1. Introducción al Aprendizaje de Máquinas

Facundo Bromberg Ph.D.
Laboratorio DHARMa Dept. de Sistemas de Información
U. Tecnológica Nacional - Facultad Regional Mendoza - Argentina
fbromberg@frm.utn.edu.ar
http://dharma.frm.utn.edu.ar

Agosto 2012

Índice

1. Preliminares

Este documento introduce y motiva temáticas de Aprendizaje de Máquinas: ¿Que problemas resuelve los algoritmos propios del Aprendizaje de Máquinas? ¿Como estos problemas se enmarcan en problemáticas mas generales de la Informática, Ingeniería y otras disciplinas? ¿Que es aprender, y como es que las máquinas aprenden? ¿Que tipos de aprendizaje existen? ¿Como evaluamos algoritmos que aprenden?

El documento se ha construido extrayendo pasajes y/o figuras de las siguientes referencias: Mitchell (1995), Bishop (2006), Nilsson (1998), Duda et al. (2002), y de notas de clases del Dr. Honavar, Honavar (2010).

1.1. Definición: ¿Que es el *Aprendizaje de Máquinas*?

Aprendizaje, al igual que inteligencia, cubre una gama tan amplia de procesos que es dificil definirla de manera precisa. El diccionario provee frases como *Adquirir conocimiento, o entendimiento de, o habilidad en, a través del estudio, instrucción, o experiencia* y *Modificación de una tendencia de comportamiento por medio de la experiencia*. Zoologistas y psicólogos estudian aprendizaje en animales y humanos pero aquí nos enfocamos en el aprendizaje de *máquinas* (**AM**). Así entonces, como muchas de las técnicas de aprendizaje de máquinas derivan del esfuerzo de psicologos de modelar computacionalmente al aprendizaje, se espera que técnicas del AM iluminen ciertos aspectos del aprendizaje biológico.

Con respecto a las máquinas, decimos que aprenden cuando ante algún input de información cambia su estructura, programa o datos de tal manera que su performance futura mejore. Mas formalmente:

> *Un programa M aprende de la experiencia E con respecto a una clase de tareas T y medida de rendimiento R si se producen cambios en el programa M de tal manera de que su rendmiento sobre las tareas T en cierto entorno Z mejora con la experiencia E.*

AM se refiere usualmente a cambios en sistemas que realizan tareas asociadas a la IA tales como: *reconocimiento de patrones, diagnóstico, planificación, control de robots, predicción*, etc., o como ejemplos mas detallados tenemos:

Ejemplo 1:

T : Diágnostico de cancer

E : Conjunto de diagnosticos.

R : Precisión en el diagnóstico de nuevo cancer.

Z : Mediciones ruidosas con errores ocasionales

M : Programa corriendo en una PC.

Ejemplo 2:

T : Anotación de proteínas con etiquetas de su función

E : Dataset con secuencias de proteínas anotadas.

R : Precisión de anotaciones en conjunto de testeo.

Para ser mas precisos, los cambios de un sistema de IA se pueden producir en cualquier parte de su arquitectura, mostrada en detalle en la Fig. 1.

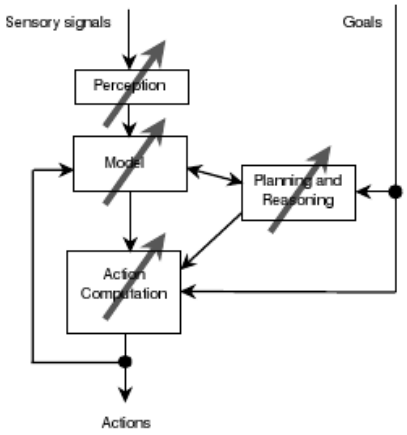

Figura 1: Un sistema de IA. Extraído de Nilsson (1998).

1.2. ¿Porqué deberían aprender las máquinas?

¿Porqué deberían aprender las máquinas? ¿Porque no diseñar directamente máquinas que hagan lo que uno desea? Hay varias razones. Por un lado, como ya adelantamos, comprender el proceso de aprendizaje de humanos y animales. Pero hay razones ingenieriles también:

- Muchas tareas pueden definirse solo por medio de ejemplos. Es decir, es posible especificar pares de input/output pero no la relación en forma concisa (e.g., se sabe que $< 0,0 >, < 1,1 >, < 2,4 >, < 3,9 >, < 4,16 >$, etc., pero no se sabe x^2). Nos gustaría entonces que las máquinas pudieran ajustar su estructura interna para modelar internamente la relación desconocida implicita en los datos de entrada.

- En otros casos, no existe intensión de describir una tarea, sino que simplemente se han generado datos (e.g., bases de datos transaccionales). Aprendizaje de máquinas podría usarse de todas maneras para extraer relaciones o correlaciones implicitas en estos datos (*data mining*).

- Algunas caracteristicas del entorno de trabajo de las máquinas puede no ser conocido en su totalidad al momento de diseñar el programa. AM puede usarse para mejoras del diseño existente, a partir de observaciones del entorno.

