Aprendizaje Automático 2016-2017
Grado en Ingeniería Informática
Universidad de Granada

Apuntes de la asignatura

Carlos Manuel Sequí Sánchez

19 de abril de 2017

Índice

Índice de figuras

Índice de tablas

1. Introducción

- nombre del profesor: Nicolas Pérez
- correo electrónico: nicolas@decsai.ugr.es

2. Sistema de evaluación continua

- Trabajos de teoría y prácticas ->65 % de la nota final (3 pruebas)
- Asistencia y participación ->15 %
- Proyecto final ->20 % (hay que sacar un 40 % al menos en los trabajos y prácticas)
- En caso de no reunir los 40 puntos en los trabajos y prácticas, hay un examen de repesca de 20 puntos.

3. Bibliografía

- Learning from data (coger de la biblioteca, los demás están en PDF)

4. Learning from data (Machine learning)

Definición - campo de estudio que le da a un computador la habilidad de aprender sin ser explícitamente programado. Un programa de ordenador se dice que aprende desde la experiencia (E, los datos) respecto a una clase de tareas (T, algo en concreto, por ejemplo, predecir el tiempo) con una medida de rendimiento(P) si logramos que la tarea(T) que queremos realizar mejore(P) con la experiencia(E)

Hay 3 elementos por tanto: (ejemplo con el ajedrez)

- Experiencia (E) ->jugar muchas partidas a dicho juego
- Tarea (T) ->la tarea de jugar al ajedrez
- Medida de rendimiento (P) ->probabilidad con la que el programa ganará la siguiente partida

Si el error decrece (aumenta su rendimiento, es mejor) conforme los datos(E) aumentan, el programa que lo realiza es un programa que "aprende" a partir de datos.
Las tareas consisten en predecir cosas que no están en los datos.

En machine learning a la computadora le damos las entradas del programa y las salidas, de forma que como "salida", esta computadora nos da el programa (reglas que relacionan las entradas con las salidas proporcionadas). Esto es algo distinto a lo tradicional, donde le introducimos el programa y los datos de entrada, y este nos ofrece los datos de salida. El objetivo es predecir con acierto cosas que no están en los datos.

5. Aprendizaje supervisado

Existe un patrón ->No lo conocemos ->Tenemos datos para aprenderlo.
Para "aprender" usamos aproximaciones mediante funciones. En cuanto conocemos un patrón a la hora de resolver un cierto problema, es decir, conocemos COMO resolverlo, ya no es un problema de machine learning. En otra palabras, machine learning se encarga de resolver problemas mediante aproximaciones, pero no nos permite conocer como lo realiza.
Dos paradigmas principales supervisados:

- Regresión: salida = variable continua (**grados** de temperatura, **número** de casas vendidas...en conclusión, números)
- Clasificación: salida = etiqueta sin significación (cara o cruz de una moneda, predecir si un correo electronico ha de ser spam o no....son problemas en los que la salida del problema es un número finito, podría decirse un número real).Resultados de "Sí" o "No".

6. Aprendizaje no supervisado

Consiste en dar al computador tan sólo los datos de entrada para tratar de descubrir patrones, estructuras...para poder pasar el problema a uno de aprendizaje supervisado.En este tipo de aprendizaje no ofrecemos las salidas. Buscamos agrupamientos(clusters) a partir de los cuáles el programa nos dé información interesante sobre esos agrupamientos que antes no conocíamos.
Reducción de dimensionalidad ->para aportar datos significativos a un problema, hemos de tener en cuenta que no podemos añadir cualquier variable para mejorar el rendimiento de los programas(disminuir los errores), para saber que variables(datos) meterle a los computadores usamos la reducción de dimensionalidad, la cual nos ayuda a reducir ese número de variables inicial en un menor número. Esta reducción se basa en realizar combinaciones lineales de las variables de las que partimos. El problema que se presenta es que esas combinaciones no sabremos lo que significan. Gracias a esta reducción del número de variables se pueden conseguir mejores resultados, es decir, un menor porcentaje de error.

