en la predicción (redes neuronales, árboles de decisión, ...). No es el caso de k-NN que es un algoritmo estable.

Fase 1: Generación de modelos

1. Sea n el número de ejemplos en la BD y m el número de modelos a utilizar

2. Para i=1,...,m hacer
   - Muestrear con reposición n ejemplos de la BD (Bagging)
   - Aprender un modelo con ese conjunto de entrenamiento
   - Almacenarlo en modelos[i]

Fase 2: Clasificación

1. Para i=1,...,m hacer
   - Predecir la clase utilizando modelos[i]

2. Devolver la clase predicha con mayor frecuencia

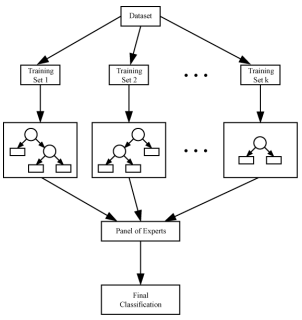
Random Forest (Leo Breiman, Adele Cutler): Boostraping (muestreo con reposición, 66%), Selección aleatoria de un conjunto muy pequeño de variables (m<<M) (ej. log M + 1) para elegir entre ellas el atributo que construye el árbol (medida de Gini), sin poda (combinación de clasificadores débiles).

Algorithm 1 Bagging

Input: S: Training set; T: Number of iterations;
n: Bootstrap size; I: Weak learner

Output: Bagged classifier: $H(x) = sign\left(\sum_{t=1}^{T} h_t(x)\right)$ where $h_t \in [-1, 1]$ are the induced classifiers

1: **for** $t = 1$ to T **do**
2: $S_t \leftarrow$ RandomSampleReplacement(n,S)
3: $h_t \leftarrow$ I(S_t)
4: **end for**



4.1.2 Boosting

Muestreo ponderado (ejemplos): En lugar de hacer un muestreo aleatorio de los datos de entrenamiento, se ponderan las muestras para concentrar el aprendizaje en los ejemplos más difíciles. Intuitivamente, los ejemplos más cercanos a la frontera de decisión son más difíciles de clasificar, y recibirán pesos más altos.

Votos ponderados (clasificadores): En lugar de combinar los clasificadores con el mismo peso en el voto, se usa un voto ponderado. Esta es la regla de combinación para el conjunto de clasificadores débiles. En conjunción con la estrategia de ponderación anterior, produce un clasificador más fuerte.

Los modelos no se construyen independientemente, sino que el modelo i-ésimo está influenciado por los anteriores. La idea es prestar más atención a los ejemplos mal clasificados por los modelos anteriores.

Fase 1: Generación de modelos

1. Asignar a cada ejemplo el mismo peso (1/n)
2. Para i=1,...,m hacer
   - Aprender un modelo a partir de la BD con pesos
   - Almacenarlo en modelos[i]
   - Calcular el error εi sobre el conjunto de ejemplos
   - Si εi=0 o εi>=0.5 terminar
   - Para cada ejemplo bien clasificado multiplicar su peso por εi/(1-εi)
   - Normalizar los pesos de todos los ejemplos

Fase 2: Clasificación

1. Asignar peso cero a todas las categorías de la variable clase
2. Para i=1,...,m hacer
   - Sumar–log(εi/(1-εi))a la categoría predicha por modelos[i]
3. Devolver la categoría con mayor peso

AdaBoost, abreviatura de **"Adaptive Boosting"**, es un algoritmo de aprendizaje meta- algoritmo formulado por Yoav Freund y Robert Schapire que ganó el prestigioso "Premio Gödel" en 2003 por su trabajo. Se puede utilizar en conjunción con muchos otros tipos de algoritmos de aprendizaje para mejorar su rendimiento. La salida de los otros algoritmos de aprendizaje ("algoritmos débiles") se combina en una suma ponderada que representa la salida final del clasificador impulsado.