- El conocimiento existente sobre ciertas tareas es demasiado como para ser codificado explicitamente. Puede automatizarse la obtención de este conocimiento.

- El entorno es dinámico (cambia con el tiempo). Máquinas que puedan adaptarse a estos cambios evitarian un re-diseño constante (e.g., maquinarias se desgastan, calibración automática).

1.3. Disciplinas afines al Aprendizaje de Máquinas

La siguiente es una lista (no exhaustiva) de disciplinas que sustentan al AM:

Ciencias de la Computación Inteligencia Artificial, Algorítmos y Complejidad, Bases de Datos, y Mineo de Datos.

Estadística Inferencia estadística es el problema de determinar la distribución de la cual cierta muestra fue extraída en base a observaciones de otras muestras. Inferencia estadística puede considerarse un problema de AM, y de hecho muchos algoritmos de AM se basan en técnicas de inferencia estadística. Además, estadística contribuye con técnicas de diseño de experimentos, y análisis exploratorio de datos.

Matemáticas Algebra lineal, Lógica, Teoría de la Información, Teoría de Probabilidades.

Teoría de Control Adaptativo La teoría de control estudia el problema de controlar sistemas con parametros desconocidos, los cuales deben estimarse durante la operación del sistema. Usualmente estos parametros cambian durante la operación y el sistema de control debe adaptar los parametros en base a inputs sensorios.

Psicología y Neurociencia Comportamiento, Percepción, Aprendizaje, Memoria, y principalmente modelos coneccionistas del cerebro llamados redes neuronales. Arboles decisión, redes semánticas y aprendizaje por refuerzo son otros ejemplos de algoritmos de aprendizaje inspirados en modelos psicológicos de aprendizaje.

Modelos Evolutivos En la naturaleza, los animales no solo aprenden a a comportarse mejor en cierto entorno, sino que las especies *evolucionan* para sobrevivir mejor en sus nichos respectivos. Como los ejemplos más prominentes de técnicas que modelan la evolución biológica de especies podemos citar a los *Algoritmos Genéticos* y *Programación Genética*.

Filosofía Ontologías, Epistemología, Filosofía de la Mente, Filosofía de la ciencia.

1.4. ¿Que puede aprenderse?

Ortogonal a las preguntas de ¿porque debería aprender? y las fuentes históricas y disciplinares del aprendizaje de máquinas esta la pregunta de ¿Que puede aprenderse? Ya adelantamos que es lo que podemos aprender es algún tipo de *estructura computacional*. Existen las siguientes posibilidades:

- Funciones
- Programas lógicos y conjuntos de reglas (a.k.a. bases de conocimiento deterministas)
- Bases de conocimiento probabilísticas (e.g., redes Bayesianas)
- Distribuciones de probabilidad en general.
- Máquinas de estado finito
- Gramáticas
- Sistemas resolvedores de problemas

En este curso estudiaremos algoritmos de aprendizaje de funciones y bases de conocimiento probabilístico. En Nilsson (1998) se discuten algoritmos para los otros casos.

2. Aprendizaje de funciones *entrada-salida*

El problema de aprender funciones a partir de datos ha sido resuelto variadas veces por cientificos. Un ejemplo clásico es la ley de caída libre de los cuerpos de Galileo Galilei. A partir de un conjunto finito de mediciones de altitud y tiempo de piedras cayendo de la torre de Pisa, Galileo determina una función $f(t)$ de la altitud en función del tiempo t para cuerpos que caen libremente.

En general, el problema del aprendize es estimar una funcion f desconocida a partir de observaciones del "comportamiento" de la función, es decir, un conjunto de ejemplos de entrada y salida de la función. La hipótesis de cual es la función a aprender la denotamos por h. La entrada de f y h es una entrada multivariada $\mathbf{x} = (x_1, x_2, \ldots, x_j, \ldots, x_n)$ con n componentes. Asumimos a priori que f es parte de un conjunto $\mathcal{H}$ de posibles funciones hipotesis. Algunas veces sabemos con certeza que f efectivamente pertenece a $\mathcal{H}$. Al conjunto de N ejemplos de entrada lo llamamos *conjunto de entrenamiento* y lo denotamos por $\Xi = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$.

2.1. Tipos de aprendizaje

Existen tres tipos de aprendizaje de funciones:

Aprendizaje supervisado Se conoce (alguna veces de forma aproximada) los valores $t_i = f(\mathbf{x}_i)$, $i = 1, \ldots, N$ de f para las N muestras del conjunto de entrenamiento Ξ. La suposición es que si podemos encontrar una hipótesis h cuyos valores se parecen mucho a los de f para los miembros de Ξ entonces h es un buen modelo de f, en especial si Ξ es grande.

Aprendizaje no-supervisado En este caso tenemos como entrada un conjunto de entrenamiento de vectores que no posee los valores de f para ellos. En este casos los problemas pueden ser de dos tipos: *clusterización* o *estimación de probabilidad*. El problema de clusterización es particionar Ξ en subconjuntos $\Xi_1, \ldots, \Xi_R$ de alguna manera apropiada. Aún podemos pensar en este problema como un problema de aprendizaje de funciones, donde el valor de la función es el nombre asignado

al conjunto al cual pertenece el vector de entrada. Un ejemplo de aplicación de este problema es el de categorizar en fonemas los sonidos emitidos por una persona de habla hispana.