En cuanto vemos que en los datos no hay etiquetas, ya sabemos que es un aprendizaje no supervisado. Si decimos que vamos a predecir la altura en función de la edad, el parámetro altura es una etiqueta, mientras que la edad es el supervisor(este es un problema supervisado)

7. Ejemplo del problema de la aprobación de crédito en un banco

- **Input**: es el vector de características, es decir, lo que yo le mido a algo/alguien, los datos que voy a usar para aprender mi función. **Datos que debo de medir**. Lo supondremos dado y, además, supondremos que todos los datos son continuos, número reales. El espacio de donde saco los vectores de características lo denotamos por X.
- **Outputs:** Este siempre será el espacio que me dice cual es la salida. Clase/etiqueta/valor real (dependiendo del tipo de dato).
- **Función objetivo(f)**: función desconocida que ante un valor de X, sabe cual es el valor de Y.
- **Conjunto de datos(D)/Muestra de entrenamiento**: nos lo da alguien para que aprendamos la aproximación de f a partir de ellos.

8. Aprendizaje en acción

H: conjunto de funciones con el que vamos a tratar de aproximar la función f. Puede ser infinito, pero ha de estar definido. La función f ha de ser miembro de ese conjunto H, si no está dentro de esta, no funcionará.

g: es la solución de H que hemos encontrado, la mejor de las propuestas de H que hemos elegido para aproximar f. El **algoritmo de aprendizaje** es el que me permite seleccionar g de H (la mayor parte de los que veremos serán de optimización).

Ante un problema de aprendizaje supervisado están los elementos:

- Vector de características X (las muestras). Estas muestras han de representar de la mejor forma a la población, sino puede haber graves errores. Para ello el muestreo ha de ser uniforme e independiente. Sin esta regla no es posible aprender.
- Espacio Y: definido por el tipo de problema, clasificación o regresión.
- Datos D.

Nuestro papel es el de decidir H(clase de funciones con la que vamos a trabajar) y A (algoritmo a usar para encontrar solución). Esto es lo primero que hemos de fijar antes de comenzar cualquier problema.

9. La hipótesis del perceptrón

A la hora de elegir la H, la parametrizamos con unos valores de peso (W), para definir la forma de las funciones, cada función de H será un vector de pesos. Es decir, una función para nosotros será un vector de pesos. Los pesos son como los coeficientes en una función, teniendo estos pesos ya podemos dibujar la gráfica de esa función (solo habremos de dar valores a las incógnitas). h(x,w) = x1*w1+x2*w2 (damos valores a x1 y x2 en este caso para dibujar la gráfica).

9.1. Regla de asignación

si la sumatoria de las hipótesis tienen valor mayor que el del umbral(threshold) la aprobamos, sino la denegamos. A la hora de transcribirlo matemáticamente, el umbral(b) lo metemos en el vector de incógnitas Wt, que es el que contiene todas las W. En el vector de las x metemos un 1 simplemente para poder realizar el producto escalar con el vector de las incógnitas. Esta clase H nos hace separar los puntos rojos de los azules (en el ejemplo).
es un **algoritmo iterativo**: fija un valor inicial de W, y desde ahí va iterando hasta encontrar el W que separa lo mejor posible.

Algoritmo:

- Fijamos W inicial a 0(es irrelevante el valor inicial), y clasificamos todas las muestras de entrenamiento a partir ese W inicial(damos una vuelta completa a todas las muestras clasificándolas). Es decir, vamos punto por punto y clasificándolos según si están bien colocados o mal colocados con respecto a la linea trazada(recordar que el problema consiste en trazar una linea que separe los puntos rojos de los azules) (vemos donde se equivoca y donde no).
- una vez ha dado una vuelta completa, el algoritmo coge una muestra donde se ha equivocado(al azar) y realiza un movimiento de ese elemento para ver si corrige ese error (aunque ahora se equivoque en una muestra que antes estaba bien). Si el signo es distinto de la etiqueta (es erróneo)(la etiqueta yi, es decir, f(xi) = yi), cambio a un nuevo W, que es la suma del W anterior mas el producto del signo anterior y la muestra actual, es decir: Wnew = Wold + yi*xi. Donde yi es el signo de la muestra actual, xi es la muestra actual.
- Lo hace reiteradamente sobre las muestras del conjunto de entrenamiento hasta que comprueba que todos los elementos están en su sitio.