Algorithm 1: Adaboost Algorithm (Freund and Schapire)

Input : A weak learning algorithm $WeakLearn$, an integer T specifying number of iterations, and N labelled training data $\{(x_1, y_1), \ldots, (x_N, y_N)\}$.

Output : A strong classifier F.

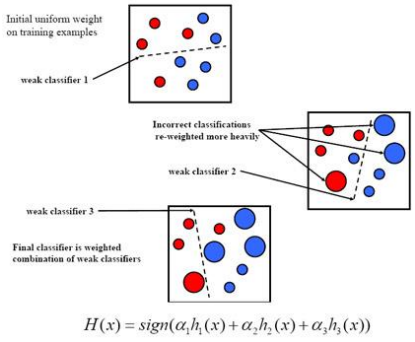
Initialize the weight vector $w_i^1 = \frac{1}{N}$, *for* $i = 1, \ldots, N$.

for $t \leftarrow 1, 2, \ldots, T$ **do**

1. $\mathbf{p}^t \leftarrow \mathbf{w}^t / \sum_{i=1}^{N} w_i^t$.
2. Call $WeakLearn$, providing it with the distribution on $\mathbf{p}^t$; get back a weak learner $h_t : X \rightarrow \pm 1$.
3. Calculate the weight error of h_t: $\epsilon_t = \sum_{i=1}^{N} p_i^t \frac{1}{2} |h_t(x_i) - y_i|$.
4. $\alpha_t \leftarrow \log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$.
5. $w_i^{t+1} \leftarrow w_i^t \exp\left(\alpha_t \frac{1}{2} |h_t(x_i) - y_i|\right)$, for $i = 1, 2, \ldots, T$.

Output the final strong classifier:

$$F(x) = \begin{cases} 1, & \text{if } \sum_{t=1}^{T} \alpha_t h_t(x) \geq 0, \\ -1, & \text{otherwise.} \end{cases}$$



Resumen: Idea General. Bagging and Boosting

Training data ➡ Classifier C
CM (classification method)

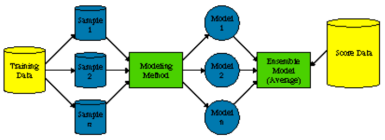
Altered Training data ➡ Classifier C1
CM

Altered Training data ➡ Classifier C2
CM

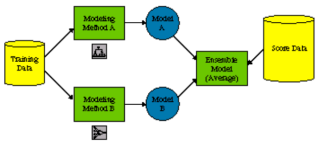
……..

Aggregation… ➡ Classifier C*
CM

Bagging= *Manipulation with data set*



Boosting = *Manipulation with model*



La idea general consiste en crear un pool de clasificadores más grande de lo habitual y luego reducirlo para quedarnos con los que forman el ensemble más preciso. Esta forma de selección de clasificadores se denomina **ensemble pruning.**

Los modelos basados en ordenamiento generalmente parten de añadir 1 clasificador al modelo final. Posteriormente, en cada iteración añaden un nuevo clasificador del *pool* de los no seleccionados en base a una medida establecida. Y generalmente lo hacen hasta llegar a un número de clasificadores establecido. Según este artículo, en Bagging, teniendo un pool de 100 clasificadores, es suficiente con usar 21. Boosting-

based (**BB**): Hace un *boosting a posteriori* (¡muy interesante!). Coge el clasificador que minimiza más el coste respecto a *boosting*, pero no los entrena respecto a esos costes porque los clasificadores ya están en el *pool* .

Bagging: combina múltiples clasificadores.
Boosting: la forma de generar la diversidad de clasificadores es dando ejemplos. Se le vale dando más peso según pasan los ejemplos. Los ejemplos más difíciles de clasificar son los cercanos a la frontera. Genero un clasificador, hago un muestreo ponderado y así sucesivamente.