Aprendizaje semi-supervisado En cierto sentido el *aprendizaje por refuerzo* puede verse como un problema de aprendizaje de funciones. Aquí el problema consiste en aprender a actuar, es decir, consiste en aprender una función cuya entrada son estados del mundo y la salida son acciones óptimas a realizar. En estos casos el entrenador no conoce el valor exacto de la función (es decir, que acción es la óptima), pero si sabe, a través de lo que se llama *recompenza*, si el estado actual es bueno o malo. El problema de aprendizaje por refuerzo es por lo tanto mas complejo aprender una función que ya que conlleva la problemática de propagar las recompenzas a accions pasadas, es decir, un problema de aprendizaje de secuencias de acciones.

2.2. Vectores de entrada $\mathbf{x}_i$

Debido a que el aprendizaje de máquinas deriva de tantas tradiciones diferentes, su terminología esta plagada de sinónimos, y estaremos usando casi todos ellos en este curso. Asi, al *vector de entrada* suele llamarsele *vector de entrada*, *patrón de entrada*, *muestra*, *ejemplo de entrada*, *instancia*, o *tupla*. En inglés, estos sinónimos corresponden a: *input vector*, *pattern vector*, *feature vector*, *sample*, *example*, *instance*, o *tuple*.

De igual manera, a las componentes x_i del vector de entrada se les llama: *features*, *atributos*, *variables de entrada*, o *componentes*.

Los valores de las componentes (i.e., sus dominios) pueden ser de tres tipos. Pueden ser valores reales, discretos, o categóricos. Un ejemplo de valores categóricos son las características que desciben un alumno: *año de cursado*, *carrera*, y *genero*. Así, un alumno en particular estaría por ejemplo descripto por un vector [cuarto, sistemas, masculino]. Adicionalmente, los valores categóricos pueden estar *ordenados* (e.g., {pequeño, mediano, grande}) o *no-ordenados* (e.g., {sistemas, electrónica, electromecánica}).

En la práctica se asume que las variables del vector de entrada se encuentran ordenadas (de allí que lo llamamos vector). Algunos autores utilizan una representación alternativa de variable-valor, e.g., (año de cursado:cuarto, carrera:sistemas, genero:masculino). Nosotros utilizaremos siempre la forma vectorial.

Un caso especial muy utilizado es el de variables binarias que puede considerarse como un caso de variables discretas $(0, 1)$ o variables categóricas

$(True, False)$.

2.3. Outputs t_i

Los problemas de aprendizaje de funciones puede sub-dividirse en problemas de *clasificación* o problemas de *regresión*, dependiendo de si la salida es categórica o real, respectivamente. En el primer caso se dice que h es un *clasificador* y a la salida se le llama *etiqueta*, *clase*, *categoría*, o *decisión* (en inglés *label*, *class*, *category*, or *decision*), mientras que en el segundo se dice que h es un *estimador de funciones* o *regresor*, y a la salida se le llama *valor de salida* o *estimate* en inglés.

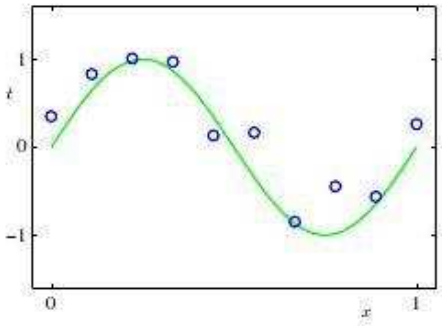

Figura 2: Ejemplo de regresión extraído de (Bishop, 2006). Los circulos (azules) corresponden al conjunto de entrenamiento Ξ de $N = 10$ ejemplos de entrenamiento indicando el valor del vector de entrada x y su correspondiente valor t de entrenamiento de la función. La curva (verde) corresponde a la función $f(x) = sin(2\pi x)$ que se desea estimar.

La Fig. 2 ejemplifica un problema de regresión uni-dimensional (sinusoidal), donde los circulos (azules) corresponden al conjunto de entrenamiento Ξ de $N = 10$ tuplas, y la curva (verde) corresponde a la función a estimar $f(x) = sin(2\pi x)$. La regresión tiene infinididad de aplicaciones, como por ejemplo mapear los movimientos de un brazo robótico, donde la entrada podría ser un vector con información visual del objeto a recojer y la salida el angulo de la articulación.

Los clasificadores tienen aplicación en problemas de reconocimiento, como por ejemplo el reconocimiento de caracteres escritos a mano, donde la entrada es alguna representación adecuada de los caracteres como ser una imagen de $p \times q$ pixeles, y la salida es una de 64 categorias (para mayusculas

y minusculas).

Adicionalmente, vectores de salida multi-dimensionales también son posibles, tanto para componentes reales, categóricas o mixtas (e.g., en el ejemplo del robot, la salida podría ser un vector de los angulos de mas de una articulación).