Tras aplicar el algoritmo no hemos terminado, para finalizar hay que comparar la salida (función g) con la función f, es decir, ver si g se aproxima lo suficientemente bien a f, con unos nuevos datos que me den.

9.2. propiedades del algoritmo

- El algoritmo de aprendizaje itera en los datos pero sin memoria.

- El algoritmo para si el conjunto de datos es separable linealmente (sino no para). Es decir, la propiedad de separabilidad lineal está solo en los datos, no en f; de f no sabemos nada, solo hemos cogido una muestra de esta.
- El algoritmo encuentra una solución óptima usando solo datos.

10. Aprender Vs. Diseñar

Aprender consiste en ajustar a partir de datos, es decir, sin saber nada, sacarlo todo de los datos. En el diseño tenemos el modelo y conseguimos sacar datos a partir de él.

11. ¿Qué es el aprendizaje?: la regla del ERM (minimización de riesgo empírica)

ERM es el primer principio del aprendizaje (algoritmo) que vamos a estudiar. Consiste en tratar de hacerlo lo mejor posible en los datos de entrenamientos(minimizar el error dentro de la muestra) para garantizar que obtenemos lo mejor posible fuera de las muestras de entrenamiento.
Algoritmo: Tratamos de que nuestra H se ajuste lo mejor posible a f en los valores de la muestra. Esta es la regla de aprendizaje ERM.

- En clasificación: contamos las veces que nos equivocamos (yi != h(xi)) y lo dividimos por el número de muestras)
- En regresión: realizamos la resta de los cuadrados de yi y h(xi) y dividimos por el número de muestras

En el ejemplo de las funciones de 1/0, suponiendo una a una, que la función f(sub)i es la buena y comparando con las demás, no nos sirve de nada para aprender de los datos. Por tanto el aprendizaje es inductivo, es decir, cuando se plantea un problema donde nos dan las muestras y las etiquetas. Para saltar esto podemos hacer varias cosas:

- Hacer asunciones sobre f: como la desconocemos realmente esto no es una opción.
- No tratar de afirmar cosas con seguridad a cerca de los sucesos, es decir, vamos a usar probabilidad y aleatoriedad.

Ahora cada uno de los items(datos) que nos lleguen de la muestra no solo son items que me dan, sino también muestras independientes y distribuidas de una distribución de probabilidad P. Ahora sacamos items(x) de ahi y muestreamos.
Consecuencias:

- D: es una variable aleatoria.

- No es realista esperar que todas las muestras que saquemos representen a la población, por lo que habrá veces que la muestra no represente bien a la población y por tanto no aprendamos bien.
- La función g depende de la D. Si la muestra representa bien a la población, por tanto representará bien a f, sino no, por lo que la elección de la función g también dependerá de lo buena que sea la muestra o no. (la función g será bien escogida para la muestra tomada ,sea esta buena o no).
- Novedad: La probabilidad muestra que hay dependencias probabilísticas entre una variable aleatoria y la población en general.

Ejemplo de las bolas rojas y verdes:
Muestra independiente con reemplazamiento (se devuelve cada bola a la urna después de haberla extraido y los sucesos son independiente) : para que haya la misma probabilidad (no hacer trampas)
Con una muestra no podemos decir **con seguridad** que la muestra(unas cuantas bolas rojas y/o verdes) tomada representa a la población entera.
r1(mu) = bolas rojas en muestra
r2(nu) = bolas rojas en urna

11.1. Heffding/Chernoff:

Probabilidad del suceso de que la diferencia entre r2 y r1 sea mayor que un valor e para valores de e mayores que 0 y siendo menor que $2e^{(-2eN)}$ (donde N es el tamaño dela muestra)
Esta probabilidad nos permite saltar sin conocimientos sobre la población, es decir, garantizar **con una probabilidad** X algo sobre la población a partir de una sola muestra. Con infinitos ensayos, esa probabilidad muestra la frecuencia relativa con la que va a suceder lo que pasa en esa muestra. Los resultados que nos da esta probabilidad son resultados **probablemente aproximadamente correctos(PAC)** es decir, aproximamos algo, pero esa aproximación sera probable en un número X de casos.