4.2. Descomposición de Problemas de Multiclase. Binarización (*pairwise learning*)

Descomposición de un problema con múltiples clases:
Estrategia Divide y Vencerás
Multi-clase →Es más fácil resolver problemas binarios. Para cada problema binario tenemos 1 clasificador binario que es igual al clasificador base.

Problema: ¿Cómo hacer la descomposición? ¿Cómo agregar las salidas?

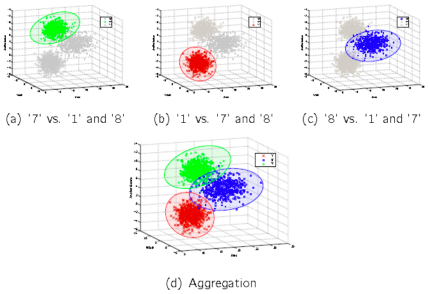
Estrategias de descomposición: "One-vs-One" (OVO)
▪ 1 problema binario para cada par de clases
• Nombres en la literatura: Pairwise Learning, Round Robin, All- vs-All...
• Total = m(m-1) / 2 classificadores
Para cada par de claves construyo m(m-1)/2 clasificadores. Divido el problema en subproblemas.

Pairwise Learning: Combinación de las salidas
Fase de agregación: Se proponen diferentes vías para combinar los clasificadores base y obtener la salida final (clase). Se comienza por una matriz de ratios de clasificación.

$$R = \begin{pmatrix} - & r_{12} & \cdots & r_{1m} \\ r_{21} & - & \cdots & r_{2m} \\ \vdots & & & \vdots \\ r_{m1} & r_{m2} & \cdots & - \end{pmatrix}$$

▪rij =confianza del clasificador en favor de la clase i
▪rji =confianza del clasificador en favor de la clase j
• Usualmente: rji = 1 – rij
▪ Se pueden plantear diferentes formas de combinar estos ratios.

Estrategias de descomposición: "One-vs-All" (OVA)
Hago las parejas, veo las diferentes certezas de cada clase y la que más certeza tenga es la que escojo.
Cogemos las parejas, vemos las certezas y hacemos la agregación.



ECOC (Error Correcting Output Code): funciona tanto para OVO como para OVA. Se asigna la clase que tiene el código más cercano al obtenido. Es un código-Matriz (0, 1, -1) que representa la descomposición. Las salidas forman una palabra clave. Se utiliza un ECOC para decodificar la palabra clave » La clase viene dada por la decodificación.

Ventajas: Clasificadores más pequeños (menor número de instancias). Fronteras de decisión más simples.

Un algoritmo que está en el "estado del arte" en comportamiento y utiliza la estrategia OVO para múltiples clases: SVM (Support Vector Machine).

Conclusiones

Clasificación es el tipo de problema más estudiado en el ámbito de Minería de Datos.
Existen múltiples aproximaciones algorítmicas que presentan buenos resultados y que deben se consideradas para su uso en función de lo que queremos obtener en cuanto a modelos:

- ✓ sistemas basados en reglas (interpretables),
- ✓ prototipos de clasificación (algoritmos basados en instancias),
- ✓ algoritmos de caja negra (no interpretables) cuya finalidad es la aproximación (redes neuronales, SVM ...)

Existen muchos tipos concretos de problemas de clasificación que están siendo objeto de estudio en la actualidad, que dependen de la tipología de los datos.

- Multi-label Classification (MLC)
- Multi-Instance Learning (MIL)
- Semi-supervised Learning (SSL)
- Monotonic Classification
- Label Ranking

Adicionalmente existen muchos otros estudios asociados a la calidad de los datos, escalabilidad, etc. Algunos de ellos son:

- Imperfect Data (noise, missing values ….)
- Data Complexity Analysis of Data
- Big Data Algorithms for Classification

Clasificación no balanceada: Los clasificadores estandar tienden a sobreaprender la clase mayoritaria, ignorando la minoritaria.