El caso de salidas binarias ha sido muy estudiado y recibe el nombre especial de *aprendizaje de conceptos*, y al modelo aprendido h se le llama *concepto*.

2.4. Regimenes de entrenamiento

En la práctica se consideran tres posibles regimenes de entrenamiento:

batch En este caso todo el conjunto de entrenamiento Ξ se encuentra disponible y se lo utiliza todo junto para estimar h. Alternativamente puede también mejorarse h iterativamente, utilizando todo el conjunto Ξ en cada iteración para mejorar h.

incremental La hipotesis h es también mejorada iterativamente, pero esta ves extrayendo de Ξ una tupla a la ves. El proceso de selección de tuplas puede ser al azar (con reemplazo) o puede ciclar Ξ iterativamente.

online En muchos casos no existe el conjunto Ξ en su totalidad sino que las tuplas van a apareciendo una a la ves. Algoritmos de aprendizaje que utilizan las muestras de entrenamiento a medida que van apareciendo se les llama métodos *online*. Un ejemplo típico de aprendizaje online ocurre cuando el siguiente ejemplo depende de la hipótesis actual, como es el caso en que se quiere aprender la siguiente acción de un brazo robotico en base a la percepción sensora actual. La siguiente percepción depende de la acción actual.

2.5. Ruido

Algunas veces los vectores de un conjunto de entrenamiento se encuentran corrompidos por ruido. Existen dos tipos de ruido: el *ruido de clase* que afecta a los valores de la función, y el *ruido de atributos* que afecta los valores de las componentes del vector de entrada. En ambos casos sería inapropiado insistir que la hipótesis h se asemeje exactamente a la función f.

2.6. Evalución de Performance

Aunque discutiremos esto en detalle mas adelante, podemos adelantar que el modelo aprendido es generalmente evaluado sobre un conjunto de entrada distinto del de entrenamiento llamado *conjunto de testeo*. Se dice que una hipotesis *generaliza* cuando son buenas sus estimaciones sobre el conjunto de testeo. Error cuadrático medio (e.g., ver Fig. 3) y número total de errores son medidas comunes de performance para regresión y clasificación respectivamente. Cuando el error sobre cierto conjunto de ejemplos es nulo, se dice que la hipotesis es *consistente* con ese conjunto.

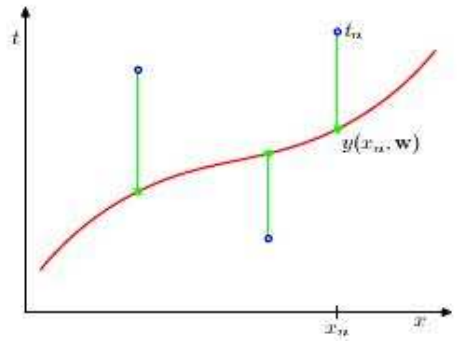

Figura 3: Ejemplo de calculo del error cuadrático medio extraído de (Bishop, 2006)

.

La elección de esta medida de performance se sustenta en la siguiente premisa:

Premisa 1. *Una hipótesis que es consistente con un número suficientemente alto de ejemplos de entrenamiento representativos es probable de clasificar correctamente instancias nuevas extraídas del mismo universo.*

Esta premisa puede no cumplirse en la práctica debida a alguno de los siguientes motivos:

- Conjunto de entrenamiento pequeño,
- Datos ruidosos,
- Datos no representativos.

3. Aprendizaje requiere de *bias*

¿Porque el aprendizaje de funciones es posible? O alternativamente, ¿Como es que por observar algunos ejemplos podemos descartar todos menos uno de los posibles valores de la función? Además, hay un número inconctable de funciones candidatas que coinciden con los 8 ejemplos de la Fig. 2. Por ejemplo tenemos las otras cuatro alternativas mostradas en la Fig. 4. ¿Porque considerar una particular de ellas? ¿Porque asumir que la hipótesis es un polinomio, o una sinusoide? ¿Elegimos una hipotesis consistente (e.g. la de abajo a la derecha de la figura)?