Hechos fundamentales sobre esto:

- Las muestras son i.i.d (independientes e idénticamente distribuidas)
- El tamaño de la muestra juega un papel relevante para decirme con seguridad lo que ocurre fuera y dentro de la muestra para cualquier distribución de probabilidad. Si N(el tamaño de la muestra) tiende a infinito, entonces el R1 es casi idéntico a R2, no es seguro, pero es muy muy probable.
- El límite ($2e^{(-2eN)}$) no depende de R1 ni del tamaño del contenedor.

11.2. Learning setup

- Tratamos de explicar f mediante h. Comparamos f con h, (colores rojo y verde). Nos preguntamos cual es el error de explicar f tratando con h. El error será:

  E(h) = Px[h(x) != f(x)] (sobre la zona roja de la gráfica).

- Ahora comparamos con el ejemplo de las bolas:

  Sacaremos entonces el error **fuera** de la muestra, donde las bolas rojas es en lo que se distinguen h y f (h(x) != f(x))

  Eout(h) = Px[h(x) != f(x)]

- Segunda parte: muestrear(tomar una muestra y comparar)

  Para distintas muestras comparamos los valores en f y h y vemos donde no coinciden. Ahora la fórmula consiste en:

  Ein(h) = fracción de puntos rojos a la hora de compararlos (error **dentro** de la muestra, puntos en los que no coinciden)

En el ejemplo de las bolas, la bola verde sería que f y h coincide, y la roja es donde no coinciden, r2 es Eout y r1 es Ein. Elijo la h y, para ver como de bien nos irá fuera de la muestra, usamos la desigualdad de Hoeffding. Esta desigualdad nos da la probabilidad de que lo que hay dentro y fuera de la muestra no sea muy distinto. Aprender significa que yo pueda hacerlo muy bien fuera de la muestra, es decir, el objetivo es Eout, conseguir una h para que Eout sea 0. Esta desigualdad nos da la probabilidad de que lo que hay fuera y dentro no es muy distinto.
Conclusión:

- Si consigo elegir un h que dentro de la muestra sea muy pequeño(Ein) y la muestra sea muy grande(N), la desigualdad garantiza que el error fuera de la muestra también se muy pequeño. Si no conseguimos que Ein sea pequeño, es decir, si es grande, la Eout también lo sera, por lo que no serán mayores que epsilon, por lo que no nos sirve de nada, es decir, en cuanto la Eout sea grande, la h elegida no se parece a f.
  Si Ein = 1 (error es total), sabemos que la verdad es la contraria, puesto que el complementario es la solución, es el que mas se aproxima a f, entonces el mayor de los errores es el de 0.5
  Si la clase H es muy grande, hay más probabilidad de que el Ein sea más cercano a 0.

Esto lo hemos hecho para una sola h pero, ahora hemos de ver como lo hacemos para infinitas h y para un número mayor de h's (finito).

12. En el caso finito(varias h's):

haciendo esto para cada una de las hi obtenemos varios Ein (todos distintos dependiendo de la hi). El sentido común nos dice que cojamos el Ein más pequeño, pero nuestro objetivo es que Eout sea 0, y esto no se garantiza cogiendo el Ein más pequeños de entre las hi. ¿Cómo elegimos la buena? La probabilidad crece conforme más h tome, es decir, si h es muy grande tengo casi garantizado que alguna lo hará bien en la muestra (sesgo). Juntamos todas las hi en las que la diferencia entre Ein y Eout sean mayor que epsilon, es decir, me quedo con lo que sea verdad independientemente de la hi elegida, esto es la unión de todos los sucesos malos para todas las hi. Si H es muy grande, el hecho de que conozca Ein no me dice nada sobre Eout. Ahora el N ha de ser mucho más grande para poder garantizar lo mismo que garantizábamos antes con una sola muestra. Interpreting de bound¿?¿?

¿En qué interviene el tamaño de H?

- si |H| es pequeña: Ein es cercano a Eout
- si |H| es grande: Ein es cercano a 0

Para garantizar que Eout será cercano a 0 (que podamos aprender) han d cumplirse 2 condiciones:

- Ein ha de ser cercano a Eout
- Ein ha de ser cercano a 0

¿Qué valor de |H| cojo entonces?
Pues hemos de tomar un valor medio adecuado, un punto donde el error de la salida sea óptimo, un compromiso entre los dos errores para que el valor de Eout sea óptimo.
¿En qué interviene f?

- si f es simple: podemos usar una H pequeña para conseguir Ein cercano a 0
- si f es compleja: podemos usar una H grande para conseguir Ein cercano a 0

12.1. Casos prácticos

Es importante dar vectores de características bastante elaborados para que más tarde, la función de resolución del problema sea más sencilla. En el ejemplo de los números, medimos intensidad(suma de los colores de todos los pixels) y simetría. x = (x0, x1, x2).
Donde x1: intensidad
x2: simetría.
Comenzamos aplicando la hipótesis del peceptrón para separar los puntos azules(unos(creo)) de los rojos(cincos). El perceptrón solo es capaz de parar si es posible separar los puntos, por lo que en este caso nunca parará, pues no es posible separar los puntos rojos de los azules.

Con el perceptrón podemos llevar una cuenta de cuantos puntos están bien y están mal en cada iteración, de esta forma, podemos quedarnos con la mejor de las iteraciones ocurridas en un máximo de iteraciones dado por nosotros(importante esto), es decir, aquella en la que haya menos puntos que estén mal, esta es una solución posible, pero costosa (llamado el algoritmo pocket).

Pasamos ahora a un problema de regresión: cantidad de crédito que proporciona un banco. H sigue siendo un conjunto de funciones lineales de las características. Por cada vector de características me quiero aproximar a un número. La medida de error ya no es me he equivocado/he acertado sino, la proximidad al número que debo obtener (la solución real). Queremos minimizar el error medio de la muestra, para ello usamos mínimos cuadrados. Primer algoritmo para resolver el problema de regresión:
Queremos encontrar el Wt que hace mínima la expresión, teniendo el valor de las etiquetas yn, el número de muestras N y los vectores de características X(es una matriz que contiene los xi vectores de características por filas). La norma de un vector es la suma de los cuadrados de sus elementos.
(Transparencia de Math details:) Descomposición de una matriz en valores singulares: Dada una matriz X cualquiera, esta puede descomponerse en 3 matrices UDV$\hat{t}$, dos ortogonales(U y V) y una diagonal(D). Con esta descomposición ya tenemos lo necesario para realizar (X$\hat{T}$*X)$\hat{}$(-1) y una vez tenemos esto, simplemente hemos de meterlo en la ecuación Wlin = (X$\hat{X}$)$\hat{}$(-1)*X$\hat{T}$*y y de esta manera obtener los coeficientes w.

12.2. Regresión lineal para clasificación

Si usamos regresión en un problema de clasificación y calculamos un w inicial como en un problema de regresión podemos tener una buena solución. Hay ciertos problemas en los que no es posible el uso de funciones lineales para la resolución de estos (el ejemplo de la transparencia "Linearity in data has limitations). La linealidad se mide respecto de los pesos(w), un modelo lineal es lineal en los parámetros (w) no en los datos (x). El espacio de características nos lo dan(los x) pero luego podemos trabajar sobre ellos(expandirlo, es decir, añadir todas las que queramos o transformarlos). El modelo es lineal si al final lo que usamos es un w por las características. Haciendo el cambio(añadiendo o modificando los datos iniciales) comprobamos que el problema ya es linealmente separable. En resumen, se transforman los datos de entrada(eso es lo que indica el símbolo phi), las etiquetas no.