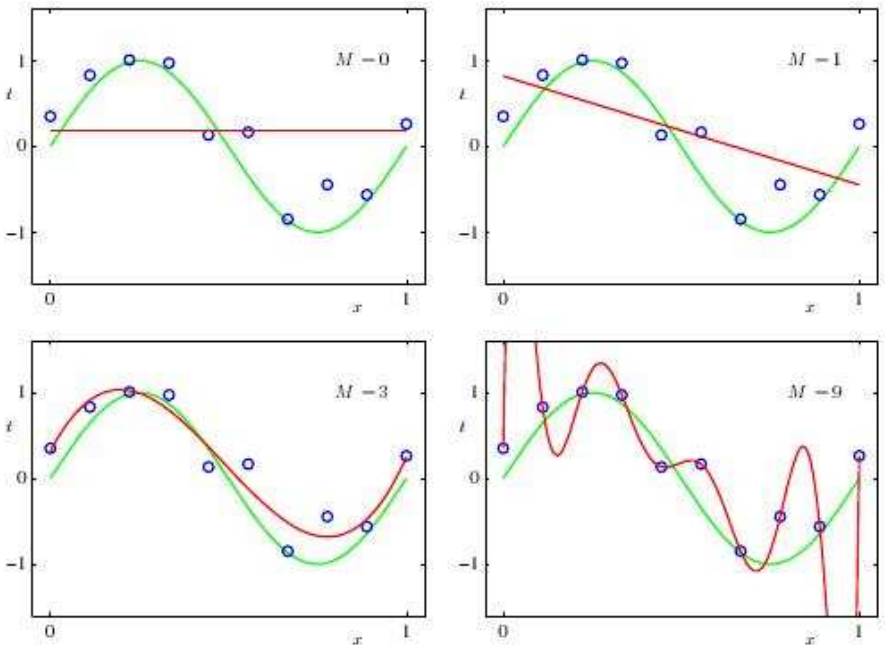

Figura 4: Graficos de polinomios de distintos grados, mostrados como curvas rojas, fiteados al conjunto de entrenamiento de la Fig. 2 (en azul). En verde se muestra el modelo subyacente $f(x) = sin(2\pi x)$ (Bishop, 2006)

.

La respuesta a todas estas preguntas no puede provenir de los datos mismos, y es lo que se llama *bias* en la jerga estadística.

Veamos mas detenidamente porque es que aprendizaje sin bias es *imposible*. Consideremos el caso especial de querer aprender una función Booleana de n variables. Hay 2^n posibles inputs Booleanos. Supongamos que no ten-

emos bias, es decir el espacio de hipótesis $\mathcal{H}$ es el conjunto de *todas* las 2^{2^n} funciones Booleanas y no tenemos preferencia alguna entre todas las consistentes con el conjunto de entrenamiento.

Supongamos que se le presenta un ejemplo de entrenamiento, es decir, una entrada $\mathbf{x}$ y un valor de la función t. De entre todas las funciones Booleanas de $\mathcal{H}$, exactamente la mitad evaluan en 0 para $\mathbf{x}$ y la otra mitad evaluan en 1. Es decir, la mitad entonces serán descartadas como hipotesis candidatas. Esto continua con los ejemplos sucesivos, donde cada uno descarta exactamente la mitad de las hipótesis remanentes. Esto se ilustra en la Fig. 5(Izquierda). Esto indica porque es imposible una generalización en este caso (y por ende el aprendizaje de una hipotesis que tenga una chance de error menor al 50 %): para cada ejemplo no visto, la mitad de las hipotesis restantes predice un valor de 1 y la otra mitad un valor 0. Solamente cuando hayamos visto $2^n - 1$ ejemplos de entrenamiento el espacio de hipotesis consistentes se habrá reducido a uno.

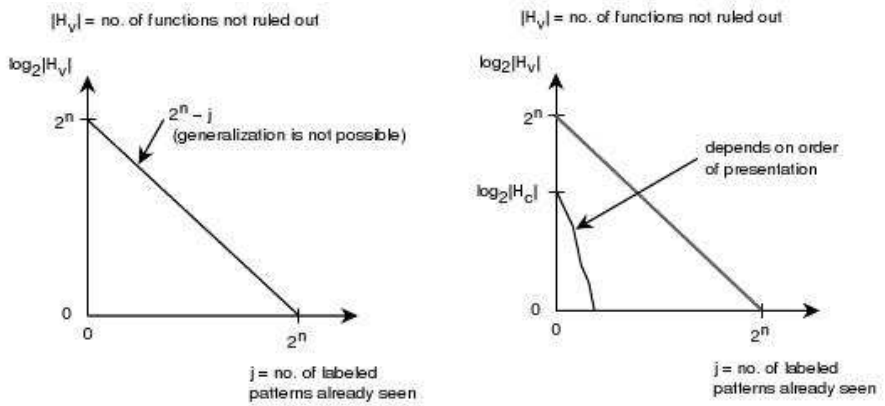

Figura 5: (Izquierda) Hipótesis remanentes como función del tamaño del conjunto de entrenamiento (Derecha) Hipótesis remanentes en un espacio de hipotesis restringido. Extraído de (Nilsson, 1998)

.

Supongamos que nuestro bias nos lleva a reducir nuestro espacio de hipotesis $\mathcal{H}$ a un subconjunto $\mathcal{H}_c$. Dependiendo del subconjunto elegido y del orden en que se presentan los ejemplos de entrenamiento podriamos obtener una curva de hipotesis no descartadas aún podría parecerse a la mostrada en la Fig. 5 (Derecha), es decir, que una sola hipotesis quede consistente con el conjunto de entrenamiento mucho antes de que este crezca a un tamaño

de 2^n tuplas. Mas aún, a pesar de quedar mas de una hipotesis consistente, es posible que para cada tupla no vista aún, la mayoría de las hipotesis remanentes evalúen en el mismo valor (permitiendo una estimación del valor por decidir todas parecido). Esta intuición es formalizada por la teoría del aprendizaje *Probablemente Aproximadamente Correcto* (**PAC**), i.e., que con alta probabilidad el error en la estimación es pequeño. En este curso no estudiaremos esta teoría, aunque remitimos al lector interesado al capítulo 8 de Nilsson (1998) para mas detalles.