12.3. Medidas de error y datos con ruido

Los errores no tienen la misma importancia en todos los casos reales (huella digital en supermercado o en la CIA), por ello necesitamos que el usuario no diga el peso de los errores, es decir, el peso que tiene cada tipo de error en un mismo problema planteado. Para calcular la función de optimización minimizamos la suma de los errores ponderados por los pesos asignados por el usuario.
Datos ruidosos: nuestro objetivo para los datos ruidosos ahora va a ser una distribución de probabilidad, es decir, dado un X tengo una distribución de probabilidad sobre sus

etiquetas, por lo que las etiquetas pueden tener errores de variabilidad.
Antes teníamos una función f desconocida, ahora sabemos que tiene un ruido, por lo que la f se convierte ahora en una distribución de probabilidad.
Regla de Bayes: si la función de probabilidad es conocida, para cualquier distribución de probabilidad podemos definir la mejor estrategia para realizar decisiones. Nosotros nos centraremos en discriminaciones aproximativas.

13. En el caso infinito

13.1. La dimensión de Vapnik

Hemos de medir la diversidad de H, es decir, cuantas funciones útiles y equivalentes (efectivas) tiene, es decir, las que dan el mismo error a la hora de analizar muestras. Vamos a tratar de ver para un conjunto de muestras la diversidad de H, una vez tengamos esa diversidad, podremos hacer más pequeño |H| en la fórmula, lo que nos dará mayor simplicidad. Para medir la complejidad de H, cogemos una muestra de tamaño fijo (lo más pequeño, 2 puntos por ejemeplo), vemos si con la clase H puedo explicar todas las configuraciones posibles que puedan tener esos 2 puntos. Poco a poco aumentamos el tamaño fijo de la muestra y vamos viendo si H puede separar, así hasta llegar a un número N de muestras en el que no se pueda separar, o que veamos que se va al infinito (que puede separar siempre).
Ejemplo: con 2 puntos, tenemos 4 posibles configuraciones/etiquetados (2 círculos, 2 equis, un círcuclo y una X, una X y un circulo). Con 3 puntos, como vemos en las transparencias, no todas las posibles configuraciones son separables pero, si al menos hay un etiquetado de 3 puntos la cual pueda separarse, entonces ya lo damos por explicado.
Con cuatro puntos, no es posible se explicarlo, pues no es posible separar con una línea todos los posibles etiquetados, por tanto este es el límite del perceptrón. En resumen, hemos de ver, para todos los puntos, sus posibles posiciones y, para cada posición los posibles etiquetados, si para cada posición, hay etiquetados separables, entonces lo metemos como función útil. Definimos una función que define el tamaño del problema (el número de posibles configuraciones que podemos separar): mH(n), donde n es el tamaño de la muestra. $2\hat{N}$ son todos los posibles etiquetados, hemos de contar cuantos de esos posibles etiquetados son útiles con mH(n) (tamaño de funciones de la clase efectivo).
La dimensión de Vapnik: es el tamaño máximo de conjunto que una clase de funciones puede separar totalmente. En el caso del perceptrón por ejemplo es 3, por lo que la dimensión de VC es 3 para el perceptrón. Tamaño más grande para el cual podemos separar todas las posibilidades. Resultado: la función de crecimiento está acotada superiormente por N y la dimensión de VC, es decir, que depende de estas dos cosas. En general en el perceptrón, la dVC es igual a d+1. Ahora en la fórmula sustituimos mH(2n) por la cota que usaremos siempre: $N\hat{}(dVC)+1$ (en las transparencias se usa 2N en lugar de N).
Si tenemos que el tamaño de la dVC es finito y el valor de N (tamaño de la muestra) suficiente, podemos asegurar que el Ein es igual al Eout.
Con un N aproximadamente 10 veces la dVC se podrá hacer...¿?.

Si la complejidad de H es grande (es decir qu hay funciones con mayor grados de libertad, que dependen de más parámetro) entonces se produce una penalización del error. Si quiero sacar la mejor información posible de mis datos debo elegir, de todas las funciones que tengo posibles, la que menor dVC tenga.

Si queremos que Ein baje, tendremos que considerar funciones complejas, lo que hará que dVC suba, lo que en consecuencia penalizará el error (lo incrementará). Si queremos que Eout-Ein baje, tendremos que hacer que dVC baje. SI aproximamos la muestra al tope, dVC será alta, si queremos generalizar, dVC bajará. Hay que buscar un término medio, un compromiso entre ambas cosas.
La verdadera función no tiene por que estar dentro de H, no es necesario, pues solo buscamos aproximar a esa verdadera función f.

modelo = funciones + algoritmo. Cada modelo, hay problemas que no pueden resolver, por lo que no existe el modelo óptimo universal que es capaz de aprender todos los problemas, por lo que el aprendizaje no es algo que pueda resolverse de manera universal. Solo podemos aprender sobre el tipo de clases de dVC finita. Cuando nos enfrentamos a un problema hemos de sesgar nuestra clase hacia esa, para poder aprender. Sesgo inductivo: para poder pasar de la muestra a la población, he de aportar información.
datos test: nuevas muestras para comprobar si he aprendido, es decir, para evaluar la función elegida. Ofrecemos esa muestra a la función elegida, a partir de los datos nos dará las etiquetas y medimos el error Etest. Aplicamos la desigualdad de Hoeffding para terminar con la evaluación, usando el Etest en lugar del Ein esta vez. Los datos test los sacamos de los datos de la muestra, es decir, extraemos unos cuantos datos de la muestra para convertirlos en datos test, lo que hace que N sea más pequeño, por lo que será un poco más grande el Eout; esto no hay manera de evitarlo.

14. Aproximation-Generalization Tradeoff

Usando pérdidas cuadráticas en lugar de perdidas 0/1 (acierto o no acierto) veremos un nuevo punto de vista para elegir la función ganadora en el proceso de aprendizaje. Para ello necesitaremos una nueva regla de inducción, ya que la de VC es ERM, es decir, hacer mínimo el error de la muestra no es útil para todos los casos de aprendizaje. Hemos de conseguir N/dVC $<$20. Hay que elegir un buen compromiso entre el valor de N y la calse (dVC). Minimización del riesgo estructurado(SRM): tenemos una secuencia de clases de funciones cuya dVC es monótona creciente (están contenidas unas en las otras, por ejemplo polinomios de grado 1, polinomios de grado 1 y 2, polinomios de grados 1,2 y 3...). Primero selecconamos la clase, y tras ello, cual es la función dentro de la clase que mejor ajusta. Tenemos un error Ein y una dVC. Estamos eligiendo en que clase la suma de la dimensión(N) y el dVC es lo más pequeña posible. Buscamos minimizar la suma.
Técnicas SRM:

- Fijar un modelo y minimizar el error empírico

- Dejar constante el error empírico y minimizar la dVC

Buscamos minimizar Eout. Elegimos la función que queramos G$\hat{(}$D). Vamos a intentar descomponer el Ex[...].

15. Regresión logística(LGR)

Es un clasificador que nos da las probabilidades de las etiquetas, no como PLA que nos daba las etiquetas (1,-1). Es una función que sea cual sea el rango de valores que tome, está entre 0 y 1. h(x)-probabilidad de una etiqueta, 1-h(x)-probabilidad de la otra etiqueta.

15.1. Medida de error

f(x) = probabilidad de que la etiqueta valga 1
1-f(x) = probabilidad de que sea -1
Ahora para un X sale una etiqueta y, para el siguiente X, se pide la probabilidad para cada una de las etiquetas. Antes, para un X salía una etiqueta y, para el siguiente X se pedía que etiqueta iba a salir.