3.1. Tipos de bias

El aprendizaje de máquinas distingue dos tipos de bias: *absolute bias* y *preference bias*. Absolute bias, también llamado *bias del espacio de hipotesis restringido*, restringe $\mathcal{H}$. Por ejemplo: polinomios, funciones Booleanas linearmente separables. En *performance bias* (también conocido como *inductive bias*) uno elije la hipotesis minimal respecto a algún ordenamiento. El criterio mas usado es el de minimización de la complejidad (e.g., polinomio de menor grado, grafos con menor numero de aristas), inspirado por el principio de *La navaja de Occam* (*Occam's razor*), introducida por el filosofo Inglés William of Occam en el siglo XIV y dice: "*non sunt multiplicanda entia praeter neccessitatem*", que significa "entidades no deberian ser multiplicadas innecesariamente".

4. Ejemplo detallado: Regresión polinomial

En la práctica, claramente, la función objetivo f es desconocida y lo único que se tiene a mano es el conjunto de entrenamiento $\Xi = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\}$ de N vectores de entrada $\mathbf{x}_i$ etiquetados con los valores de la función $t_i = f(\mathbf{x}_i)$. Para facilitar la presentación de este ejemplo asumiremos sin embargo que f es una función unidimensional conocida $f(x) = sin(2\pi x)$, y que Ξ se ha generado artificialmente sampleando N valores de entrada $\mathbf{X} = (x_1, \ldots, x_N)$ al azar uniformemente de $[0, 1]$ y etiquetandolos con $\mathbf{t} = (t_1, \ldots, t_N)$, donde cada etiqueta corresponde al valor de la función en x_i mas un ruido Gaussiano, i.e., $t_i = f(x_i) + ruido$. La Fig. 2 ilustra estas cantidades.

El conjunto de entrenamiento generado de esta manera captura dos propiedades de conjuntos de datos reales: *poseen una regularidad subyacente* la cual queremos aprender, pero observaciones individuales han sido corrompidas por *ruido*.

Nuestro objetivo es entonces predecir el valor $\hat{t}$ para cierto input $\hat{x} \notin \Xi$. Para ello tomamos la opción de fitear los datos a una curva. Nuestro bias

absoluto restringe el fiteo a curvas polinomiales de grado M:

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j \qquad (1)$$

donde los coeficientes $w_0, \ldots, w_M$ son denotados colectivamente como $\mathbf{w}$. Fitear la curva implica en la práctica estimar estos coeficientes a partir de los datos. Antes de explicar como se estiman estos coeficientes notemos que a pesar de que la función polinomial $y(x, \mathbf{w})$ es no-lineal en x, es *lineal* en los coeficientes $\mathbf{w}$. Las funciones que son lineales con respecto a los parametros desconocidos tienen importantes propiedades y se les llama *modelos lineales*.

Para determinar los valores de los coeficientes fitearemos el polinomio a los datos minimizando una *función de error* que mida, para cierto valor de $\mathbf{w}$, los errores entre $y(x, \mathbf{w})$ y el valor de entrenamiento t:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} [y(x_n, \mathbf{w}) - t_n]^2$$

Vemos así que este error valdrá cero cuando la función $y(x, \mathbf{w})$ pase exactamente a través de cada punto de entrenamiento, por ejemplo, el polinomio de grado $M = 9$ de la Fig. 4 (Abajo a la Derecha).

Fitear la curva puede hacerse entonces eligiendo el valor de $\mathbf{w}$ que minimice el error $E(\mathbf{w})$. Dado que la función error es cuadrática en $\mathbf{w}$, su derivada será lineal, y por lo tanto existiría una sola solución que denotamos $\mathbf{w}^\star$, calculable analíticamente.

4.1. Performance bias y el problema del sobre-fiteo

Hasta aquí hemos llegado con el bias absoluto, quedandonos la importante decisión de decidir el bias de performance de complejidad del modelo, también conocido como el problema de *selección de modelos*, que en este caso se traduce en decidir sobre el grado del polinomio M. La Fig. 4 muestra las curvas $y(x, \mathbf{w}^\star)$ para distintos Ms ($M = 0$, $M = 1$, $M = 3$, y $M = 9$).

Vemos que para $M = 0$ y $M = 1$, los fiteos (en rojo o gris oscuro) son malos, no pareciendose ni a los puntos de entrenamiento (circulos azules) ni a la función f subyacente (verde o gris claro). El tercer caso, $M = 3$, pareciera ofrecer el mejor fiteo de f a pesar de que comete algunos errores con el conjunto de entrenamiento (la curva roja no pasa por encima de los puntos azules). Para $M = 9$ obtenemos un fiteo excelente de los datos de entrenamiento, con un error nulo: $E(\mathbf{w}^\star) = 0$, aunque la curva obtenida

(roja) esta lejos de parecerse a una sinusoide. El comportamiento del modelo para $M = 9$ es un claro ejemplo de ***sobre-fiteo***.

Es en este punto que recordamos al lector que nuestro objetivo es la ***generalización***, es decir, no un error pequeño en ejemplos en el conjunto de *entrenamiento* (como es el caso de $M = 9$), sino un error pequeño en ejemplos nuevos. Para ello generamos un conjunto de *testeo* con ejemplos nuevos o no vistos, de manera similar que el de entrenamiento (valores de entrada x al azar y valores de función t equivalentes a $f(x)$ mas un ruido aditivo Gaussiano). Evaluaremos entonces el error sobre este conjunto de testeo para distintos grados M del polinomio. Por conveniencia reportaremos el *root-mean-square* (RMS) error:

$$E_{RMS} = \sqrt{2E(\mathbf{w}^\star)/N}$$

donde la division por N permite una comparación igualitaria para distintos tamaños de datasets, y la raíz cuadrada asegura que E_{RMS} es medido en la misma escala que los valores de salida t.

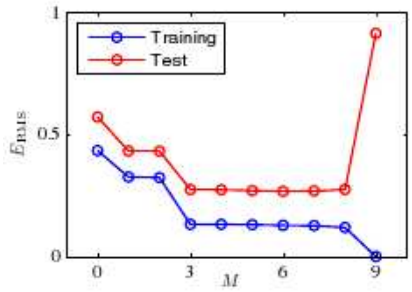

Figura 6: E_{RMS} (root-mean-squared error) evaluado en el conjunto de entrenamiento (circulos azules) y testeo (rojo o gris oscuro) para varios valores del grado del polinomio M. Extraído de (Bishop, 2006)

.

La Fig. 6 presenta el *root-mean-squared* error del conjunto de entrenamiento (circulos azules) y conjunto de testeo (rojo o gris oscuro) para distintos valores de M. El error para el conjunto de testeo mide cuan bien el polinomio $y(x, \mathbf{w}^\star)$ predice los valores de t para observaciones nuevas de x, i.e., que tan bien generaliza.

Podemos observar en la figura que para M chicos el error es grande. Esto puede explicarse por el hecho de que polinomios de grado pequeño son inflexibles e incapaces de capturar las oscilaciones en la función $sin(2\pi x)$.

	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

Figura 7: Tabla de coeficientes $\mathbf{w}^\star$ para polinomios de distintos ordenes. Extraído de (Bishop, 2006)

.

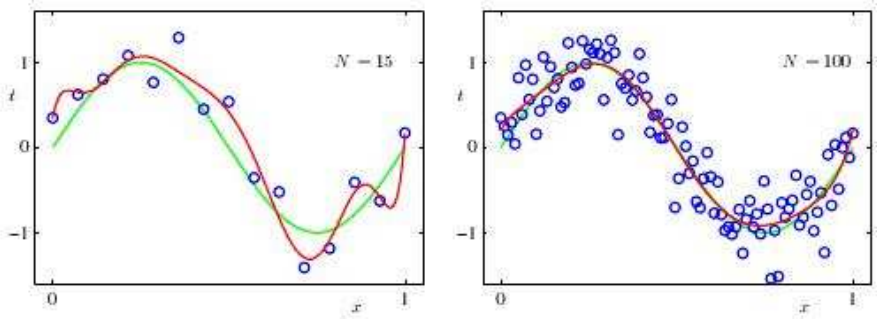

Figura 8: Fiteo de conjuntos de entrenamiento con $N = 15$ (izquierda) y $N = 100$ (derecha) tuplas. Es posible ver como el problema del sobre-fiteo (oscilaciones espureas) mejora para N grandes. Extraído de (Bishop, 2006)

.

Valores de M en el rango $3 \leq M \leq 8$ dan errores pequeños en el conjunto de testeo además de resultar en buenas representaciones de la función generadora $sin(2\pi x)$, tal como como puede verse para $M = 3$ en la Fig. 4.

Finalmente, para $M = 9$ el error se vá a cero para el conjunto de entrenamiento, lo que es esperable ya que es sabido que un polinomio de grado 9 tiene 10 grados de libertad, con lo que puede fitearse para pasar por cualquier conjunto de 10 puntos. Como contrapartida, el error en el conjunto de testeo es muy alto, y tal como lo muestra la Fig. 4 para $M = 9$ el polinomio aprendido es extremadamente distinto a la sinusoide.

A pesar de estos insights, seguimos con la dificultad de comprender intuitivamente porque empeora el error de testeo para Ms grandes, mas cuando

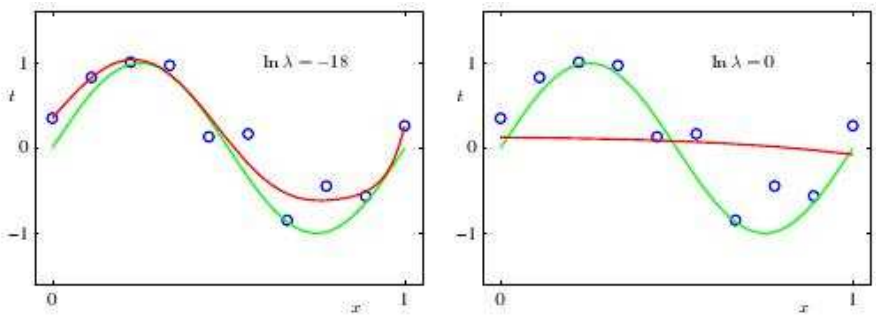

Figura 9: Fiteos regularizados con $\ln \lambda = -18$ y $\ln \lambda = 0$ y $M = 9$. Extraído de (Bishop, 2006)

.

la función a predecir es una sinusoide, cuya expansión en series contiene terminos de todos los ordenes. La explicación corta es que ***se esta fiteando el ruido***. La explicación corta es que ***se esta fiteando el ruido***.