15.2. Gradient Descendent

Estima el mínimo local más cercano al punto de inicio del algoritmo. Todos los algoritmos que vamos a usar ahora son iterativos (por eso lo del ïnicio del algoritmo"). He de poder calcular el gradiente en todos los puntos de la función. Todas las funcionesen las que pueda aplicar la derivada, puedo aplicarles gradiente descecndiente para hallar un mínimo.

15.3. GD & LDR Algorithms

GD:

Iniciamos los pesos a 0 o valores aleatorios pequeños.

- iteramos
  - 1.Calcular el gradiente
  - 2.Coger la dirección - una vez tenemos el vector gradiente, cambiarle el signo
  - 3.Actualizar los pesos - w(t+1) = w(t) + nvt
  - 4.Iterar otra vez hacia arriba.

En el algoritmo LDR quizás lo más complicado sea la parada, la cual ha de definirse con un número máximo de pasos o por valor pequeño de la función de error en combinación con que el gradiente prácticamente no avance. Si el gradiente está avanzando quiere decir que habrá un óptimo más adelante.

15.4. Gradiente descendiente estocástico

Método que encuentra lo mejores óptimos a la hora de minimizar. Es capaz de saltar mínimos locales. Cada vez que pasamos todas las muestras completas pasamos por una época. Cuando pasamos por una época barajamos las muestras y volvemos a hacer la iteración por todas las muestras, y así hasta terminar el máximo número de iteraciones. Se suelen usar entre 32-128 muestras. Momento: cuando hacemos gradiente descendiente y llegamos a un punto, cambiamos la dirección en ese punto, el momento nos dice que usemos una combinación lineal entre el gradiente y la dirección que traíamos antes, en lugar de utilizar solo el gradiente.

15.5. Método de Newton

Método de optimización que dice que en un número finito de pasos se saca el óptimo (siempre y cuando estemos cerca de él). Este calcula pesos óptimos para decirle al gradiente como tiene que actuar(como ha de avanzar). Suponemos que estamos cerca del óptimo y que por esa zona, la superficie es de forma cuadrática. El problema de este método es que hay que invertir una matriz en cada iteración (de orden n$\hat{3}$, por lo que se retrasa). Aún así, si estamos cerca del óptimo es el método más rápido para terminar.

15.6. Coordenadas descendentes

Cogemos la coordenada inicial, descendemos todo lo que podemos con x1, hacemos lo mismo con x2, con x3 etc.

15.7. SGD: una nueva regla de inducción

Para que lo sea, hemos de convertir los problemas a problemas de minimización. (en esta transparencia se muestra como conectar SGD con las demás técnicas, las equivalencias entre estas)

15.8. NLT-discussion

Cada vez que usamos funciones distintas a la que usamos inicialmente hacemos crecer la dVC. Si exploramos los datos, fijándonos en ellos, también incrementamos dVC de manera exponencial.

16. Modelizacion: el problema y el remedio

Overfitting(sobreajusta): no existen daots sin ruidos, siempre tienen ruido. Esto hace que sea difícil elegir la función de H correcta. Un proceso sobreajusta cuando al elegir un menor error dentro de la muestra (Ein) produce un mayor error fuera de la muestra (Eout). Cuando tenemos que ajustar un modelo a los datos, hemos de ajustarlo olvidándonos de si tiene o no ruido, pero ese .ºlvido"tiene un coste, aun así, esperamos que las muestras de entrenamiento no estén muy alejadas de las muestras del test.

16.1. Validación y selección de modelo.

16.2. selección de modelo

17. Aprendizaje a partir de datos

17.1. Perceptrón multicapa

Perceptrón el cual nace de la mezcla de perceptrones. Veremos si es capaz de calcular más funciones que las que calcula un perceptrón. Se basan en una función XOR (f = (h1*not(h2))+(not(h1)*h2), solo verdadera cuando una única parte de ella lo es. Usando el perceptrón queremos llegar a ver como implementamos esa función. En la transparencia con los grafos del OR y AND, las flechas son los valores de los pesos por los que tengo que multiplicar. h1 y h2 son los dos perceptrones que hemos combinado.

17.2. Redes neuronales artificiales