Podemos comprender en mas detalle lo que esta ocurriendo estudiando los valores de los coeficientes $\mathbf{w}^\star$ aprendidos para distintos Ms, los cuales se muestran en la tabla de la Fig. 7. La figura muestra como para $M = 9$ los coeficientes han resultado finamente ajustados a los datos (ruidosos) con valores positivos y negativos extremadamente altos, de tal manera de que el polinomio resultante coincida exactamente con los puntos de entrada (y presente oscilaciones enormes entre estos puntos).

Debido a cancelaciones mutuas de los ruidos, es esperable que datasets mas grandes sean menos propensos a sobre-fiteo. Demostramos esto con los resultados de la Fig. 8. Esta figura muestra que el fiteo mejora para dos casos con creciente número de tuplas, $N = 15$ y $N = 100$, e igual complejidad (i.e., $M = 9$ en ambos).

Lo visto hasta ahora indica que debemos mantener el modelo *lo más simple posible*, pero ¿cuan simple? La *regularización* es una técnica que resuelve esta dificultad aceptando modelos complejos, pero penalizando (*regularizando*) la función de error de tal manera de desincentivar coeficientes grandes. La penalización más simple posible es una suma de los coeficientes al cuadrado:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}[y(x_n, \mathbf{w}) - t_n]^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

donde $\|\mathbf{w}\|^2 = \mathbf{w}^T\mathbf{w} = w_o^2 + w_1^2 + \ldots + w_M^2$, y el coeficiente λ gobierna la importancia relativa del término de la regularización y el error cuadrático medio.

¿Como encontramos el valor de λ entonces? Al igual que antes, minimizamos el error con respecto a todos los coeficientes: $\mathbf{w}$ y λ. La Fig. 9 muestra fiteos de polinomios de grado $M = 9$ regularizados con $\ln\lambda = -18$ (izquierda) y $\ln\lambda = 0$ (derecha). Vemos que para $\ln\lambda = -18$ el sobre-fiteo ha sido eliminado, aunque aumentando demasiado el valor de λ (i.e., penalizando demasiado el error) obtenemos nuevamente un mal fiteo.

	$\ln\lambda = -\infty$	$\ln\lambda = -18$	$\ln\lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

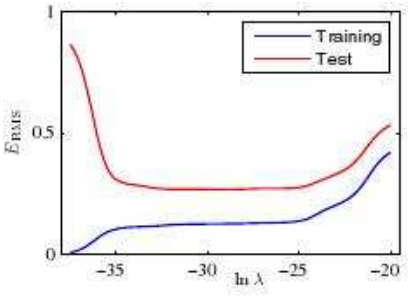

Figura 10: **Izquierda**: Tabla de coeficientes $\mathbf{w}^\star$ para $M = 9$ para regularizaciones con distintos pesos λ. **Derecha**: *root-mean-squared* error versus $\ln\lambda$ para el polinomio $M = 9$. Extraído de (Bishop, 2006)

.

En la tabla de la Fig. 10 (izquierda) mostramos los coeficientes obtenidos para distintos valores de λ ($\ln\lambda = -\infty, \ln\lambda = -18$ y $\ln\lambda = 0$). Vemos que la regularización ha tenido el efecto deseado de reducir la magnitud de los coeficientes.

Por último, veamos lo que mas nos interesa: el impacto de la regularización en la *generalización*. Esto puede verse en la Fig. 10 (derecha) que grafica el E_{RMS} del conjunto de entrenamiento (circulos azules) y conjunto de testeo (rojo o gris oscuro) en función de $\ln\lambda$, para $M = 9$. Vemos que nuevamente λ controla el sobre-fiteo produciendo errores de testeo similares a los de entrenamiento para valores adecuados de λ ($-34 \leq \ln\lambda \leq -25$).

Con esto concluimos esta introducción en donde hemos ilustrado el proceso de aprendizaje y sus intricancias por medio del proceso (simplista) de fiteo de curvas.

Referencias

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer Verlag.

Duda, R. O., Hart, P. E., and Stork, D. G. (2002). *Pattern Classification*. Wiley and Sons.

Honavar, V. (2010). Vasant honavar's webpage. `http://www.cs.iastate.edu/~honavar/`.

Mitchell, T. M. (1995). *Machine Learning*. McGraw-Hill.

Nilsson, N. J. (1998). Introduction to machine learning. `http://robotics.stanford.edu/people/nilsson/mlbook.html`.